Liquid Diagrams

A Suite of Visual Information Gadgets

Martin Lessacher

Liquid Diagrams

A Suite of Visual Information Gadgets

Master's Thesis

at

Graz University of Technology

submitted by

Martin Lessacher

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

11th October 2010

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

Liquid Diagrams

Eine Sammlung von Visualisierungsapplikationen

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Martin Lessacher

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

11. Oktober 2010

Diese Arbeit ist in englischer Sprache verfasst.

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

Abstract

Until now, information visualisation was largely used by academics and research companies, but is becoming increasingly popular among a more general audience of computer users with a need or desire to visualise their own data.

This thesis surveys existing online visualisation solutions and the technologies used to create them. The thesis then describes Liquid Diagrams, an information visualisation framework written in Flex, which enables information visualisation gadgets to be associated with user data and visualisations to be created within the web browser. A suite of nine visualisations is currently implemented: line chart, bar chart, pie chart, parallel coordinates, area chart, star plot, treemap, heatmap (choropleth map), and voronoi treemap.

User data stored in a Google spreadsheet (or otherwise encoded within a web page) is used as the data source and is interactively visualised. Users can configure aspects of each visualisation and can then export high-quality raster graphic (PNG) and vector graphic (SVG) images of their visualisations.

Kurzfassung

Während die Visualisierung von Daten früher vorwiegend von Unternehmen und in der Forschung eingesetzt wurde, findet sie heutzutage immer häufiger im Alltag von Personen Anwendung, welche ihre persönlichen Daten visualisieren wollen.

Diese Arbeit verschafft einen Überblick über die derzeit verfügbaren Online-Softwarelösungen und ihre zugrundeliegenden Technologien. Außerdem beschreibt die Arbeit ein Flex Framework namens Liquid Diagrams, welches die Erstellung von auf eigenen Daten basierenden Visualisierungs-Gadgets im Web-Browser ermöglicht. Dabei stellt Liquid Diagrams neun verschiedene Visualisierungen zur Verfügung: Linien Diagramm, Balken Diagramm, Kreis Diagramm, Parallel Coordinates, Flächen Diagramm, Star Plot, Treemap, Heatmap (Choropleth Map) und Voronoi Treemap.

Als Daten zur Erstellung von interaktiven Visualisierungen dienen Daten, welche in Google Spreadsheet enthalten oder auf Webseiten generiert werden. Dabei stehen dem Anwender viele Optionen zur Gestaltung der Visualisierung und auch der Export als Rastergrafik (PNG) oder Vektorgrafik (SVG) zur Verfügung.

Pledge of Integrity

I hereby certify that the work presented in this thesis is my own, that all work performed by others is appropriately declared and cited, and that no sources other than those listed were used.

Place:	____
Date:	____
Signature:	____

Eidesstattliche Erklärung

Ich versichere ehrenwörtlich, dass ich diese Arbeit selbständig verfasst habe, dass sämtliche Arbeiten von Anderen entsprechend gekennzeichnet und mit Quellenangaben versehen sind, und dass ich keine anderen als die angegebenen Quellen benutzt habe.

Ort:	____
Datum:	____
Unterschrift:	____

Contents

List of Figures

List of Tables

Acknowledgements

I wish to thank my advisor, Keith Andrews for his regular support and feedback during the course of this and the two other projects I worked on under his supervision. I also want to thank him for correcting the draft versions of this thesis.

Furthermore I want to thank my family and friends in supporting me throughout my study, especially my girlfriend.

The most special thanks goes to my father, Friedrich Lessacher, who unremittingly supported me during my years of study and througout my life and thus made this work possible.

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, 2006].
- Figure 3.5 is used with kind permission of Jeff Schiller.

Chapter 1

Introduction

Data and its interpretation are an essential part of many kinds of information work. By interpreting data, organisations and individuals are able to benchmark their efforts, enhance their workflows, and generate new ideas. According to Lyman and Varian [2003] the shear amount of data doubles every three years.

One possibility to facilitate the interpretation of data is to use information visualisation. Information visualisation makes use of the human visual perception system to pre-attentively detect patterns and changes in size, colour, shape, movement or texture [Shneiderman, 1996]. For example, it can be much faster and easier to compare and analyse companies based on several indicators when looking at a graphical representation of the data, rather than looking at a table of data.

Nowadays, information visualisation is increasingly popular and is not only used by companies and academics, but also by individuals in their everyday life. People use visualisations to gain insight into their own data and share the resulting visualisations with other users. The process of sharing leads to social interaction, new knowledge and insights.

This thesis builds upon previous work by the same author. Lessacher [2009a] investigated a range of visualisation solutions available to users and Lessacher [2009b] introduced the Liquid Diagrams framework that enables the creation of highly interactive, user friendly visualisations. This thesis presents the framework and its available visualisations. A summary of the Liquid Diagrams framework was published at IV'10 [Andrews and Lessacher, 2010].

In Chapter 2 an overview of the history of information visualisation is given. The necessary technologies used to implement information visualisation solutions are discussed in Chapter 3. Chapter 4 takes a look at existing visualisation applications and software and points out their strengths and weaknesses. Chapter 6 introduces changes and additions to the Liquid Diagrams framework. It also provides an overview of the existing visualisations in the Liquid Diagrams framework. In Chapter 8 a more detailed description of the implementation of certain special functions and algorithms is given. Finally, Chapter 9 is a brief review of possible future work.

Chapter 2

Information Visualisation

" A picture is worth ten thousand words."

[Frederick R. Barnard (advertisement for Royal Baking Powder, 1921)]

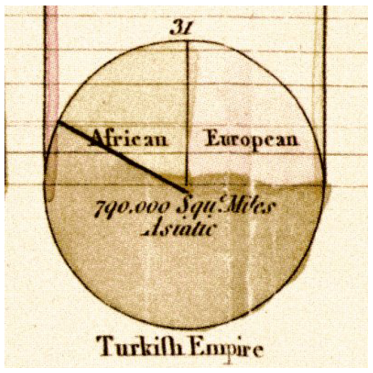
2.1 Origins

As Card et al. [1999] states, one of the earliest attempts to use abstract visual properties to represent data was carried out by William Playfair [1786]. Two of his charts are illustrated in Figure 2.1 and are considered to be the foundation of classical methods for plotting data. In 1914 Brinton [1914], the first American book on graphic techniques for business was published. It illustrated various visualisation types and described how to optimally lay out visualisations. Jacques Bertin [1981] identified the basic elements of diagrams and described a framework for their design and Edward Tufte [1983] led to the development of principles of visual information design. According to Spence [2007] the field of information visualisation greatly advanced over the last 20 years because of computational support. This is due to the possibility to store vast amounts of data, powerful and fast computation, and the availability of high-resolution displays.

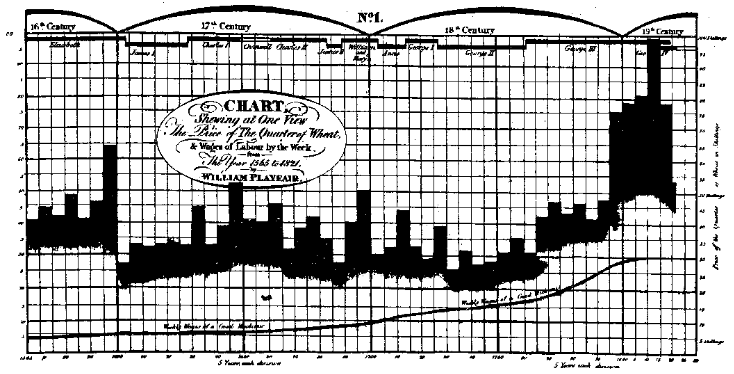
2.2 Definition and Principles

Card et al. [1999, page 7] define information visualisation as "The use of interactive visual representations of abstract, nonphysically based data to amplify cognition". This is illustrated in Table 2.1. According to Spence [2007] information visualisation does not necessarily involve a visual experience due to the fact that sound and other sensory modalities can be employed to represent data. Information visualisation tries to make use of the human visual perception system to pre-attentively detect patterns and changes in size, colour, shape, movement or texture [Shneiderman, 1996]. Ware [2004] states that information visualisation is all about external cognition, due to its boost in the cognitive capabilities of the mind. The sense of vision acquires more information than all other senses combined. If presented well, this sheer quantity of information can be rapidly interpreted by the eye and the brain to detect patterns of differences or changes over time. This also reveals problems and errors in the data set, making visualisations invaluable in quality control. Ware [2004] also states that visualisations are made up of symbols and differentiates between two different types of symbols:

- **Arbitrary Symbols:** Arbitrary symbols have no perceptual basis and thus need to be learned. They are often based on culture and need to be standardised in order to be effective.



(a) The earliest published pie chart by William Playfair in 1801.



(b) A chart using bars to show the price changes of a Quarter of Wheat, and Wages of Labour by the Week, from 1565 to 1821.

Figure 2.1: Some of the first charts, drawn by William Playfair at the beginning of the 19th century (images taken from Wikimedia [2010]).

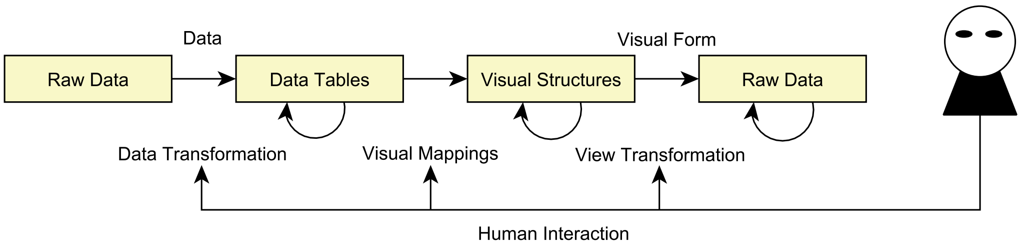


Figure 2.2: The information visualisation reference model shown in Card et al. [1999].

- **Sensory Symbols:** Sensory symbols derive their expressive power without the brain having to learn them. They are well matched to the early stages of neural processing, and,because all humans have more or less the same visual system, they tend to be stable across individuals and cultures. Thus using sensory symbols can produce better displays and better tools for thinking.

According to Ware [2004], human visual perception can be illustrated as a three-tiered information processing model. In the first stage, billions of neurons work in parallel to rapidly extract features (orientation, colours, and textures) from every part of the visual field. Thus, the information is acquired in bursts, a snapshot for each fixation. In the second stage, visual field is divided up into simple patterns and regions, based on the objects textures and colours. In the third stage, objects are held in visual working memory by the demands of active attention which is determined by the preattentive processing.

Ware [2004] states that preattentive processing is probably the most important contribution which vision science can make to data visualisation, because it enables things to be recognised "at a glance". Two factors are important in determining whether something stands out preattentively:

- The degree of difference between the target and non-targets.
- The degree of difference of non-targets from each other.

Unlike scientific visualisation, which is usually based on physical data, information visualisation is based on abstract data. Therefore appropriate visual representations have to be designed [Andrews, 2009, page 1]. One way to compare and design visualisation systems is to use a reference model. The information visualisation reference model was developed by Chi [1999]. The interpretation of this model

Increased Resources	
High-bandwidth hierarchical interaction	The human moving gaze system partitions limited channel capacity so that it combines high spatial resolution and wide aperture in sensing visual environments (Resnikoff, 1987).
Parallel perceptual processing	Some attributes of visualizations can be processed in parallel compared to text, which is aerial.
Offload work from cognitive to perceptual system Expanded	Some cognitive inferences done symbolically can be recoded into inferences done with simple perceptual operations (Larkin and Simon. 1987).
Expanded working memory	Visualizations can expand the working memory available for solving a problem (Norman. 1993).
Expanded storage of information	Visualizations can be used to store massive amounts of information in a quickly accessible form (e.g., maps).
Reduced Search	
Locality of processing	Visualizations group Information used together, reducing search (Larkin and Simon, 1987).
High data density	Visualizations can often represent a large amount of data in a small space Tufte. 1983).
Spatially indexed addressing	By grouping data about an object, visualizations can avoid symbolic labels (Larkin and Simon. 1987).
Enhanced Recognition of Patterns	
Recognition instead of recall	Recognizing information generated by visualization is easier than recalling that information by the user.
Abstraction and aggregation	Visualizations simplify and organize information, supplying higher centers with aggregated forms of information through abstraction and selective omission (Card. Robertson, and Mackinlay, 1991; Resnikoff. 1987).
Visual schemata for organization	Visually organizing data by structural relationships (e.g.. by time) enhances patterns.
Value, relationship, trend	Visualizations can be constructed to enhance patterns at all three levels (Berlin, 1977/1981).
Perceptual Inference	
Visual representations make some problems obvious	Visualizations can support a large number of perceptual inferences that are extremely easy for humans (Larkin and Simon. 1987).
Graphical computations	Visualizations can enable complex specialized graphical computations (Hutchins. 1996).
Perceptual Monitoring	Visualizations can allow for the monitoring of a large number of potential events if the display is organized so that these stand out by appearance or motion.
Manipulate Medium	Unlike static diagrams, visualizations can allow exploration of a space of parameter values and can amplify user operations.

Table 2.1: How information visualisation amplifies cognition (taken from Card et al. [1999]).

by Card et al. [1999] is illustrated in Figure 2.2. First the raw data is transformed into a more structured set of relations which are easier to map. These data tables are then mapped to visual structures. By adding graphical parameters like size or position the visual structures are transformed to views. User interaction is very important during the whole process, because the user determines the output. Interaction support is just as important as the underlying visual representation [Andrews, 2009]. Ben Shneiderman [1996] defined seven basic principles of user interactions in information visualisation systems. The first four principles are known as Shneiderman's "visual information seeking mantra":

- **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 obtain details as 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 query parameters.

2.3 Information Visualisation for the Masses

Until now information visualisation was often only used by academics and research companies to identify outliers, detect patterns or simply to improve performance. Nowadays, information visualisation is growing more popular and thus is more present in everyday life [Kosara, 2007a]. People use it to visualise and share their own data or to look at visualisations made by others, such as NameVoyager [GG, 2010] [Wattenberg, 2005]. This leads to communication and social interaction between people talking about visualisations, leading to newly created knowledge and insights [Viégas, 2010].

The number of applications and web sites providing the ability to share, visualise, and explore data has increased. There are many possibilities to visualise data without necessarily having special training or knowledge.

2.4 Examples of Visualisations

2.4.1 Line Chart

Line charts were introduced by William Playfair [1786] and are most commonly used for time-based data. Each line represents a different entity and is constructed by connecting several data points corresponding to this entity. The characteristic line reveals trends over time, which is why line charts are very popular for financial data.

2.4.2 Pie Chart

First seen in Playfair [1786] the pie chart is among the most simple and most popular type of chart. It consists of a circle composed of several sectors, each representing a different entity. The area of each sector represents its proportional significance. The larger the area of a sector, the larger its significance. The main problem of pie charts is that if its sectors are roughly evenly distributed, it is very hard to distinguish between their significance. There are many variants of pie charts, including for example 3D pie charts and exploded pie charts.

2.4.3 Bar Chart

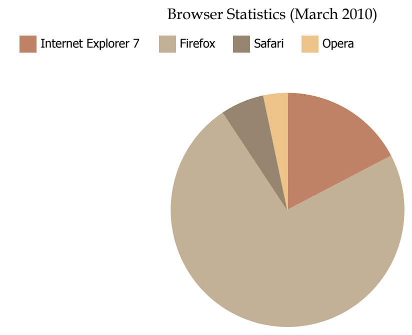
Bar charts also date back to Playfair [1786]. In a bar chart, each entity is represented by one or more horizontal or vertical rectangles. The width or height of each rectangle is determined by its value and indicates the entity's influence. Bar charts are especially useful for comparing two or more values. Figure 2.3 illustrates a data set visualised by the three basic chart types introduced by Playfair.

2.4.4 Area Charts

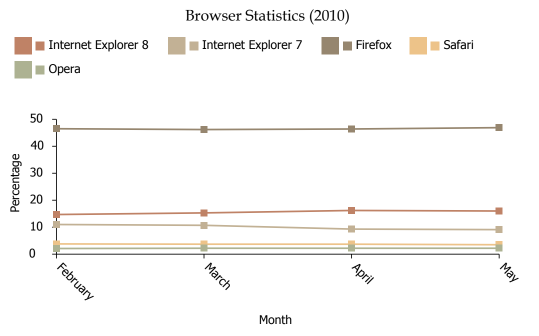
Area charts are very similar to line charts. Like in line charts, the data points of each entity are drawn and then connected by lines to give an entity characteristic line. The main difference is that the area beneath the characteristic line is filled. The resulting areas can be easily compared. The benefit of line charts to reveal trends in the data is maintained. The most common types of area charts are stacked area charts and overlay area charts, both illustrated in Figure 2.4.

	IE 8	IE 7	Firefox	Safari	Opera
February	14,7	11	46,5	3,8	2,1
March	15,3	9,1	46,2	3,7	2,2
April	16,2	9,3	46,4	3,7	2,2
May	16	9,1	46,9	3,5	2,2

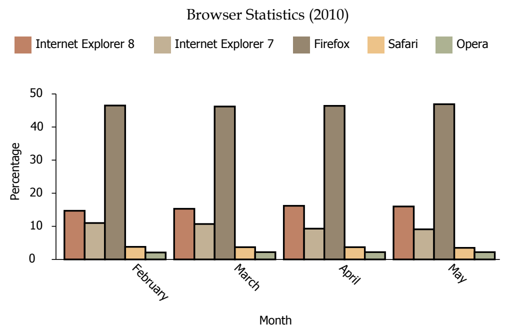
(a) Data table used for three visualisations. The Data shows the browser usage for specific months in 2010 (data taken from Refsnes Data [2010]).



(b) Pie charts are used to compare different entities but lack comparability if two values are nearly the same.

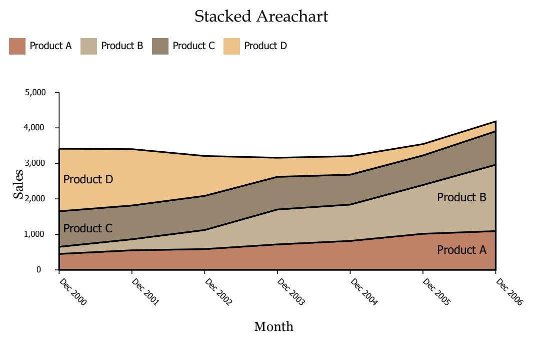


(c) Line charts reveal trends over time.

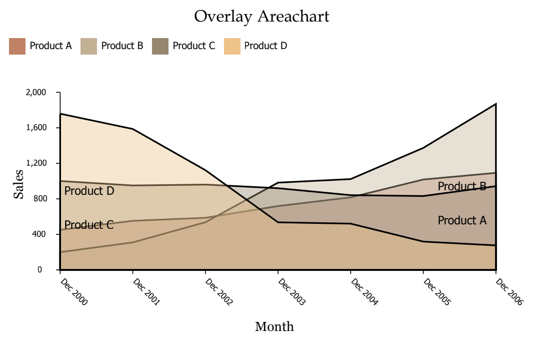


(d) Bar charts are used to compare different entities.

Figure 2.3: Same data visualised by three different visualisation types. All of these visualisation types date back to William Playfair [1786].



(a) Stacked areachart. In stacked area charts, the values are drawn on top of each other. Thus the height of the drawn values at a specific point in time represents the sum of all the entities' values at this specific time.



(b) Overlay areachart. In overlaid area charts each entity's area is drawn as if there were no others, resulting in overlapping diagrams. Each area is drawn semi-transparently, so that underlying areas can still be seen. This variant can be very useful for direct comparison of entities. Too many overlapping entities can result in an overcluttered display.

Figure 2.4: Two different variants of area charts displaying the same data.

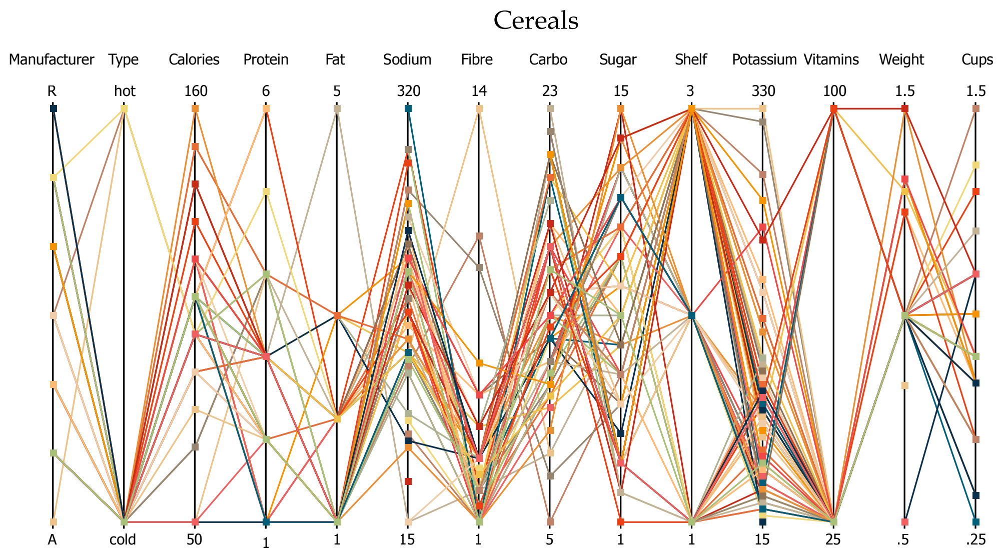


Figure 2.5: The cereals data set shown in this parallel coordinates visualisation has 14 dimensions (resulting in 14 axes and 398 records). Each data entity is shown as a characteristic line of its values.

2.4.5 Parallel Coordinates

The parallel coordinates visualisation was invented in 1885 by Maurice [d'Ocagne, 1885] and later independently rediscovered by Alfred Inselberg in 1959 [Inselberg, 1985] [Inselberg, 2009]. It is a popular and effective way to visualise high-dimensional data. High-dimensional data can be imagined as a number of dimensions or attributes (for example for a car: speed, price, acceleration, ...) and appropriate data entities or records (objects such as individual charts) containing one value for each of these dimensions.

The visualisation is constructed by dividing up the available space into N separate axes, where N is the number of data dimensions. The highest and the lowest value of each dimension's entities are taken as the corresponding upper and lower axis limits. The next step is to take an entity, and draw its values relative to the upper and lower boundary on the corresponding axis. Afterwards the data points are connected by lines, leading to an entity characteristic line or polyline. This line drawing process is then carried out for each entity.

Parallel coordinates are especially useful for detecting patterns and comparing individual data entities. In order to detect patterns it is very helpful to be able to rearrange specific axes. An example of a parallel coordinates visualisation is shown in Figure 2.5.

2.4.6 Scatter Plots, Bubble Chart

According to Friendly [2008] the first semi-graphic scatterplot and correlation diagram was drawn in 1874 by Francis Galton. A scatter plot is a chart where the value of one dimension of a data entity is drawn on the x-axis and another dimension on the y-axis. The point of intersection of these two values is compared to intersection points of other data entities. This way the user is able to detect patterns, for an example look at Figure 2.6.

Like a scatter plot, a bubble chart also uses two data dimensions to obtain a point of intersection.

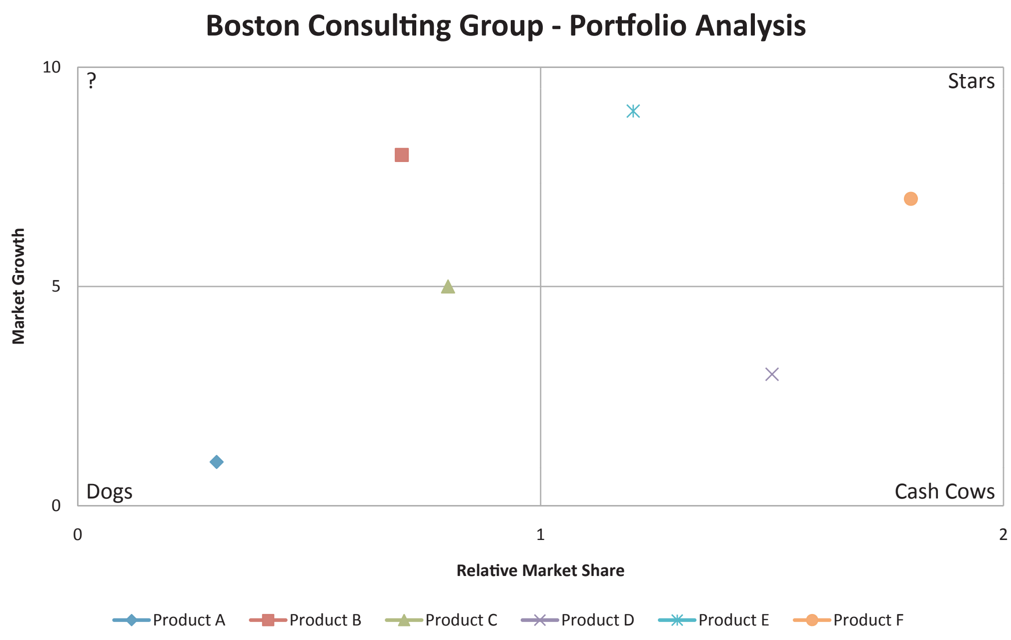


Figure 2.6: This scatter plot displays a portfolio analysis of six different products. Each product is one data entity. The two used dimensions of each entity are its market growth and its relative market share. The point of intersection of both dimensions tells the user in which stage the product is. According to the current stage he can initiate an appropriate strategy.

The difference to scatter plots is that another dimension of the data is drawn into the chart as the size of the intersection point. That way bubble charts contain three different dimensions of one data entity as illustrated in Figure 2.7. The glyph drawn can be varied according to several dimensions: say by size, colour, type. Hence five data dimensions could be represented: two in the x and y axes and three within the glyph.

2.4.7 Star Plot (Radar Chart, Star Chart, Spider Chart)

Star plots date back to Mayr [1877]. Star plots are also called star diagrams or spider charts because they look like a star or spider's web. A radar chart is drawn as n-gon where the radial axes are categories (for example years). Each category typically ranges from the outside (highest value) to the centre (lowest value) of the n-gon and is symbolised by a line giving the chart its spider-web-like look. For each data entity a line is drawn between the corresponding values of each neighbouring category. The result is a star-like shape for each of the data entities which characterises them. This enables comparison between the different data entities, as shown in Figure 2.8.

2.4.8 Tree Map

The treemap visualisation was invented by Ben Shneiderman in 1990 to visualize the space usage of his hard disk in a compact and effective way [Shneiderman, 2008]. A treemap is a visualisation of hierarchical data by using nested rectangles. For each data item in the hierarchy a rectangle is drawn. All the rectangles of one hierarchy level share the space of their common parent's rectangle. The space that each rectangle takes is determined by its proportion of the parent space. This is illustrated in Figure 2.9

Tree maps display data in a very space efficient way resulting in lots of displayable items on the

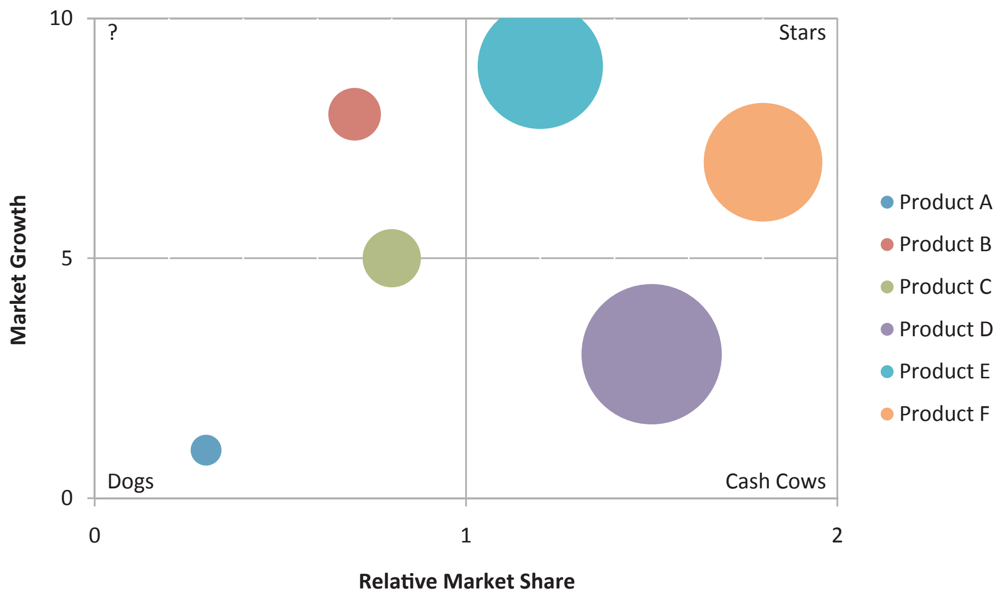


Figure 2.7: The bubble chart looks nearly the same as the scatter plot. The main difference is the size of the points of intersection. The point size is determined by a third dimension of the data entity. In this example the size is given by each products profit values.

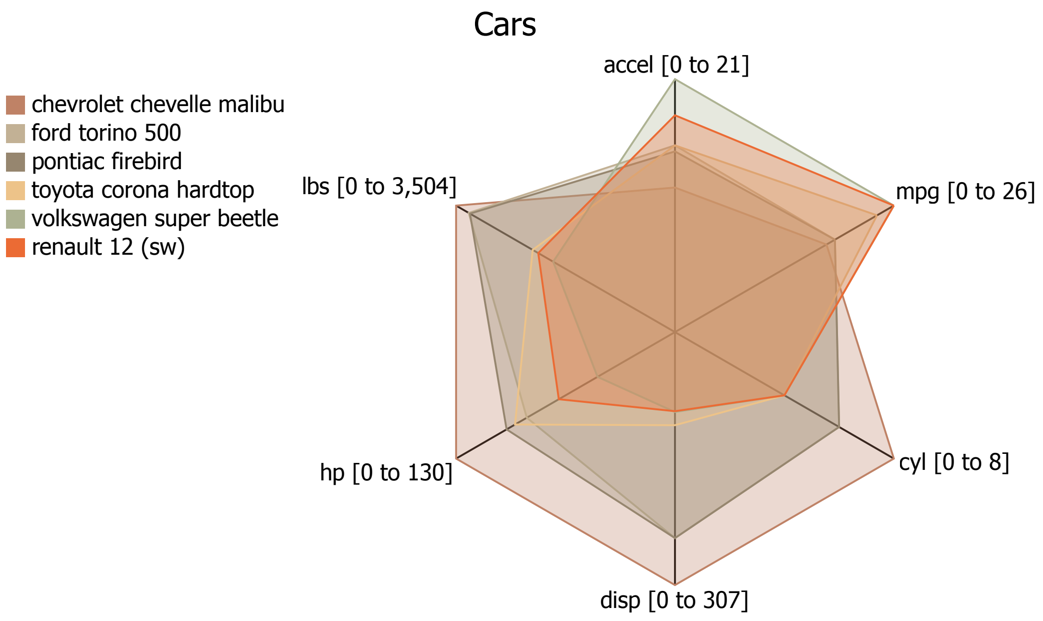


Figure 2.8: The main structure of a radar chart looks like a spider web. In this example six cars of the cars dataset are compared to each other. The connection of all values leads to a star-like and characteristic shape for each car. Due to that shape they can be compared to each other.

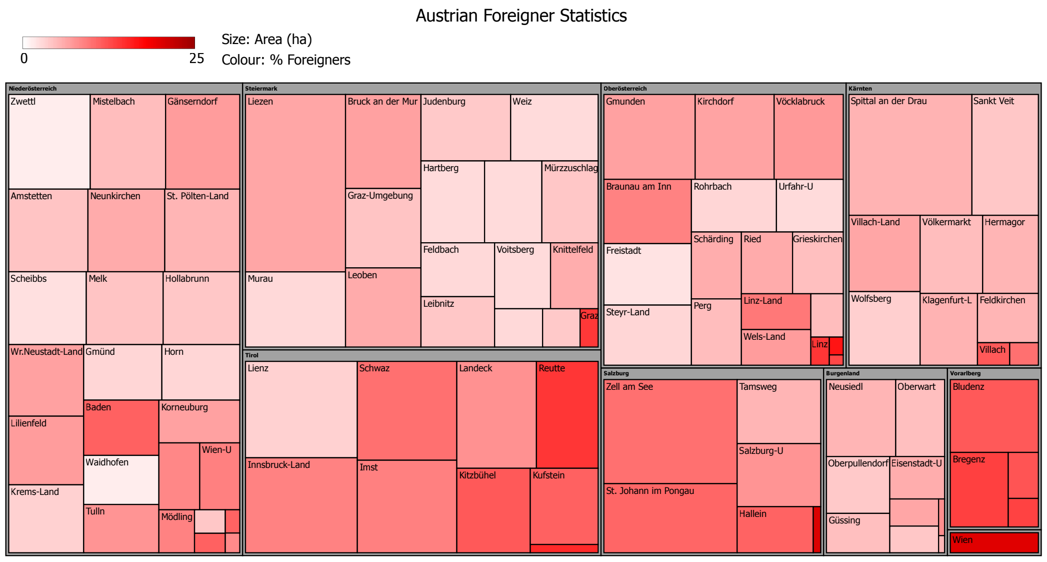


Figure 2.9: A treemap visualisation of the percentage of foreigners in Austrian provinces and districts. Each province takes its share of space according to its area. The colour coding is given by the percentage of foreigners in the district. Vienna in the bottom right corner, has the highest percentage of foreigners.

screen. By looking at the sizes of the rectangles the user is able to immediately spot outliers and by using colour coding he is also able to detect patterns.

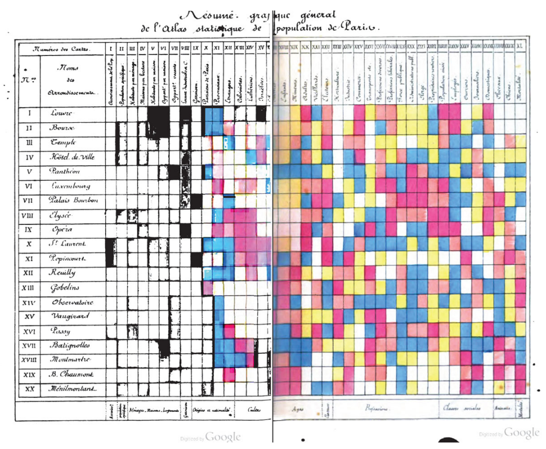
2.4.9 Heat Map (Choropleth Map)

A heatmap is a representation of data in two-dimensional areas. Each data entity is assigned a colour corresponding to its value. The history of heatmaps can be traced back to Loua [1873] who introduced colour shaded matrix display in 1873 (see Figure 2.10a). A typical type of a heatmap is a cluster heatmap shown in Figure 2.10b. A cluster heatmap consists of a rectangular tiling with each tile shaded on a colour scale [Friendly and Wilkinson, 2009]. Heat maps typically find application in web page analysis and in molecular biology.

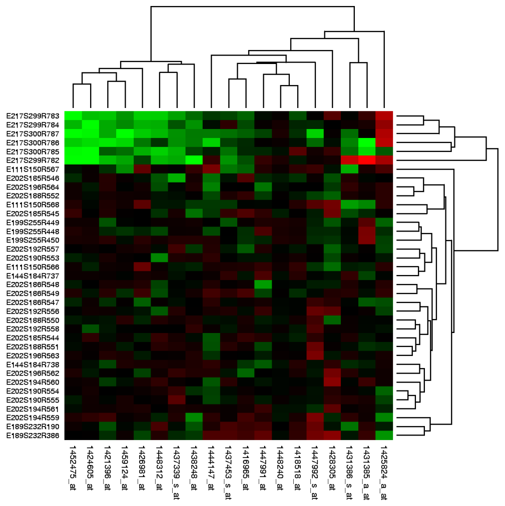
A choropleth map is a heatmap with cartographic areas. Choropleth maps display geographic areas which are shaded to reflect a value assigned to this specific area. By doing so the choropleth maps are designed to enhance the analysis of all kinds of geographical data like economical and statistical facts of countries. The first choropleth map was introduced in 1826 by Dupin [1826] and displayed the distribution and intensity of illiteracy in France (shown in Figure 2.11a). Choropleth maps have been growing more popular the past years.

2.4.10 Voronoi Diagram

According to Okabe et al. [2000] a Voronoi diagram is a concept that has been discovered many times in many different disciplines and thus is known under many different names. A few of these names are Voronoi diagrams (computational geometry), Wigner-Seitz zones (chemistry, physics), domains of action (crystallography), Thiessen polygons (geography), and Blum's transform (biology). A Voronoi tesselation partitions the available space among a number of given sites, according to the nearest-neighbour rule.

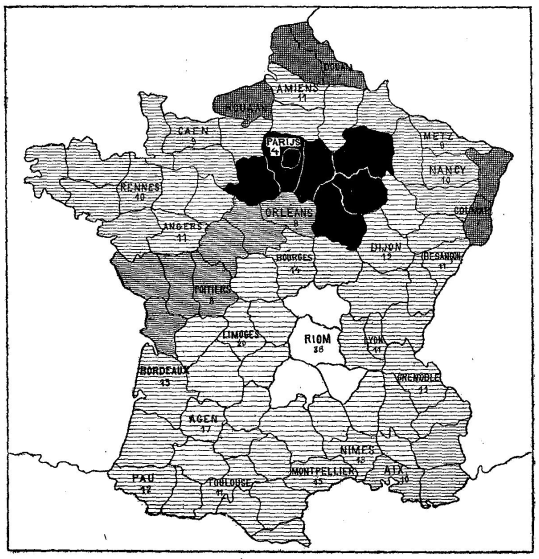


(a) The first colour shaded matix display published in Loua [1873]. This image is taken from Friendly and Denis [2010].

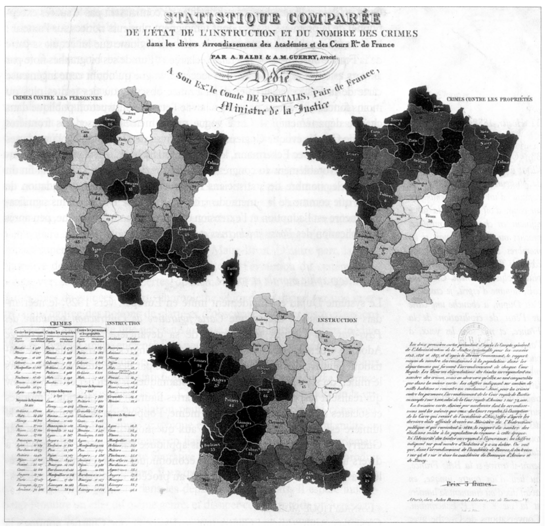


(b) This cluster heatmap displays data extracted from the StemBase database of gene expression data. The image is taken from Wikimedia [2010].

Figure 2.10: This figure shows typical cluster heatmaps which display data in a two dimensional map.

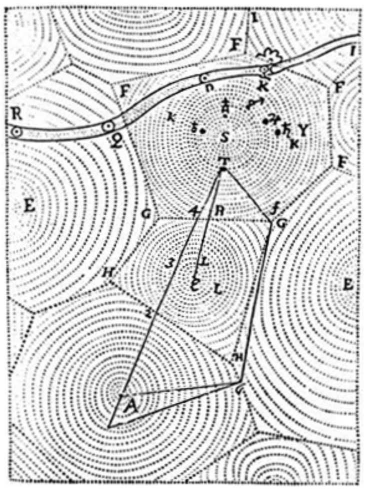


(a) The first choropleth map created by Dupin [1826] in 1826 displays the distribution and intensity of illiteracy in France.

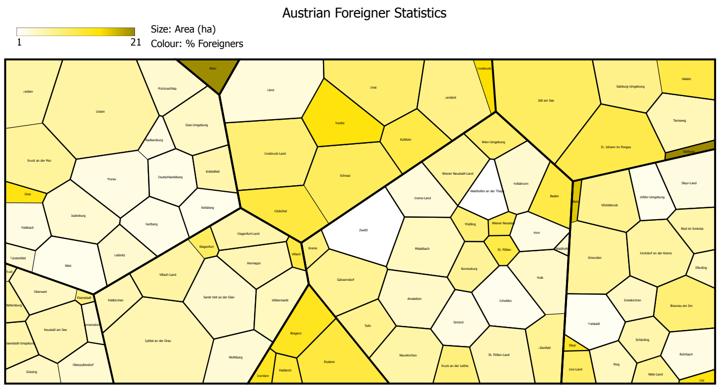

STATISTIQUE COMPARÉE
DE L'ÉTAT DE L'INSTRUCTION ET DU NOMBRE DES CRIMES
dans les divers Arrondissemens des Académies et des Cours Rᵉˢ de France
PAR A. BALBI & A. M. GUERRY
CRIMES CONTRE LES PERSONNES
CRIMES CONTRE LES PROPRIÉTÉS
INSTRUCTION

(b) A thematic choropleth map by Guerry and Balbi [1829] showing crimes against property in relation to level of instruction by departments in France.

Figure 2.11: Early choropleth maps displaying statistics of France and its departments. The images are taken from Friendly and Denis [2010].



(a) The first Voronoi-like diagram written and published by René Descartes. It shows the disposition of matter in the Solar System and its environs (image taken from Okabe et al. [2000])



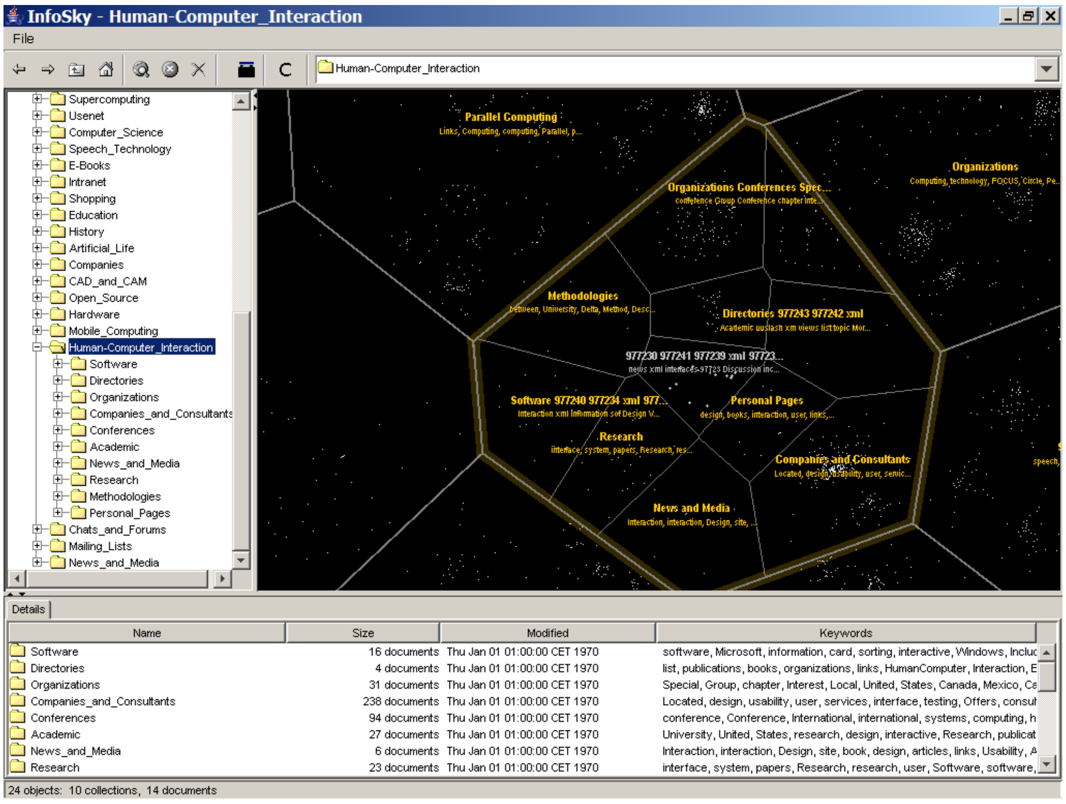
(b) A Voronoi treemap is very similar to a treemap. This Voronoi treemap displays the same data set as the treemap in Figure 2.9 but uses polygons instead of rectangles to build up the diagram.

Figure 2.12: An ordinary Voronoi diagram is different than a Voronoi treemap because in the Voronoi treemap the sites (generators) are weighted and several iterations have to be undergone to reflect the proper area sizes like given by the data set.

The first published Voronoi-like diagram dates back to René [Descartes, 1644] in 1644, which showed the disposition of matter in the solar system and its environs (shown in Figure 2.12a) [Okabe et al., 2000]. According to [Aichholzer and Aurenhammer, 2002] the term Voronoi diagrams originated from the Russian mathematician George Voronoi who is believed to be the first who formally introduced the concept in 1908. In applications Voronoi diagrams are used for collision detection, motion planning, associative file searching, clustering, scheduling, and crystal and cell growth [Telea and van Wijk, 2001].

The first use of Voronoi diagrams in information visualisation was in InfoSky [Andrews et al., 2002]. Infosky enables the exploration of large, hierarchically structured knowledge spaces illustrated in Fiugre 2.13. In InfoSky, area partitioning is done using modified, weighted Voronoi diagrams. The centroids of subcollections are used to partition the polygon representing the parent collection into polygonal sub-areas. The size of each sub-area is related to the total number of documents contained within the corresponding subcollection.

The same idea was later called a Voronoi treemap by Balzer and Deussen [2005]. A Voronoi treemap possesses the advantages of a normal treemap to display hierarchical data. Instead of using rectangles to build up the visualisation the Voronoi treemap uses polygons instead. The benefit of polygons is that the aspect ratio between width and height is not as limited as in treemaps [Balzer et al., 2005]. Another benefit is that any shape like circles, triangles, and other polygonal shapes can be used as the Voronoi maps main shape. An example for a Voronoi treemap can be seen in Figure 2.12b.



Name	Size	Modified	Keywords
Software	16 documents	Thu Jan 01 01:00:00 CET 1970	software, Microsoft, information, card, sorting, interactive, Windows, Incluc
Directories	4 documents	Thu Jan 01 01:00:00 CET 1970	list, publications, books, organizations, links, HumanComputer, Interaction, E
Organizations	31 documents	Thu Jan 01 01:00:00 CET 1970	Special, Group, chapter, Interest, Local, United, States, Canada, Mexico, Ca
Companies_and_Consultants	238 documents	Thu Jan 01 01:00:00 CET 1970	Located, design, usability, user, services, interface, testing, Offers, consul
Conferences	94 documents	Thu Jan 01 01:00:00 CET 1970	conference, Conference, International, international, systems, computing, h
Academic	27 documents	Thu Jan 01 01:00:00 CET 1970	University, United, States, research, design, interactive, Research, publicat
News_and_Media	6 documents	Thu Jan 01 01:00:00 CET 1970	Interaction, interaction, Design, site, book, design, articles, links, Usability, A
Research	23 documents	Thu Jan 01 01:00:00 CET 1970	interface, system, papers, Research, research, user, Software, software,

24 objects: 10 collections, 14 documents

Figure 2.13: The interface of InfoSky, an application which enables the exploration of large, hierarchically structured knowledge spaces by using modified, weighted Voronoi diagrams.

Chapter 3

Technologies

This chapter gives an overview of existing technologies which enable the visualisation of data. This includes both older technologies and more recent technologies.

3.1 HTML, JavaScript, DOM and AJAX

The Hypertext Markup Language (HTML) is the original markup language for web pages. Its first specification dates back to the year 1991. HTML uses a limited set of predefined tags to describe the content of a web page. Since HTML was made to generate static content, it is rather unsuitable for visualising data. Except for animated GIF images pure HTML supports no animations. More significantly, HTML alone does not support any user interaction beyond a mouse click inside an image map.

JavaScript is a scripting language which can extend HTML to tackle one of its limitations — the inability to support user interactions. JavaScript code is therefore into the HTML code allowing it to register and handle user-triggered events. It is a client-side script and therefore executes within the web browser on the computer of the user. Using JavaScript it is possible to access specific elements of the web page, in particular form fields.

To gain access to all elements of a page, the document object model (DOM) is required [W3C, 2010a]. The DOM describes the elements of a web page. Using JavaScript it is possible to access specific elements of the document and manipulate them, resulting in dynamic changes to the displayed content. This extended web page is then called dynamic HTML, because it is capable of changing its content in response to user input.

Although JavaScript and the DOM greatly enhance the functionality and capability of HTML, there is still one downside. Each time the client-side content changes and new data from the server is needed, a new HTTP request has to be sent to the server, which creates a new response page and sends the whole page back to the client. This can lead to a large overhead of unnecessary data and to high client-side loading times. Asynchronous JavaScript and XML (Ajax) is the solution to this problem. It allows the client side to request only specific data and thus avoid a complete page refresh, as illustrated in Figure 3.1.

Using HTML in combination with JavaScript and AJAX, it is possible to implement interactive user-friendly visualisations for the web. These technologies can be used by everyone without having to first install special software. The user only has to have a JavaScript enabled browser.

However, there are some drawbacks too. JavaScript was originally developed by Netscape for its

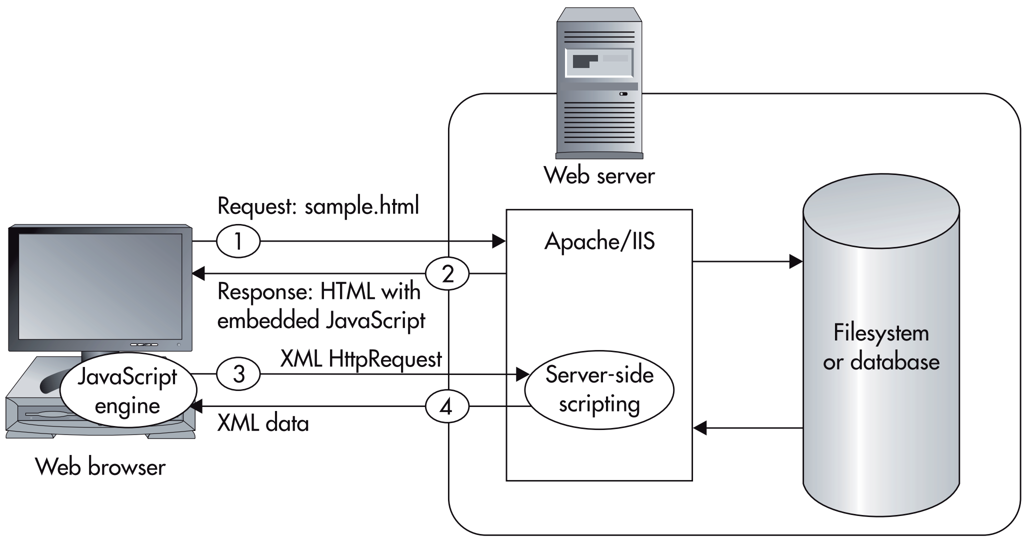


Figure 3.1: Client-Server architecture and communication using JavaScript and Ajax [taken from [Davis and Phillips, 2008]].

Netscape Navigator. Microsoft later developed their own version of JavaScript - JScript - for Microsoft Internet Explorer. This led to several problems concerning interpretation of JavaScript in different browsers. Even nowadays, although W3C's DOM was supposed to tackle this problem, there are still differences in interpretation leading to different results and errors SELFHTML e.V. [2010].

Another drawback is that none of these technologies really focuses on graphical elements. There are several JavaScript graphic libraries adding this support, but compared to other technologies which focus on graphical elements and animations, they are not as powerful and simple to use. Ajax is mostly intended for use across the internet, limiting its use on the local desktop Kosara [2007b].

3.2 Java

Developed by Sun Microsystems in 1991, Java is one of the most popular programming languages. Java is object-oriented and has a very large function library. Java is also operating system independent. This is achieved by compiling Java applications into a special byte code, which is then executed inside a Java Runtime Environment (JRE). Applications written in Java and intended for use on the web are usually called Java applets. These applets run client-side and thus are able to provide interactive features.

The major benefit of using Java to create visualisation gadgets is its huge function library. In its 6th version, the Java programming language has amassed a variety of powerful but easy-to-use libraries (shown in Figure 3.2) including graphical libraries as well as libraries to support user interaction.

Another advantage is that because Java is a licensed product, the specifications are the same everywhere, leading to no problems concerning different programming and execution environments. A further benefit of Java is that applets can be converted by open source tools to standalone platform-independent applications which can be executed offline Boy and Senapati [2010].

However Java also has some downsides. The main reason why Java applets are not as often used as

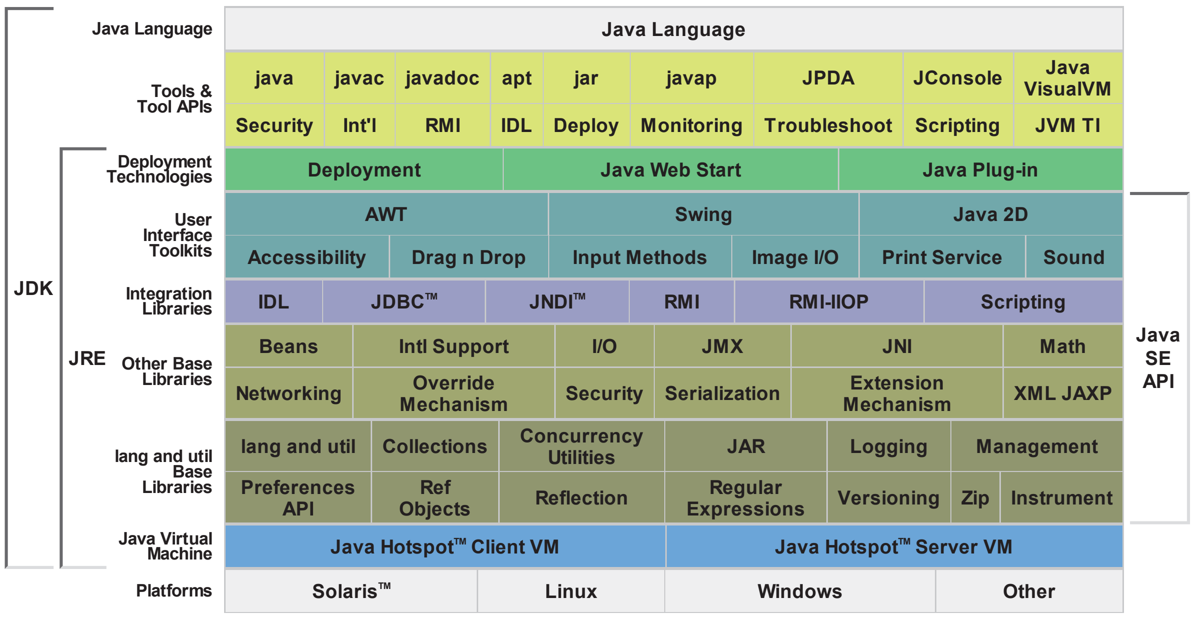


Figure 3.2: The architecture of the Java Development Kit Version 6, with its tools, toolkits, and libraries. (Image taken from Oracle [2010b]).

they could be, is the fact that the client needs to have the Java Runtime Environment installed to launch the applets. While having a JRE installed was common some years ago, this is not the case nowadays Kosara [2007b].

3.3 Java FX

JavaFX is Sun Microsystems answer to Adobe Flex and Microsoft Silverlight to develop rich internet applications (RIAs) [Oracle, 2010a]. JavaFX has its own scripting language called JavaFX Script that adds a new API with graphical and network functions. Java code can be integrated to make use of the large Java API. The JavaFX applications are, like usual Java applications, transformed to Java bytecode which is interpreted by the Java Runtime Environment, thus forcing the user to install the Java Runtime Environment on the local computer.

3.4 Adobe Flex

Adobe Flex is an open source framework for building rich internet applications. For more information on Flex see Chapter 5.

3.5 Adobe AIR

Adobe Integrated Runtime (AIR) helps to develop deployable desktop applications with Adobe Flash, Adobe Flex, HTML or Ajax. According to Adobe [2010b] AIR gives developers access to a set of Adobe AIR API functions that enable to access a broad variety of desktop functionality and resources (seen in Figure 3.3) . This includes full local file access, drag-and-drop support, access to multiple servers and other desktop applications, data base access, background processing, system notifications and more. AIR applications can be directly deployed onto the desktop and executed without the need for a browser. To be able to execute Adobe AIR applications the AIR runtime needs to be installed. This can be done when

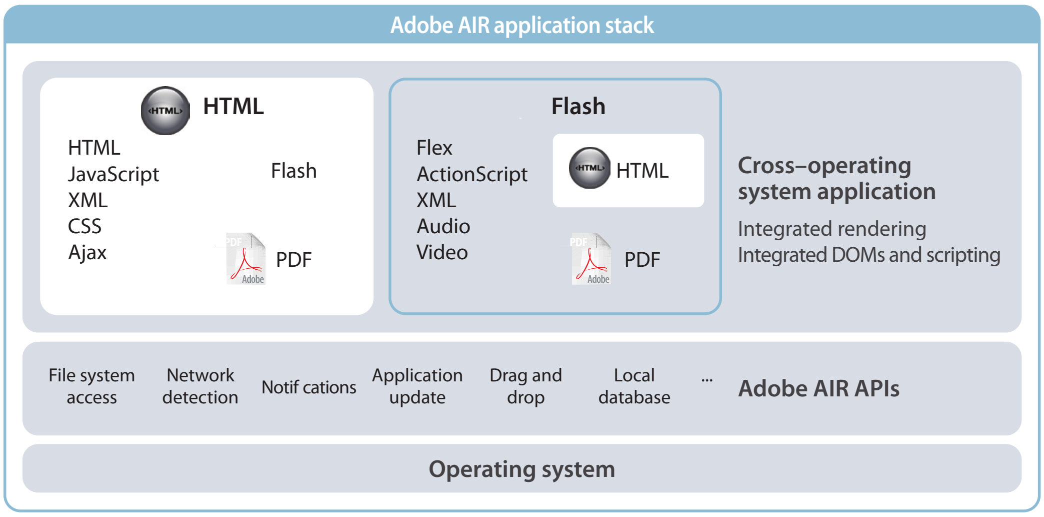


Figure 3.3: The integration of the Adobe AIR API (Image taken from Adobe [2010b]).

installing an application or prior to that by installing the runtime separately.

Like Adobe Flex, the AIR SDK is available without charge. The SDK can be used in combination with any text editor to create applications. However the Adobe Flash Builder (formerly known as Flex Builder), an integrated development environment (IDE) for the creation of rich internet applications is subject to charge.

3.6 Microsoft Silverlight

In April 2007 Microsoft launched Silverlight, their answer to Adobe's Flash and Flex. Like Flex, Silverlight is also designed to enable developers to easily create rich internet applications. Silverlight uses XAML (Extensible Application Markup Language) to describe vector graphics and animations. One of the benefits of XAML is that the textual content created with Silverlight is searchable and indexable by search engines. This is achieved by not compiling the textual content. JavaScript is used to access XAML and to alter the document to achieve changes in the user interface.

The differences between Silverlight and Adobe's Flex was summarised by nirajswami [2007] as:

- **Indexable**
  The textual content of Silverlight applications is searchable by search engines. Adobe [2010g] is also looking to enhance the search engine indexing of the Flash file format (SWF) to uncover information that is currently undiscoverable by search engines.

- **Not available for Linux**
  Although Microsoft intended to make Silverlight platform-independent, they have not fully achieved this, because Silverlight is not available on Linux systems. A third-party corporation is currently working on a Linux implementation of Silverlight called Moonlight Mono [2010].

- **Not all image formats supported**
  Adobe Flash supports all common image types, Silverlight only supports PNG and JPG files.

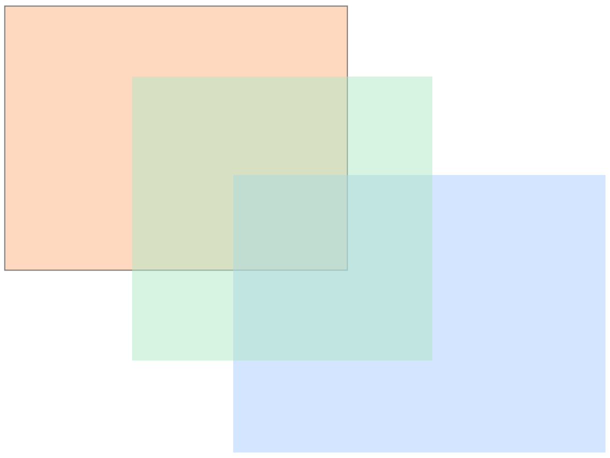


Figure 3.4: A rendering of the SVG code in Listing 3.1. The resulting image is fully scalable and can be easily edited at a later time.

Prior to Silverlight version 3 there was no option to create Silverlight applications which support offline use. The technology enabling this, called Silverlight Out of Browser (OOB), was introduced with Silverlight 3. The difference between the Silverlight OOB and AIR was that Silverlight OOB ran in the web browser's sandbox thus limiting the possible actions to the browser level (no full file access) [Steward, 2010]. With the recent release of Silverlight version 4 in April 2010 Microsoft addressed this drawback and introduced a Fully Trusted OOB capability [Huckaby, 2010].

3.7 Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) is an XML-based language to describe two-dimensional graphics [W3C, 2010c]. The language is under development since 1999 by the World Wide Web Consortium (W3C) and was introduced in September 2001 [W3C, 2010e]. As the name indicates the major benefit of Scalable Vector Graphics is that they are completely scalable without loss of quality.

The language offers three types of elements to work with [W3C, 2010c]:

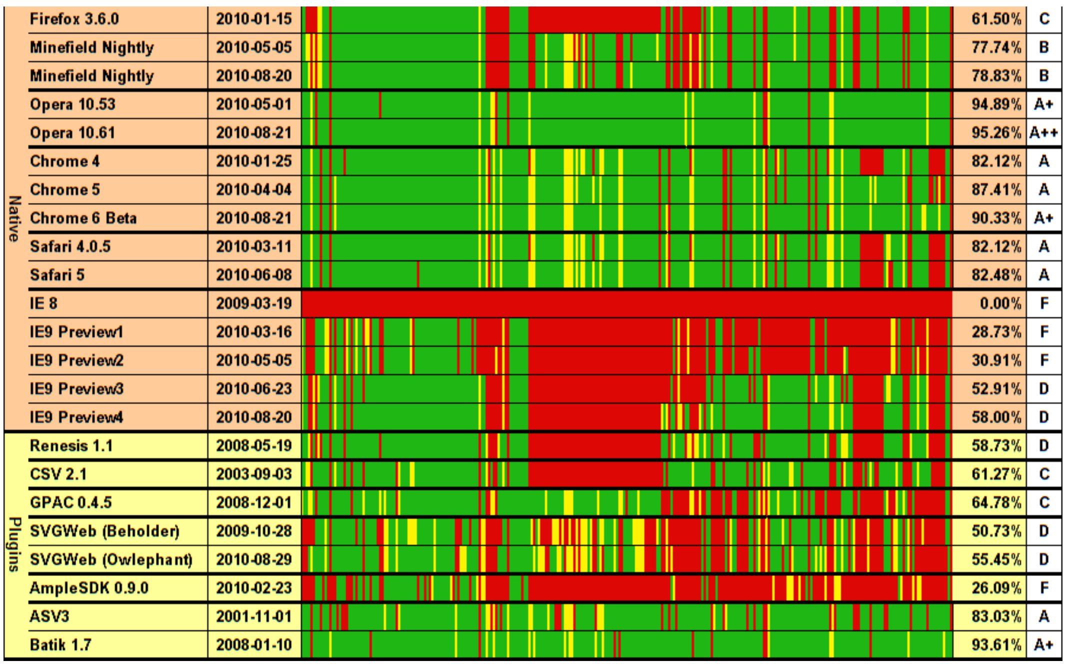
- **Shapes:** Shapes are the basic drawing elements like lines, curves or paths.
- **Images:** This element allows the import of an external image file into the SVG document.
- **Text:** Text is entered as plain text and rendered according to the attributes and properties.

Scalable Vector Graphics are not limited to static content. By scripting or embedding animation elements, the drawings can be interactive and thus dynamic. According to the W3C [2010c] it is also possible to access all elements, attributes and properties using the SVG Document Object Model (DOM). An example of a Scalable Vector Graphic can be seen in Listing 3.1 and Figure 3.4.

There are basically two ways to create Scalable Vector Graphics. The first way is to manually create the graphic by entering SVG tags in any text editor. The alternative is to draw the graphic using a graphical SVG editor. The most common graphical SVG editors are Adobe Illustrator [Adobe, 2010d], Corel Draw [Corel, 2010] and the freely available Inkscape [Inkscape, 2010]. While the creation of vector graphics using graphical editors is easier and more comfortable, it can also lead to larger file sizes due to unnecessary information added by the graphical editors. For example, the graphic shown in Figure 3.4 created with the code shown in Listing 3.1 only occupies 650 bytes, while the exact same result made

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG
    /1.1/DTD/svg11.dtd">

<svg version="1.1"
  id="Rectangles"
  width="530"
  height="395"
  xmlns="http://www.w3.org/2000/svg">

<rect width="300"
   height="231.42857"
   x="3.1428561"
   y="1"
   style="opacity:0.5;fill:#ffb380;
   stroke:#000000;stroke-width:1px;"/>

<rect width="325.71429"
      height="242.85715"
      x="203.14287"
      y="149"
   style="opacity:0.5;fill:#aaccff;"/>

<rect width="262.85715"
      height="248.57143"
      x="114.57142"
      y="63"
   style="opacity:0.5;fill:#afe9c6;"/>
</svg>
```

Listing 3.1: A Scalable Vector Graphic (SVG) code to display three rectangles. The rendered result can be seen in Figure 3.4.



Native	Firefox 3.6.0	2010-01-15	61.50%	C
	Minefield Nightly	2010-05-05	77.74%	B
	Minefield Nightly	2010-08-20	78.83%	B
	Opera 10.53	2010-05-01	94.89%	A+
	Opera 10.61	2010-08-21	95.26%	A++
	Chrome 4	2010-01-25	82.12%	A
	Chrome 5	2010-04-04	87.41%	A
	Chrome 6 Beta	2010-08-21	90.33%	A+
	Safari 4.0.5	2010-03-11	82.12%	A
	Safari 5	2010-06-08	82.48%	A
	IE 8	2009-03-19	0.00%	F
	IE9 Preview1	2010-03-16	28.73%	F
	IE9 Preview2	2010-05-05	30.91%	F
	IE9 Preview3	2010-06-23	52.91%	D
	IE9 Preview4	2010-08-20	58.00%	D
Plugins	Renesis 1.1	2008-05-19	58.73%	D
	CSV 2.1	2003-09-03	61.27%	C
	GPAC 0.4.5	2008-12-01	64.78%	C
	SVGWeb (Beholder)	2009-10-28	50.73%	D
	SVGWeb (Owlephant)	2010-08-29	55.45%	D
	AmpleSDK 0.9.0	2010-02-23	26.09%	F
	ASV3	2001-11-01	83.03%	A
	Batik 1.7	2008-01-10	93.61%	A+

Figure 3.5: The SVG support of various browsers, according to Schiller [2010b]. The image is the result of 280 SVG tests run by Schiller [2010a]. Each test is given a 2 pixel stripe in the bar of each browser resulting in a browser characteristic bar.

with Inkscape results in 2,484 bytes.

Scalable Vector Graphics are usually used and shown in a browser. As stated in Schiller [2010b], almost any major browser except Microsoft's Internet Explorer 8 has native support for Scalable Vector Graphics (see Figure 3.5). To display Scalable Vector Graphics in Internet Explorer 8, one of several available plug-ins need to be installed (for example the Adobe SVG Viewer available at Adobe [2010e]). However, as shown in Figure 3.5, Internet Explorer 9 will include much increased native support for SVG.

Chapter 4

Existing Visualisation Software

Due to the growing interest in visualising data more and more visualisation solutions are offered. Some solutions only render a static image, while others offer interactivity to arouse interest in the data and explore it. In this thesis, three different types of visualisation solutions are distinguished:

- Standalone visualisation software
- Visualisation libraries and components
- Online visualisation software

In this thesis the focus is set on online visualisation software, because the Liquid Diagrams Framework belongs to this category. A detailed view of standalone visualisation software and visualisation libraries and components can be found in Lessacher [2009a].

4.1 Standalone Visualisation Software

Standalone visualization applications do not require user's to go online to visualize their data. Instead, the application is launched locally on the user's computer. Using an application offline has the advantage that the user does not have to register or upload data. Standalone software is also usually much faster, because all the data and components are already installed and accessible on the local computer.

4.2 Visualisation Libraries and Components

Visualisation libraries and components are special packages created by third parties which enable users to embed third party charts into their web pages and web applications. Depending on the complexity of the packages, the user has to have some experience with at least managing web space, uploading files, and markup languages. In some cases, the user even needs to have experience in programming and scripting languages. Thus visualisation libraries are not suitable for most users. However, when set up by others (web developers, consultants...) these packages can be easily and effectively used by typical web users.

4.3 Online Visualisation Software

Online visualisation software are web applications which can be executed online using a web browser. One advantage of online visualisation software over offline visualisation software is that it does not have to be installed on the local computer and thus does not require space on the user's hard disk. This is due to the fact that the main files needed to execute the application are located on the server. In some cases, the user has to install some software in advance on their local computer, like Adobe Flash Player or the Java Runtime Environment. The size of these environments is generally very small and is needed often by other applications too.

Another great benefit of online visualisation software is that it enables users to collaborate in creating data and visualisations. Users can share their visualisations very easily with other people. Due to this process of sharing, people communicate with each other and exchange experiences. Thus knowledge and insight about data and its patterns is created. Nevertheless, this process of sharing can also be a drawback, namely if users must share and publish their data to be able to create visualisations. This can be a serious problem for internal and private data. A similar problem is that users usually have to register, before being able to use online visualisation software. Depending on the internet connection speed of users, load times of online visualisation applications can be very annoying. However, nowadays, most connections are fast enough to avoid long load times.

A brief summary of all the online visualisation solutions discussed in this thesis can be seen in Table 4.1. The table distinguishes 8 different characteristics of online visualisation software:

- **Technology:** The technology used to implement the software. For more information on the benefits and drawbacks of specific technologies, see Chapter 3
- **Handling:** Indicates how complex it is to create a customised visualisation. While some software solutions enable the creation of visualisations with only a few mouse clicks, others require users to enter source code.
- **Interactivity:** Highly interactive software supports Shneiderman's "visual information seeking mantra" [Shneiderman, 1996] and thereby enhances the users exploring experience.
- **Social Interactivity:**Software with high social interactivity enables the user to comment on and take snapshots of visualisations and discuss them with a community. This greatly enhances the insight of users for given visualisations and catalyses social activity.
- **Own Data:** This column indicates the possibility to use one's own data. While some solutions visualise uploaded data for free, the user is sometimes obliged to share the visualised data set. Some solutions also offer the possibility to pay a fee and visualise data without sharing.
- **Vector Export:** Determines if the visualisation software supports the export of vector graphics. This can be either achieved by providing an export function or by implementing the built in print function of the browser.
- **Image Export:** The visualisation solutions capabilities of exporting the visualisation as an image (any type).
- **Visualisation Types:** The basic visualisation types offered by the visualisation software. A plus indicates that there are a few other visualisation types available which are combined using other visualisations. For example horizontal bars and vertical bars would count as one.

4.3.1 IBM Many Eyes

In order to encourage sharing and conversation around visualisations, the Visual Communication Lab, part of IBM's Collaborative User Experience research group, was founded in 2004 IBM [2010b]. The

Solution	Technology	Handling	Interactivity	Social Interactivity	Own Data	Vector Export	Image Export	Visualisation Types
IBM Many Eyes	Java	Easy	High	High	Yes (Share)	No	Yes	14+
NY Times - Visualization Lab	Java	Easy	High	High	No	No	No	14+
Google Image Charts	Ajax	Medium	None	None	Limited	No	Limited	9
Swivel	Ajax	Easy	Medium	High	Yes (Share)	No	No	5
Verifiable.com	Flex	?	Little	None	Yes (Share)	No	Yes	3+
Google Standard Gadgets	Ajax	Easy	Little	No	Yes	No	No	12+
Google Interactive Charts	Ajax / Flex	Difficult	Little	No	Yes	No	No	16+
iCharts	Flex	?	High	Medium	No (Pay)	Yes	Yes	3+
Gapminder (Site)	Flex	Easy	High	No	No	Yes	Yes	1
OECD eXplorer	Flex	Easy	High	No	Limited	No	Yes	4

Table 4.1: Comparison of the named Online Visulisation Software solutions

founder was Martin Wattenberg, a mathematician who is known for various visualisations including the Name Voyager [Wattenberg, 2005]. Together with Fernanda B. Viégas he created Many Eyes in 2007 IBM [2010a].

Many Eyes is a web site where users can share and upload their data to create visualisations based on that data. This process of sharing is obligatory if the user wants to visualise data. Nevertheless, it enables to visualise data for free without any limitations. A focus of Many Eyes lies on social interactions between the creators and viewers of visualisations. Many Eyes offers many functions to comment, annotate and bookmark (create snapshots) visualisations. This increases social interaction and can also be helpful to discover patterns gain deeper insight into visualisations and their data. According to Viégas et al. [2007] the visualisations catalyse social activity.

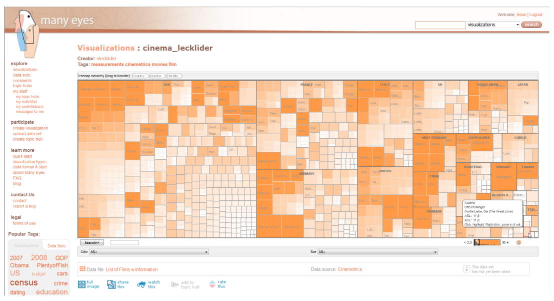
In order to upload the data the user has to prepare it in a tab separated file, according to a specific format. Due to the fact that Many Eyes offers sixteen different types of visualisations the formats needed for the visualisations vary slightly. The data is uploaded by being pasted in the specific format into a specific text box at the upload site. Afterwards, the user has to fill out some text fields describing its content. There is no way to change the data set later on, because Many Eyes includes no data editor like most standalone applications. The data is also revealed to all other internet users and is even downloadable by them.

Many Eyes offers many great looking visualisations including complex ones like tree maps, world maps and scatter plots (see Figure 4.1). All visualisations are written in Java and are highly interactive. The user has many options to rearrange, sort, and filter the data. One drawback of the visualisations is that the user has to have the Java Runtime Environment installed to see the visualisation. Each user can create all available visualisations from any uploaded data set by just selecting the dataset, selecting the visualisation, and pressing the visualise button.

A major drawback of Many Eyes is that there is no support for exporting a visualisation as a vector graphic. The only way is to save a screenshot of the maximised content. Even when printing the visualisation as PDF, the visualisation will be printed as a raster image rather than a vector graphic.

4.3.2 NY Times - Visualization Lab

The Visualization Lab is a special form of IBM's Many Eyes hosted by the New York Times [2010]. It has exactly the same visualisations as Many Eyes. The only difference is that the visualisations at the Visualization Lab are only available for data sets uploaded by the NY Times. Thus, users are not able to upload their own data sets.


(a) A dataset containing movies visualized as a tree map.

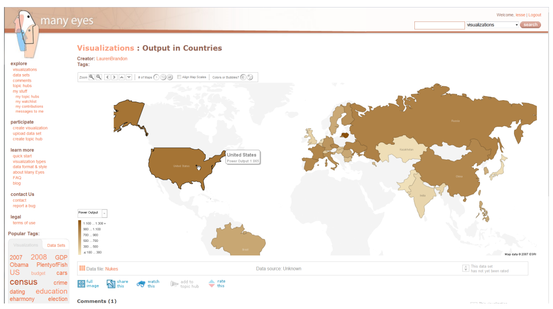

(b) Another type of visualisation available in Many Eyes is the world map. The countries in the data set are added by using their names or ISO codes.

Figure 4.1: Many Eyes offers lots of highly interactive Java written visualisations.

Thus there are by far not as many data sets available as in Many Eyes, but the existing data sets are of high quality and trustable resources. Since this feature is free, it is a nice addition to the New York Times web site, to be able to explore the original data associated with the hosted articles and facts on the site.

4.3.3 Google Image Charts

The Google Chart API offers a simple way to dynamically create charts, called Google Image Charts, which are static visulisations Google [2010b]. To create a chart the user only has to enter a specific URL containing the location of the chart API, the diagram type and options into the address bar of the browser. Figure 4.2 shows an example of such a URL.

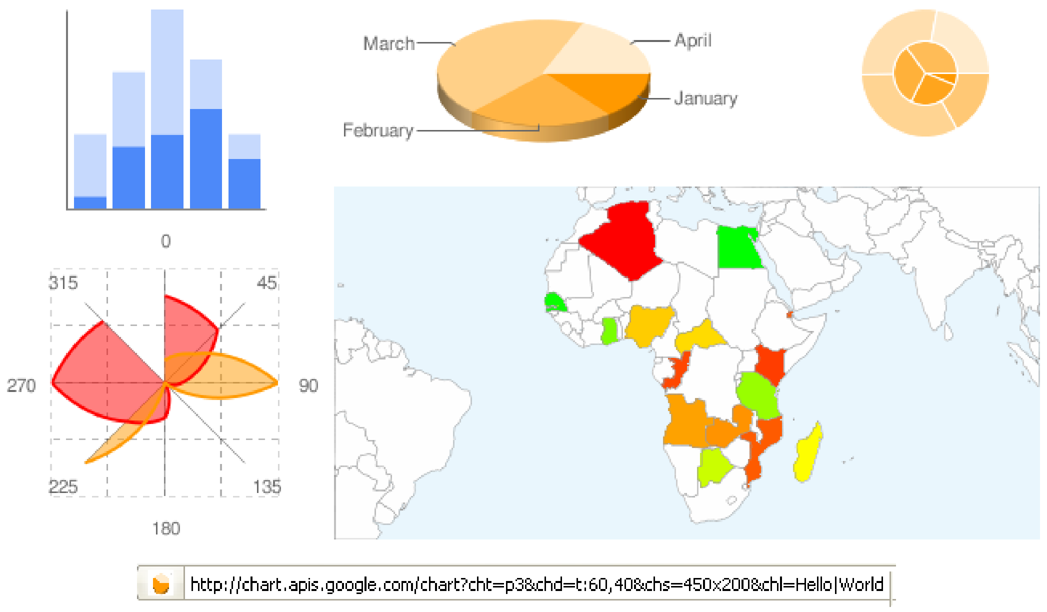
Since the data to visualise is part of the URL, there is no way to import data from a file. The user has to manually enter the data in the appropriate format through the URL in the address bar. There are nine different types of visualisations available, each with a set of options to customise the chart. Although the interface is kept very simple, there are some nice-looking charts, like radar charts and maps, available. The resulting charts are not the prettiest, are limited in size, and having no vector graphics, but depending on the complexity of the URL (data, options) they are created in a few minutes. The chart is simply drawn into the browser window, where it can be saved by taking a screenshot or using the browsers "Save Image as.." function.

In addition, Google logs the used URL for internal debugging and testing purposes Google [2010c]. Since the data is also packed into the URL, one might conclude that Google is logging the user's data.

4.3.4 Swivel

Swivel, like Many Eyes, is a web site where users can upload their data to visualise it Swivel [2010]. Founded in 2005 by two physics graduates swivel is a business with 10 employees and a business plan Kosara [2007b].

In a addition to the standard Swivel, where users publish their data to visualise it without cost, there is an option to create a private group. Private groups require a monthly payment according to the number of spreadsheets and charts available to the group. The benefit of private groups is that the uploaded data



http://chart.apis.google.com/chart?cht=p3&chd=t:60,40&chs=450x200&chl=Hello|World

Figure 4.2: Google charts are very simple but easily created (Image taken from Google [2010b]). The data is included in the callling URL.

is only available to members of the group.

In contrast to Many Eyes, Swivel is made using Ajax. The visualisations are neither as pretty nor as interactive as in Many Eyes, but run in nearly every modern browser without requiring the user to install additional environments. Swivel is also very limited in its visualisations. They only offer five different types of visualisations (bar charts, pie charts, line charts, area charts and scatter plots).

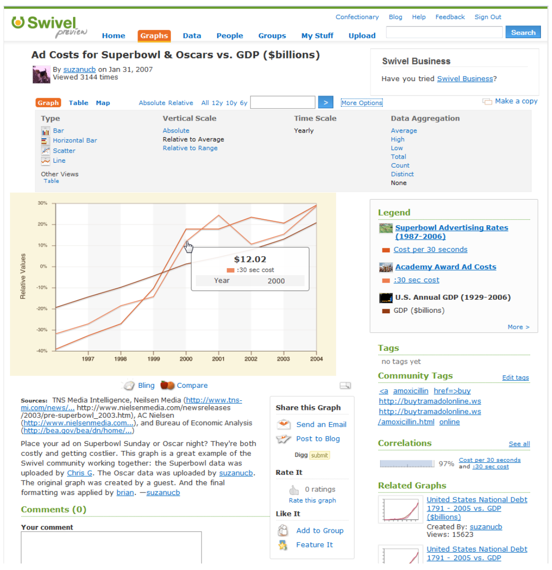
In Swivel data sets can be imported from comma separated (csv) files or by using copy and paste into a text box. After the import, the user does not choose a specific visualisation type, but instead Swivel automatically creates several visualisations resulting in a much larger number of visualisations than data sets. One benefit of this creation variant is that the user can simply switch between different visualisations by pressing the appropriate button.

In Swivel, users can comment on visualisations created by other users. In addition, they can directly compare different data sets. They just have to choose another data set and the two data sets are visually merged, as shown in Figure 4.3.

Swivel offers no export functions to obtain images of the visualisations. Except for the option to maximize (to a predefined value) the visualisation, there is no option to resize the visualisation, because it is not displayed as vector graphic.

4.3.5 Verifiable.com

Verifiable.com was a web site enabling anyone to create visualisations [Visible Certainty, 2010b]. Visible Certainty, owners of the web site, announced that it would close on 1st of August 2010, two years after it launched in July 2008 [Visible Certainty, 2010a]. According to Kosara [2010] the site offered two different user services. The first service was free of charge and allowed the users to share and upload their data to visualise it. The second service was for paying customers and allowed them to keep their



Ad Costs for Superbowl & Oscars vs. GDP ($billions)

By suzanucb on Jan 31, 2007
Viewed 3144 times

Swivel Business

Have you tried Swivel Business?

Sources: TNS Media Intelligence, Neilsen Media (http://www.tns-mi.com/news/... http://www.nielsenmedia.com/newsreleases/2003/pre-superbowl_2003.htm), AC Nielsen (http://www.nielsenmedia.com...), and Bureau of Economic Analysis (http://bea.gov/bea/dn/home/...)

Place your ad on Superbowl Sunday or Oscar night? They're both costly and getting costlier. This graph is a great example of the Swivel community working together: the Superbowl data was uploaded by Chris G. The Oscar data was uploaded by suzanucb. The original graph was created by a guest. And the final formatting was applied by brian. —suzanucb

no tags yet

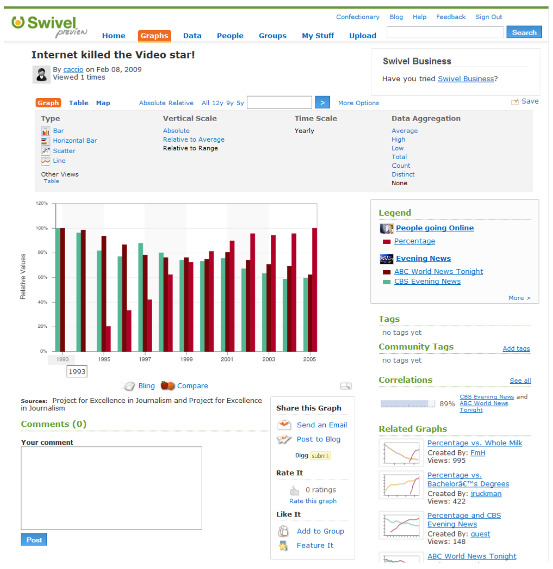
<a amoxicillin href=>buy http://buytramadolonline.ws http://buytramadolonline.ws/amoxicillin.html online

97% Cost per 30 seconds and :30 sec cost

United States National Debt 1791 - 2005 vs. GDP ($billions)
Created By: suzanucb
Views: 15623

United States National Debt 1791 - 2005 vs. GDP

(a) This visualisation is a result of one of the features of Swivel. It is a comparison of three different data sets.



Internet killed the Video star!

By caccio on Feb 08, 2009
Viewed 1 times

Swivel Business

Have you tried Swivel Business?

Sources: Project for Excellence in Journalism and Project for Excellence in Journalism

no tags yet

no tags yet

89% CBS Evening News and ABC World News Tonight

Percentage vs. Whole Milk
Created By: FmH
Views: 995

Percentage vs. Bachelorâ€™s Degrees
Created By: jruckman
Views: 422

Percentage and CBS Evening News
Created By: guest
Views: 148

ABC World News Tonight

(b) Here is another combination of data sets visualised as bar graph. This visualisation shows how the internet supplanted TV.

Figure 4.3: Swivel only supports four different types of visualisations two of them bar chart and line chart are shown here.

data private. Compared to Many Eyes, Verifiable.com offered only three basic types of visualisation (lines, points, and bars) and although the diagrams were written in Flex they were not as interactive as the Java visualisations of Many Eyes. In contrast to Many Eyes and Swivel, Verifiable.com did not offer social features like commenting on a visualisation.

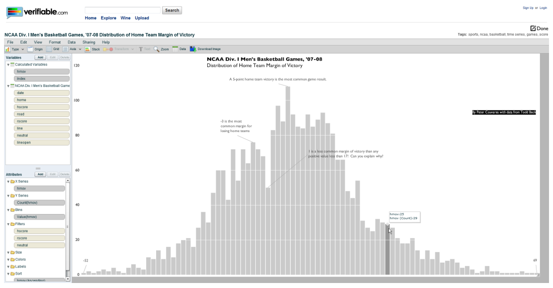
A nice feature Verifiable.com offered was the editor for visualisations. By pressing the `Edit` button displayed above the visualisation the user was able to change visual aspects of the visualisation. It was possible to change the type of the visualisation, data colours, labels, and which data column was displayed along the axes. An example of the editor can be seen at Figure 4.4a.

In the editor, the visualisation is shown as a vector graphic and thus can be resized, but there is no option to export the image as a vector graphic. The print option of the Flex right click menu has been removed and the browser's print function does not even include the graphic to its output. At least it is possible to export a high resolution PNG image, like the one shown in Figure 4.4b.

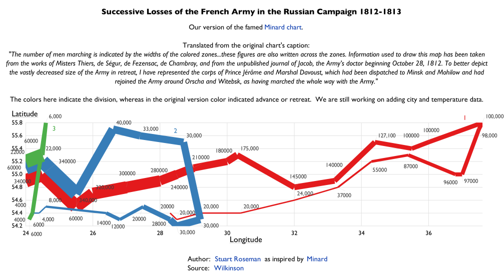
4.3.6 Google Interactive Charts

Google Docs [Google, 2010d] is Google's response to Microsoft's Office Suite. As in Microsoft Office, users can write texts, design presentations, and handle data using spreadsheets. The two main differences to Microsoft Office are that Google Docs is completely free and is typically used online. This enables the user to share documents with others, leading to interaction and collaboration.

One special feature of spreadsheets created with Google Spreadsheets is the ability to add charts in the form of externally programmed gadgets. There are thirteen basic gadgets including pie chart, area chart, and bar chart implemented by Google. Additionally, it is possible for every user to create and publish their own gadgets, as demonstrated in Figure 4.5


verifiable.com
Home Explore Wire Upload
NCAA Div. I Men's Basketball Games, '97-08 Distribution of Home Team Margin of Victory

(a) The visualisation editor offered by Verifiable.com. Several visual aspects like the type of the visualisation or which data columns are displayed on the axes can easily be set.


Successive Losses of the French Army in the Russian Campaign 1812-1813

Our version of the famed Minard chart.

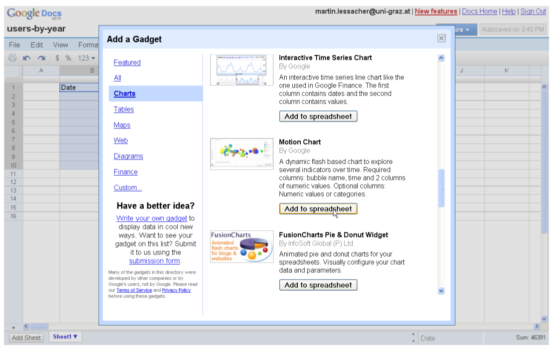
Translated from the original chart's caption:
"The number of men marching is indicated by the widths of the colored zones...these figures are also written across the zones. Information used to draw this map has been taken from the works of Misters Thiers, de Ségur, de Fezensac, de Chambray, and from the unpublished journal of Jacob, the Army's doctor beginning October 28, 1812. To better depict the vastly decreased size of the Army in retreat, I have represented the corps of Prince Jérôme and Marshal Davoust, which had been dispatched to Minsk and Mohilow and had rejoined the Army around Orscha and Witebsk, as having marched the whole way with the Army."

The colors here indicate the division, whereas in the original version color indicated advance or retreat. We are still working on adding city and temperature data.

Author: Stuart Roseman as inspired by Minard
Source: Wilkinson

(b) This high resolution export of a visualisation displays Charles Minard's data set of Napoleon's 1812 Russia campaign.

Figure 4.4: Verifiable.com offers an easy to use editor with the option to export PNG images.


Add a Gadget

Featured
All
Charts
Tables
Maps
Web
Diagrams
Finance
Custom...

Have a better idea?
Write your own gadget to display data in cool new ways. Want to see your gadget on this list? Submit it to us using the submission form.

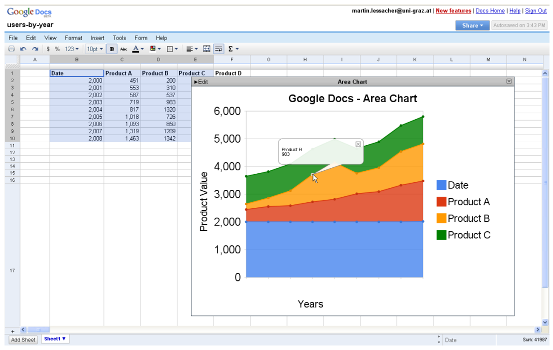
Many of the gadgets in this directory were developed by other companies or by Google's users, not by Google. Please read our Terms of Service and Privacy Policy before using these gadgets.

Interactive Time Series Chart
By Google
An interactive time series line chart like the one used in Google Finance. The first column contains dates and the second column contains values.
Add to spreadsheet

Motion Chart
By Google
A dynamic flash based chart to explore several indicators over time. Required columns: bubble name, time and 2 columns of numeric values. Optional columns: Numeric values or categories.
Add to spreadsheet

FusionCharts Pie & Donut Widget
By InfoSoft Global (P) Ltd.
Animated pie and donut charts for your spreadsheets. Visually configure your chart data and parameters.
Add to spreadsheet
Google Docs
users-by-year
martin.lessacher@uni-graz.at | New features | Docs Home | Help | Sign Out

(a) In Google Spreadsheets users can choose gadgets to visualise their data. In addition to gadgets written by Google, anyone can publish their own gadgets to share with others. There is also the possibility to choose specific gadgets located somewhere in the web using the Custom option.


Date	Product A	Product B
2,000	451	200
2,001	553	310
2,002	587	537
2,003	719	983
2,004	817	1320
2,005	1,018	726
2,006	1,093	850
2,007	1,319	1209
2,008	1,463	1342

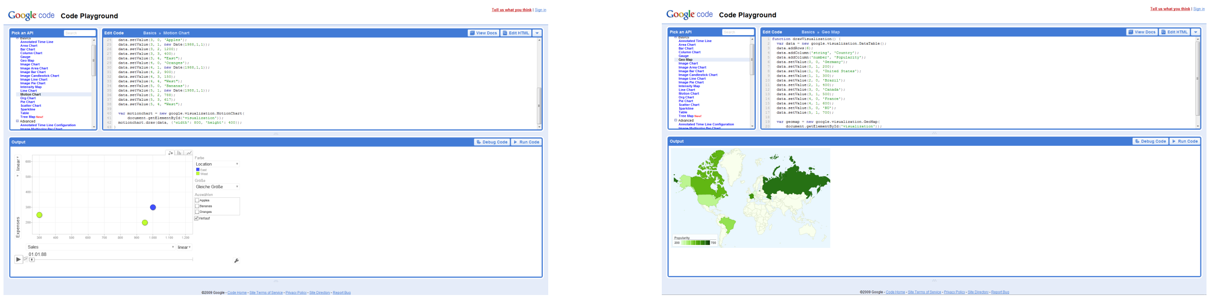
(b) An area chart gadget written by Google. The visualisation can be customised by editing parameters when creating the gadget, or by pressing the edit button in the top left corner of the gadget window.

Figure 4.5: Google Spreadsheets are used online and can be linked to externally programmed visualisation gadgets.

Gadgets are special HTML and JavaScript applications which can be used on web pages like iGoogle, Google Maps, and others [Google, 2010a]. The basic element of a gadget is its XML file which contains all necessary information about how to process and render the gadget. Every XML file meeting Google's specifications can be added to a spreadsheet to visualise its data. It does not matter where the XML file is located in the internet. There are a number of tutorials and standards written by Google on how to implement visualisations as gadgets [Google, 2010e]. These tutorials include examples demonstrating how to pass data from within spreadsheets in Google Docs to an external gadget. Each gadget must document how it requires its data to be structured in the spreadsheet. Sometimes, the meaning of rows and columns needs to be swapped (transposed).

Up to now, following groups or individuals have published gadgets for spreadsheets in Google Docs:

- **Infosoft Global:** Infosoft Global contributed three gadgets made with Adobe Flash or Flex. Although Flash is known for its strengths in creating interactive applications, these gadgets do not support user interaction. Infosoft Global also offers other libraries which enable the user to inte-



```
  data.setValue(3, 0, 'Apples');
  data.setValue(3, 1, new Date(1988,1,1));
  data.setValue(3, 2, 1200);
  data.setValue(3, 3, 400);
  data.setValue(3, 4, "East");
  data.setValue(4, 0, 'Oranges');
  data.setValue(4, 1, new Date(1988,1,1));
  data.setValue(4, 2, 900);
  data.setValue(4, 3, 150);
  data.setValue(4, 4, "West");
  data.setValue(5, 0, 'Bananas');
  data.setValue(5, 1, new Date(1988,1,1));
  data.setValue(5, 2, 788);
  data.setValue(5, 3, 617);
  data.setValue(5, 4, "West");

  var motionchart = new google.visualization.MotionChart(
      document.getElementById('visualization'));
  motionchart.draw(data, {'width': 800, 'height': 400});
}
```

```
function drawVisualization() {
  var data = new google.visualization.DataTable();
  data.addRows(6);
  data.addColumn('string', 'Country');
  data.addColumn('number', 'Popularity');
  data.setValue(0, 0, 'Germany');
  data.setValue(0, 1, 200);
  data.setValue(1, 0, 'United States');
  data.setValue(1, 1, 300);
  data.setValue(2, 0, 'Brazil');
  data.setValue(2, 1, 400);
  data.setValue(3, 0, 'Canada');
  data.setValue(3, 1, 500);
  data.setValue(4, 0, 'France');
  data.setValue(4, 1, 600);
  data.setValue(5, 0, 'RU');
  data.setValue(5, 1, 700);

  var geomap = new google.visualization.GeoMap(
      document.getElementById('visualization'));
```

Figure 4.6: The Google Visualisation Playground. On the left of the playground is a menu bar, which is used to choose the visualisation type. The right side is used to display JavaScript code, which can be altered to modify the visualisation shown at the bottom.

grate charts into web pages.

- **Viewpath:** Viewpath [2010] offers Gantt charts.
- **David Huynh and Timeline Fans:** A timeline visualisation gadget.
- **Greg Marra and Seth Glickman:** A spider chart gadget.
- **Yaar Schnitman**: A simple treemap gadget.

4.3.7 Google Visualization Playground

Although the Google Visualization API would better fit into the category of visualisation libraries, it is located in the online visualisations section because of the Visualisation Playground [Google, 2010f], which offers more than 20 available visualisation types. The user can choose a visualisation and the corresponding JavaScript code for this visualisation is shown. By altering the code the user can customise the visualisation and enter their own data. Since this code is written in JavaScript, it is necessary to know at least some programming basics to successfully adapt the visualisation. It is also rather uncomfortable to enter the data as code, making it hard to enter large amounts of data. There is no way to import data. After adapting the code, the visualisation can be displayed by pressing the run button. An example is shown in Figure 4.6.

The resulting visualisations are the same as the ones created with the Google Docs gadgets because the Google Visualisation API is used. A gadget handles the JavaScript code for the user, making it easier and more comfortable to create visualisations. The reason the playground is mentioned is that more visualisations are available, some of which cannot be added using Google Docs gadgets.

4.3.8 iCharts

iCharts offers two services to visualise data. The first is to use the iCharts blog [iCharts, 2010a] where new datasets are posted on nearly daily basis by selected people. There is no possibility to register and upload one's own data. The second option is to use the iCharts business service [iCharts, 2010b]. There is no function to register for this service. There is also no information about pricing and conditions located on the web page. The only way to subscribe and obtain a personal solution is to contact the creators of iCharts via email. Nevertheless there are a few data sets and visualisations available to be looked at.

The web site appears to offer only three different types of visualisations: pie charts, bar charts, and line charts. The visualisations use Flash Player, which suggest that they are written with either Adobe

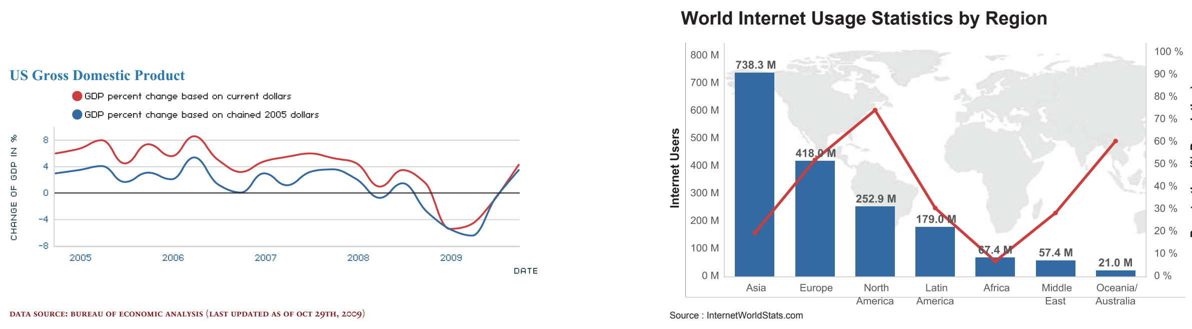


Figure 4.7: Chart examples hosted on the iCharts Site. Both charts were exported using the print function to generate a vector graphic.

Flash or Adobe Flex. The charts appear to be slightly modified versions of the standard charts included in Adobe Flex. The charts support some standard interactions like mouseover and clicking of elements, as well as sliders to zoom the data range. There is also a print function included to print out the visualisation. This is especially useful to obtain a vector graphic by using a PDF maker (shown in Figure 4.7).

A nice feature of iCharts is the possibility to embed the resulting chart into another web page. The user is given the appropriate source code snippet, which can be pasted into other web pages to include the chart. Social interactions like posting comments are also available.

4.3.9 Gapminder

Founded in February 2005 by Ola Rosling, Anna Rosling Rönnlund and Hans Rosling, the Gapminder foundation developed the Trendalyzer software, which is known as Gapminder World since 2006. Trendalyzer allows the exploration of statistical time series data in a highly interactive and animated environment [Gapminder, 2010b].

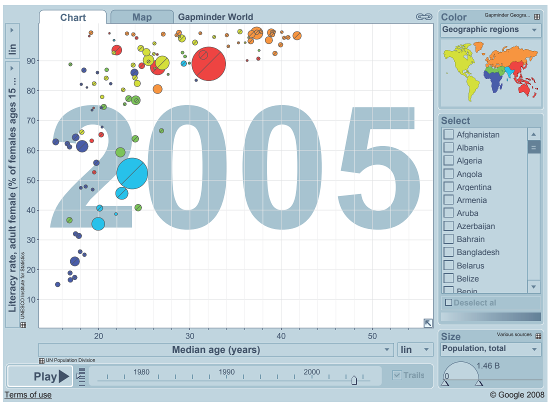
In March 2007 the Trendalyzer software and some employees who worked for Gapminder were acquired by Google. Google Docs now offers a standard gadget called Motion Chart which is very similar to Gapminder World. To use this gadget is the only way to visualise private data. There are many datasets at the Gapminder World homepage showing trends and statistical data, but they are all uploaded by the site administrators [Gapminder, 2010a].

When the displaying a time series, a time bar at the bottom of the visualisation can be used to explore the data over a time period. It can also be animated using the play button. This is very useful when analysing trends over time.

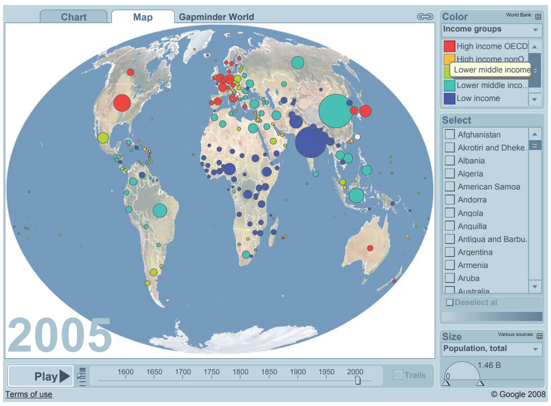
Using the print function of the browser the visualisations can be exported as vector graphic (examples shown in Figure 4.8). Unfortunately, this only works properly for the visualisations on the Gapminder homepage because the Google Gadgets version cuts off parts of the visualisation.

4.3.10 OECD eXplorer

According to the OECD [2010] Mikael Jern of the National Centre for Visual Analytics (NCVA) at Linköping University released the preliminary version of the OECD eXplorer on the OECD website



(a) A bubble chart displaying the litaracy rate of women over the median ages.



(b) Another option when using Gapminder World is to display the data on a world map.

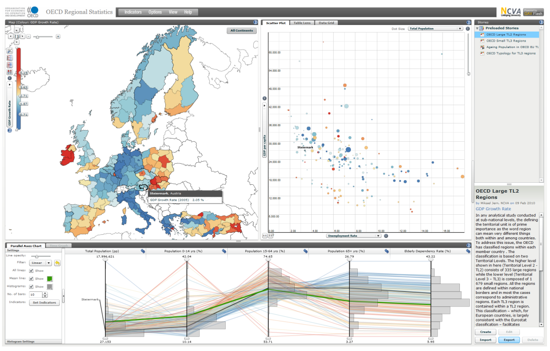
Figure 4.8: Two visualisations created at the Gapminder World homepage. Both visualisations are time series and thus can be animated by using the time bar at the bottom.

in 2008. The OECD eXplorer is a highly interactive application to view and analyse statistical data within and across the OECD countries [NCVA, 2010b] which offers the functions introduced in Jern et al. [2008].

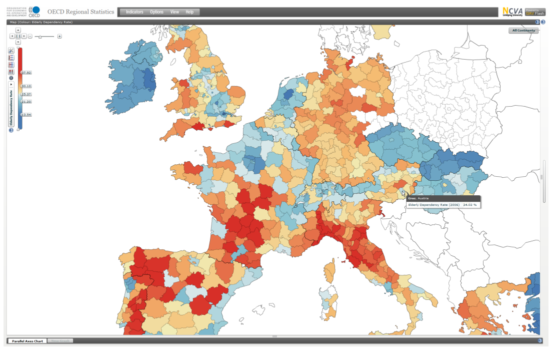
To easily detect trends and patterns the OECD eXplorer offers several visualisation types (choropleth map, scatter plot, table lens, parallel coordinates) which are embedded into the interface as separate views (shown in Figure 4.9). According to NCVA [2010b] the choropleth map is implemented using four different map layers. The base layer is an optional Google Map layer to view detailed regional information. Due to its high resource demands, this layer can be turned off to increase performance on slower computers. On top of the Google Map layer, there is a country border layer and a colour shape layer.

The OECD eXplorer also offers time animations and so-called storytelling [NCVA, 2010a]. By using the time control element the data can be explored over a period of time. All views are linked to the time control and change according to the data of the specific time period.

It is possible to import one's own data, but unfortunately only if the data is related to OECD regions and maps. Although there is a function to export an image it is limited to PNG or JPEG formats and seems to work with the map view only. There is no possibility to export a vector graphics version of a visualisation. The browser print function clips the content making it unusable.



(a) The OECD-eXplorer is divided into several views displaying different visualisations. The views are linked to each other to enhance the detection of patterns and outliers.



(b) The map view is based on several layers. In this image, only the border layer and the colour shape layer are enabled.

Figure 4.9: The OECD eXplorer is a highly interactive application offering many functions to explore statistical data.

Chapter 5

Flash and the Web

5.1 Adobe Flash

Flash was introduced by Macromedia in 1996 and is now owned by Adobe. It enables the addition of animations and interactivity to web pages by manipulating raster and vector graphics. To do so, Flash uses a scripting language called ActionScript. ActionScript is an object-oriented programming language originally developed by Macromedia and has very similar syntax and semantics as JavaScript. See Listing 5.1 for a JavaScript example.

The file format used by Flash is called SWF (Shockwave Flash). An SWF file is a compiled binary SWF object, which is executed using the Adobe Flash Player. The SWF object can then be included within a HTML page as described in W3C [2010b] and shown in Listing 5.2. Listing 5.3 illustrates how the HelloWorld SWF object, created in Listing 5.4, is added to a HTML page. The result when viewing this HTML page can be seen in Figure 5.1.

Like Java, Flex creates platform-independent applications. The downside of this portability is that the client has to have an appropriate version of the Adobe Flash Player installed to execute the SWF objects. Compared to the Java Runtime Environment the file size of the Flash Player is much smaller. According to Adobe [2010f] 99% of internet users have the Adobe Flash Player installed (see Figure 5.2). However there is no information on whether Apple users were involved in this study, so the numbers have to be taken with care. To run Flash content on iPads and iPhones the users need to use a third party browser called Cloud Browse [AlwaysOn Technologies, 2010] which can be acquired through the app store.

5.2 Adobe Flex

In 2004 Macromedia released the first version of Flex, an open source framework for building rich internet applications. Macromedia was acquired in 2005 by Adobe. According to Adobe [2010c], Flex supports the creation of highly interactive, expressive web applications which deploy consistently on all major browsers, desktops, and operating systems. Flex was not built from scratch, but was based partially on Adobe Flash. While Flash was designed for graphic artists implementing a time line concept, Flex's intended audience are developers who are used to a workflow and programming model Davis and Phillips [2008].

The code of a Flex application is, as in Flash, compiled into a binary SWF object, which is executed using the Adobe Flash Player. Therefore the Flash Player needs to be installed and Flex suffers the same

```
import flash.text.TextField;
import flash.display.Sprite;

public class HelloWorld extends Sprite {
  public function sayHello() {
    var textField:TextField = new TextField();
    textField.text = "Hello World!";
    addChild(textField);
  }
}
```

Listing 5.1: A “Hello World” application written in JavaScript. This example adds a TextField to the application and displays the text “Hello World!” into it.

```
<object width="800" height="600">
  <param name="content" value="flash-app.swf">
    <embed src="flash-app.swf" width="800" height="600"></embed>
</object>
```

Listing 5.2: The simplest way to include a Flash object into an ordinary HTML page.

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >

<head>
<title>Hello World - HTML Page</title>
</head>

<body>
<h1 class="title">Hello World - HTML Page </h1>
<object width="550" height="400">
  <param name="movie" value="HelloWorld.swf">
  <embed src="HelloWorld.swf" width="550" height="400"/>
</object>
</body>
</html>
```

Listing 5.3: The simplest way to include a Flash object into an ordinary HTML page.

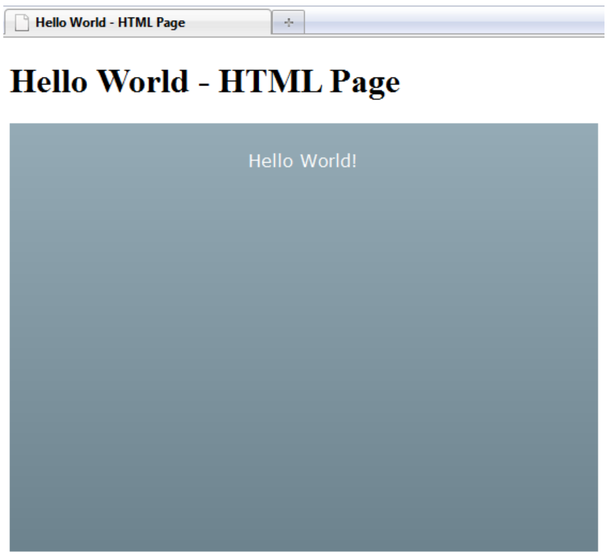


Figure 5.1: The resulting HTML page of Listing 5.3.

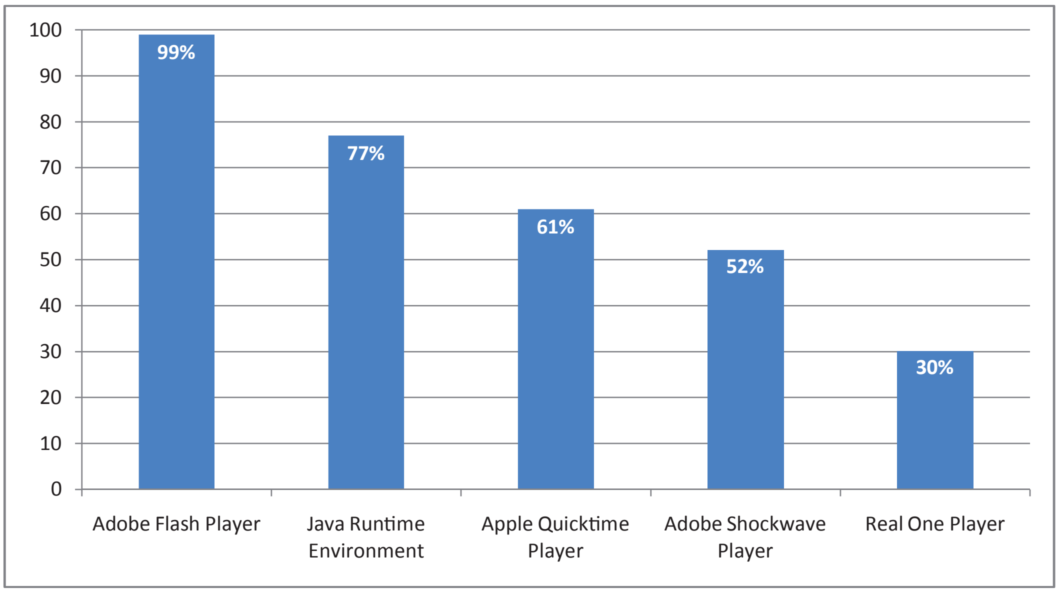


Figure 5.2: The percentage of web users having specific software packages installed in their browsers (Image taken from Adobe [2010f]). The data was taken from a study by Brown [2010].

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
        layout="horizontal" creationComplete="init()">

  <mx:Script>
   <![CDATA[
     import mx.controls.Label;
     public function init() : void {
       labelContol.text = "Hello World!";
     }
   ]]>
   </mx:Script>

  <mx:Label id="labelContol" fontSize="16" color="0xffffff"/>
</mx:Application>
```

Listing 5.4: An example for MXML code. This example adds a Label to the application and writes the text “Hello World!” into it by using ActionScript.

limitations as Flash concerning the Flash Player. The Flex framework uses two essential components, ActionScript and MXML.

MXML is an XML-based markup language allowing the developer to construct application user interfaces Coenraets [2003]. This is done in an easy and effective way by nesting tags like in an HTML or XML document (see Listing 5.4). MXML recognizes a set of specially defined interface components:

- **Controls:** Common user interface controls like buttons, progress bars, color pickers and sliders.
- **Layout:** Containers which group and distribute components inside the container.
- **Navigators:** Navigators are components like menu bars, tab bars and other components that help to navigate through oan interface.
- **Charts:** Basic customizable charts which can be easily integrated into a web application.

To include a component, a developer simply adds the appropriate component tag and provides some user-specific parameters.

Originally developed to be used in Flash, Actionscript can also be found in Flex. Each Flex application can have an ActionScript area, denoted by the opening ”$\langle mx : Script\rangle\langle[!CDATA[$” and closing ”$]]\rangle\langle/mx : Script\rangle$” tags. Inside this area developers are free to perform their desired actions including for example accessing components, handling events, or regulating control flow.

Flex was designed for creating rich internet applications like visualisation gadgets and has many libraries and control elements suited for this purpose. One limitation of SWF objects is that they preload their contents leading to an initial delay when loading. After loading, the application runs fast and smooth, without page reloads or similar delays. SWF objects can be accessed and viewed using the Adobe Flash player. If a stand-alone Flash Player is installed and the application is downloaded, it can be used offline as a stand-alone application.

Chapter 6

Liquid Diagrams Framework

The Liquid Diagrams framework is a class which offers functions enabling the creation of visualisations. The functions do not contain the logic to draw the visualisations, but offer functions to draw lines, axes, titles, legends, and more. The logic to draw a visualisation is located in the visualisation's own MXML file, which uses the framework functions. Originally startet during previous work [Lessacher, 2009b], the framework underwent numerous improvements.

6.1 Attaching Liquid Diagrams to Data Sources

Liquid Diagrams gadgets are not limited to any specific data source. The process of importing data into the Flex gadgets is to hand the data to the gadget by calling a function written in JavaScript. This implies that any data source which supports access to its data with JavaScript is suitable to be used with Liquid Diagrams gadgets. An illustration of how the gadget interacts with the data sources can be seen in Figure 6.1. Currently only one implementation is available for all Liquid Diagrams gadgets to access data sources: Google Spreadsheets, which are accessed using Google Gadget Technology. However, a group of students used the Liquid Diagrams parallel coordinates visualisation for a lecture project and used a HTML page to supply the data to the gadget (see Section 6.1.3).

6.1.1 General Embedding of Gadgets

The Flex gadget is embedded into a web page using JavaScript. As shown in Listing 6.1, a container named `chart` is added to the HTML page (line 1). This container is the host for the Flash object. The `innerHTML` element of the `chart` container is assigned the Flash object (line 7). The object definition contains many parameters (line 8-23) with the most important being:

- **classid**
  The classid defines the class of the object. The class `clsid:D27CDB6E-AE6D-11cf-96B8-444553540000` indicates the Flash plugin and must not be changed.

- **id**
  The id is the name of the object. This name is needed to reference the object when using the `GetFlexApp` function.

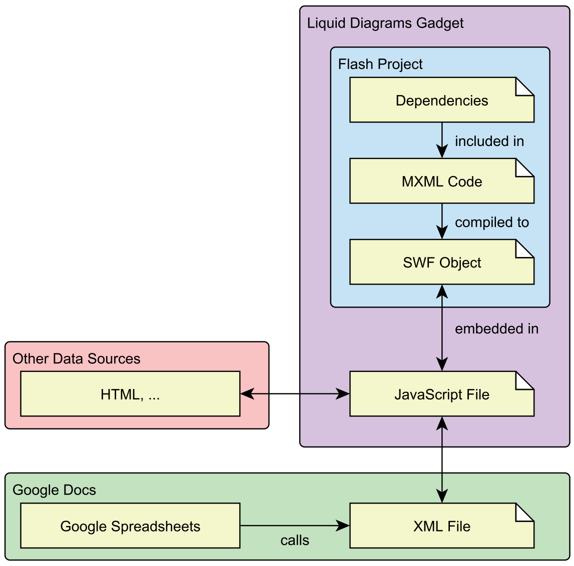


Figure 6.1: Connecting a Liquid Diagrams gadget to a data source such as a Google Spreadsheet via JavaScript. A HTML file can also be used to call the gadget and supply its data via JavaScript.

- **allowScriptAccess**
  When accessing the Flash object from different domains, script access should be set to `always` to prevent sandbox errors.

- **src**
  This is the path to the gadget's SWF object.

The embed operation causes the Flex object to be created and therefore triggers the initialisation function illustrated in Listing 6.2. The first function executed, when the creation of the Flex application finishes, is determined by the `creationComplete` event (line 3). In this first called function, an external interface is established to enable communication between the Flex gadget and its container. When the interface is established (line 5), functions can be defined which are callable by the container. The only externally callable function defined by Liquid Diagrams gadgets is the function `getDataObjects` which handles data transfer (line 6). Afterwards, a function of the container is called by the Flex gadget to signal that its initialisation process has finished and it is ready to receive data (line 7).

Listing 6.3 illustrates the final steps of the data exchange. The JavaScript function (line 2) hands the data to the Flex application by calling the previously defined function (line 3), which continues to process the data without requiring any further communication with the container (line 10). There are two different versions of the `getDataObjects` function. The first version has four parameters (as illustrated in line 3 of Listing 6.3). The second version only has two parameters (values, options) and is used by the heatmap, treemap, and voronoi visualisations. These visualisations do not need the colour parameter because the colours are computed and the headers parameter due to their different way of data computation. However, the options array handed to the visualisation is different for each visualisation. For more information on which parameters each gadget expects, refer to the individual visualisation gadget's documentation.

6.1.2 Google Docs Interface

There is currently only one external data source which can be used with all Liquid Diagrams gadgets. This data source is Google Docs, and more specifically Google Spreadsheets. To visualise Google

```
<div id="chart"></div>

<script type="text/javascript">
  var containerElement = document.getElementById('chart');

  if(containerElement != null) {
    containerElement.innerHTML =
    '<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"' +
      'id="linechart" width="320" height="240"' + 'codebase="http://fpdownload.
          macromedia.com/get/flashplayer/current/swflash.cab">' +
        '<param name="movie" value="linechart.swf" />' +
        '<param name="quality" value="high" />' +
        '<param name="bgcolor" value="#869ca7" />' +
        '<param name="allowScriptAccess" value="always" />' +
        '<embed src="linechart.swf" quality="high" bgcolor="#869ca7"' +
          'width="320" height="240" name="linechart"' +
          'align="middle"' +
          'play="true"' +
          'loop="false"' +
          'quality="high"' +
          'allowScriptAccess="always"' +
          'type="application/x-shockwave-flash"' +
          'pluginspage="http://www.adobe.com/go/getflashplayer">' +
        '</embed></object>';
    }
</script>
```

Listing 6.1: Embedding a Flex gadget into a web page using JavaScript. A container is added to a web page and the gadget's SWF object is embedded into this container.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute" creationComplete="init()"">
  public function init():void {
      if (ExternalInterface.available) {
        ExternalInterface.addCallback("getDataObjects", getDataObjects);
        ExternalInterface.call("callbackReady");
      } else {
        Alert.show("Error: No External Interface!");
      }
  }
</mx:Application>
```

Listing 6.2: After initialisation, the application establishes an interface to its container and allows it to call the getDataObjects function. It then calls the callbackReady function of the container to signal that initialisation is completed.

```
// Javascript file
function callbackReady() {
  getFlexApp('linechart').getDataObjects(vis.headers, vis.data,
                                        vis.options, vis.colours);
}

function getFlexApp(appName) {
    return document[appName];
}

// Flex Gadget
public function getDataObjects(dataHeaders:Array, dataValues:Array,
                              dataOptions:Array, dataColours:Array):void {
  // store and process the data
  ...
  drawVisualisation();
}
```

Listing 6.3: After establishing an interface to the container, a function of the container is called which hands the data to the Flex gadget.

```
if(this.containerElement != null) {
  var url = voronoi.swf';
  var options = {id: "voronoi", width: options.chartWidth, height: options.
      chartHeight, allowScriptAccess: "always"};
  gadgets.flash.embedFlash(url, "chart", "10.0.0", options);
}
```

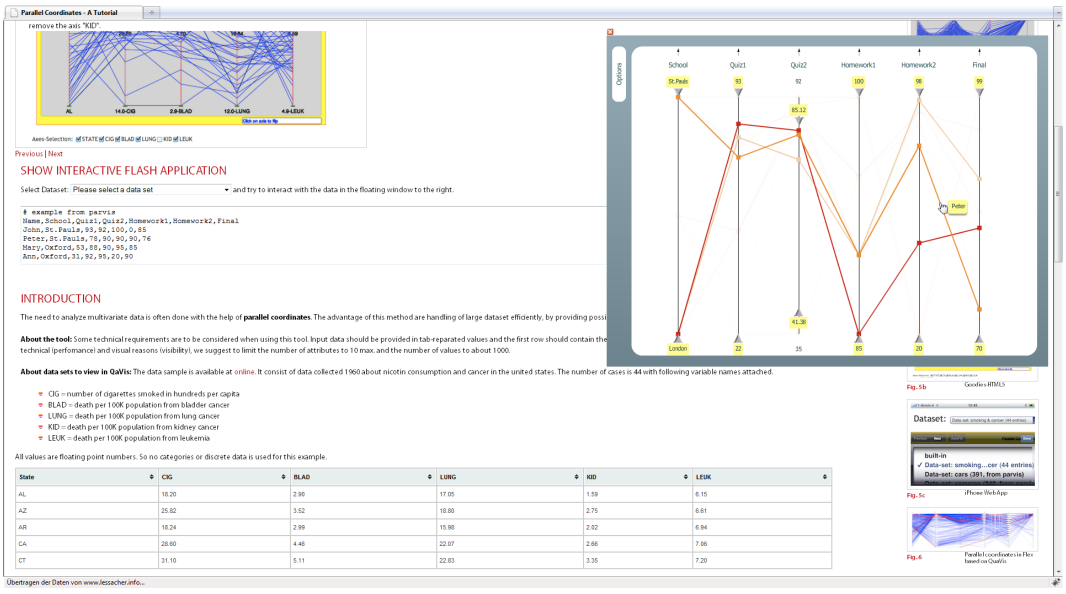
Listing 6.4: Google offers a function to include Flash-based objects into a page.

Spreadsheet data, so-called gadgets can be inserted. These gadgets are HTML and JavaScript applications and thus it is easy to embed SWF objects into the gadget.

The data transfer to the Flex gadget is done as described in Section 6.1.1. The difference is that the Flash object is included using one of Google's functions and the data has to be acquired from the Google Spreadsheets table first. As seen in Listing 6.4, Google offers a special function to include flash objects. This function is accessed by the `gadgets.flash.embedFlash` command and needs Google's flash package to be included (see line 8 in Listing 6.5). To acquire the data from the spreadsheet, a special Google Gadget XML file needs to be created. The gadget XML file is divided into three different parts:

- **Gadget Preferences** ($\langle ModulePrefs \rangle$) This section specifies the characteristics of the gadget, such as title or author.
- **Content Section** ($\langle Content \rangle$) The application code is located in this section.
- **User Preferences** ($\langle UserPref \rangle$) User preferences provide controls to specify the gadget's settings. For example, all initial settings for the gadget chosen when adding the gadget.

To obtain the data from a Google Spreadsheet, it is necessary to query the data using Google's GadgetHelper class. An example of a simple data request and the structure of a Google Gadget XML file



remove the axis "KID".

Axes-Selection: [✓ STATE] [✓ CIG] [✓ BLAD] [✓ LUNG] KID [✓ LEUK]

Previous | Next

SHOW INTERACTIVE FLASH APPLICATION

Select Dataset: Please select a data set and try to interact with the data in the floating window to the right.

```
# example from parvis
Name,School,Quiz1,Quiz2,Homework1,Homework2,Final
John,St.Pauls,93,92,100,0,85
Peter,St.Pauls,78,90,90,90,76
Mary,Oxford,53,88,90,95,85
Ann,Oxford,31,92,95,20,90
```

INTRODUCTION

The need to analyze multivariate data is often done with the help of **parallel coordinates**. The advantage of this method are handling of large dataset efficiently, by providing possi

About the tool: Some technical requirements are to be considered when using this tool. Input data should be provided in tab-reparated values and the first row should contain the technical (perfomance) and visual reasons (visibility), we suggest to limit the number of attributes to 10 max. and the number of values to about 1000.

About data sets to view in QaVis: The data sample is available at online. It consist of data collected 1960 about nicotin consumption and cancer in the united states. The number of cases is 44 with following variable names attached.

- CIG = number of cigarettes smoked in hundreds per capita
- BLAD = death per 100K population from bladder cancer
- LUNG = death per 100K population from lung cancer
- KID = death per 100K population from kidney cancer
- LEUK = death per 100K population from leukemia

All values are floating point numbers. So no categories or discrete data is used for this example.

State	CIG	BLAD	LUNG	KID	LEUK
AL	18.20	2.90	17.05	1.59	6.15
AZ	25.82	3.52	18.80	2.75	6.61
AR	18.24	2.99	15.98	2.02	6.94
CA	28.60	4.46	22.07	2.66	7.06
CT	31.10	5.11	22.83	3.35	7.20

Übertragen der Daten von www.lessacher.info...

Figure 6.2: This screenshot shows an interactive parallel coordinates tutorial created by Bauer et al. [2009]. The page uses the Liquid Diagrams parallel coordinates visualisation to explore various data sets.

is shown in Listing 6.5.

6.1.3 Embedding a Gadget into a HTML Page

Another possibility to hand data to a Liquid Diagrams gadget is to use a HTML page as data source. Such a HTML page hosts controls to import data from files or enables to manually enter data using editors. Using JavaScript the data is handed to the Liquid Diagrams gadget to visualise it.

An example for such a HTML page is the page created by Bauer et al. [2009] during a lecture project. The page is an interactive tutorial which describes the usage of parallel coordinates to analyse data. Part of this tutorial page is the feature to load datasets and explore them using the Liquid Diagrams parallel coordinates visualisation (see Figure 6.2).

Listing 6.6 illustrates how the parallel coordinates tutorial adds the Liquid Diagrams parallel coordinates gadget to the page. Therefore it initialises the options array (lines 1-10) according to the parameters required by the parallel coordinates gadget. Afterwards it creates the gadget (lines 12-15) which calls the `callbackReady` function as soon as the gadget is ready to receive the data. This `callbackReady` function then hands the data to the gadget to visualise it (lines 18-20).

6.2 Framework Structure

To incorporate new functions and features into the framework, the file architecture used in Lessacher [2009b] was modified and extended according to Figure 6.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
  <ModulePrefs title="Voronoi"
    description="Voronoi visualisation written in Flex 4.0"
    author="Martin Lessacher"
    author_affiliation="Graz University of Technology"
    author_email="lesse@sbox.tugraz.at">
    <Require feature="flash"/>
  </ModulePrefs>

  <UserPref name="_table_query_url" display_name="Data source URL"    required="
      true"/>

  <Content type="html"><![CDATA[
    <form>
      <div id="chart"></div>
    </form>
    <script type="text/javascript" src="http://www.google.com/jsapi"></script>
    <script type="text/javascript">
      google.load("visualization","1");
      google.setOnLoadCallback(initialize);

      var prefs = new _IG_Prefs();

      function initialize() {
        gadgetHelper = new google.visualization.GadgetHelper();
        var query = gadgetHelper.createQueryFromPrefs(prefs);
        query.send(handleQueryResponse);
      }

      function handleQueryResponse(response) {
        if (!gadgetHelper.validateResponse(response)) {
          return;
        }

        var data = response.getDataTable();
      }
    </script>
  ]]></Content>
</Module>
```

Listing 6.5: A Google Gadget XML file, necessary to receive data from a Google Spreadsheet table.

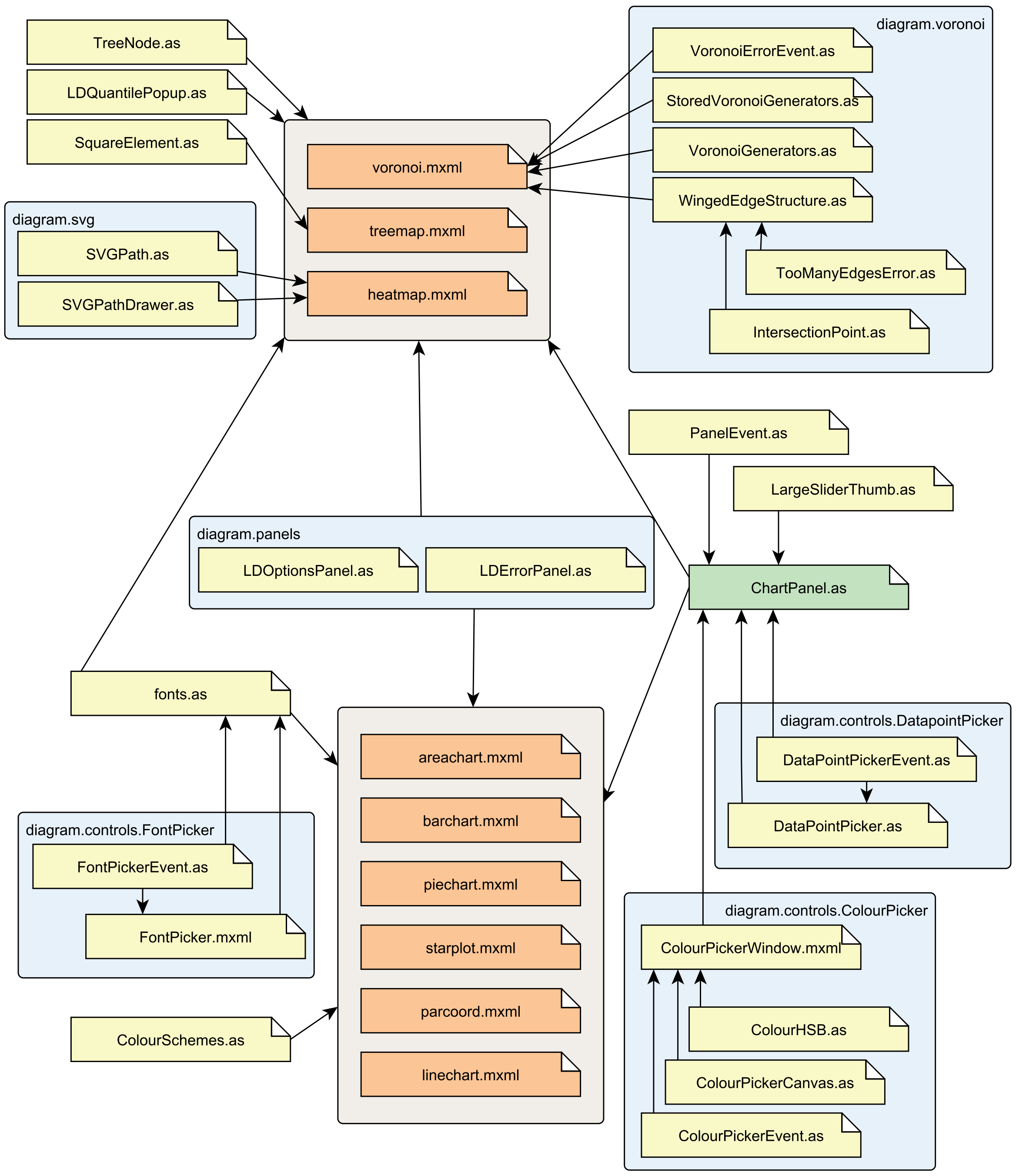


Figure 6.3: The file architecture of the Liquid Diagrams framework. Dependencies are shown using arrows. The arrow points to a file including or using functions of the file at the start of the line. If an arrow points to a container all the files in the container use the file's contents. Blue rectangles represent packets. The orange files are the main files of each visualisation, which are compiled into SWF objects.

```
Parcoord.prototype.draw = function(data,titles,types) {
  this.data = data;
  this.headers = titles;

  this.options = new Array(22);
  this.options[0] = 800;
  this.options[1] = 600;
  this.options[2] = true;
  this.options[3] = ”No Legend”;
  ...

  if(this.containerElement != null) {
    var url = host + ’flash/parcoord.swf’;
    swfobject.embedSWF(url, ”chart”, options.chartWidth, options.chartHeight,
        ”10.0.0”);
  }
}

function callbackReady() {
  getFlexApp(’chart’).getDataObjects(vis.headers, vis.data, vis.options, ”Many”);
}
```

Listing 6.6: The code illustrates how the tutorial page handles the creation of the Liquid Diagrams parallel coordinates gadget and how data is handed to the gadget.

6.2.1 ChartPanel

The ChartPanel class implemented in `ChartPanel.as` is the main class of the Liquid Diagrams framework. This class is derived from the Flex standard Canvas class thus it can be used as a panel component in visualisations.

The ChartPanel class also offers many functions to add content to its panel, including drawing axes, titles, legends, and various graphical shapes (for example lines, circles, paths, and rectangles). The ChartPanel class manages effects like mouseover and click effects for elements drawn in its panel. For a more detailed explanation of the core functions of the ChartPanel class, see Lessacher [2009b].

6.2.2 Visualisation Files

Each file coloured orange in Figure 6.3 is an MXML file containing the implementation of a specific type of visualisation. Each file uses the ChartPanel class as the main stage to host its visualisations content.

The legend and diagram titles are drawn using ChartPanel functions. Initialisation of the ChartPanel is done in the `drawVisualisation` function of each MXML file. The drawing logic for each visualisation is located in its `drawData` function. The algorithm to draw the visualisation is a call composition of shape creating functions offered by the ChartPanel class. The MXML file of each visualisation is compiled into an executable SWF object by the Flex SDK. This SWF object is the main file needed to create a visualisation.

```
var myButton:Button = new Button();
myButton.addEventListener(MouseEvent.CLICK, buttonClicked);

private function buttonClicked(event:MouseEvent) {
  ...
}
```

Listing 6.7: The creation of a button control and the registration of the click-event for it. Every time a click-event happens on the button, the function buttonClicked is called.

6.2.3 Events

Events are fired to let the developer know that something special happened within an application. This can include interaction with devices (keyboard, mouse, ...) as well as special states that are reached by the application. Events are usually used to identify changes and react properly to them. This can also involve multiple applications which partially dependent on events fired by each other.

There are pre-defined events for interactions like a mouse-click. The desired event needs to be registered by a component using the `addEventListener` function. This event is then linked to a function, which is called when the event happens. An example for this default event handling is shown in Listing 6.7.

The ChartPanel class features its own event class implemented in the `PanelEvent.as` file. This Pane lEvent is derived from the standard Flex Event class and features an additional variable `eventParameter` which is used to capture additional information about the event (for examaple the entity number of the entity which triggered the event). To fire an event of this PanelEvent class, an instance of this type needs to be created and then fired by using the `dispatchEvent` function. That is all the ChartPanel class needs to do. To process this event, the visualisation gadget needs to register the event to a function. This function is called and processed every time the event is fired.

The ChartPanel class fires three different events which can be caught by gadget visualisations including a ChartPanel object:

- **redrawAll:** This event is fired when all entities need to be redrawn. This is usually the case when the values the visualisation depends on are changed. For example, if a data entity in a pie chart is hidden all other entities and thus the whole visualisation need to be redrawn.
- **redrawEntity:** When firing a `redrawEntity` event, only one entity needs to be redrawn. The specific entity number is therefore included in the event. For example, only the specific entity needs to be redrawn if its colour is changed during runtime. This does not affect any other elements.
- **cutOffWarningEvent:** The `cutOffWarning` event is fired when there is not enough space to display all entities in a visualisation. This happens, for example, if a bar chart does not use the `fitSize` parameter and defines an entity size which would be too large to display. Thus a `cutOffWarning` event will be fired and not all data sets shown. How the visualisation gadget handles this event is up to the gadget. The current implementation will display a large warning in the visualisation area that not all data sets are shown.
- **Colour Legend Events:** These events are fired to the visualisation gadgets using the colour legend. Additional information about these events can be found in Section 6.3.2.
- **clickedEntity:** The `clickedEntity` event is fired when the user clicks an data entity drawn in

the visualisation. Depending on the visualisation gadget the reaction to the mouse-click can be handled differently.

Besides many mouseover and click events of single components included in the visualisation there are several other custom events which are listened to by the ChartPanel class:

- **Drag Events:** The `DragEnter`, `DragExit`, and `DragDrop` events are caught by the ChartPanel class and handled to implement the drag-and-drop functionality.
- **Colour Picker Events:** The Korax colour picker control features some events to let the calling application know what changes are done after calling a colour chooser instance. The `change` event is fired if a colour was changed and confirmed by hitting `Ok` button. The chosen colour can then be extracted from the event and the appropriate changes in the visualisation made. There is also a `cancel` event sent by the colour picker if the `Cancel` button is clicked, but this event does not affect the visualisation because the changes are not applied.
- **DataPoint Picker Events:** The data point picker events are like colour picker events, but feature other event variables than the chosen colour (data point type and colour).
- **Font Picker Events:** These events are used to inform the ChartPanel class about changes made in the font picker.

6.2.4 Pickers

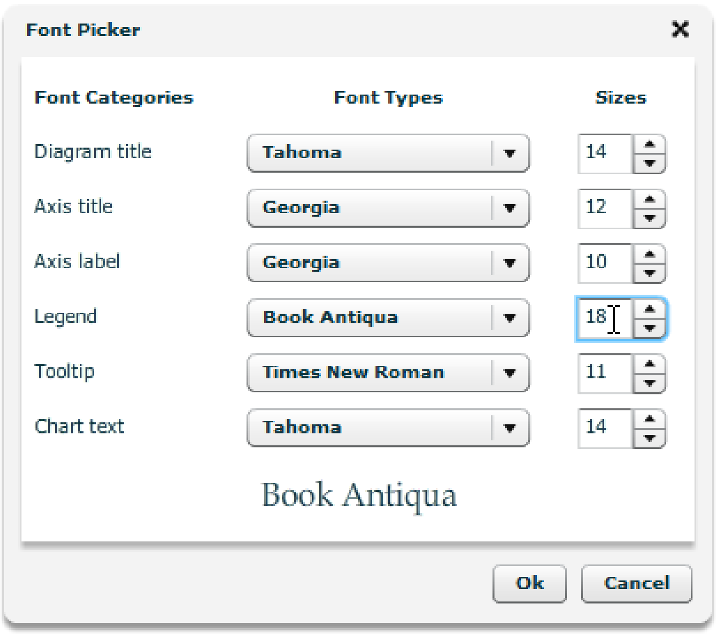
Since the standard Adobe Flex colour picker lacks some basic functions and only offers a small amount of available colours, a different colour picker implementation is used in the Liquid Diagram framework. The new colour picker implementation was created by Nuzha [2010] and was slightly modified to fit into the framework. The implementation of the colour picker is located in the package `diagram.controls.ColourPicker`.

The data point picker is used in line-based visualisations. By clicking on the currently assigned data point the data point picker pops up and enables the user to select a different data point representation from 8 different types. The files located in the `diagram.controls.DatapointPicker` package shown in Figure 6.3 contain the implementation of the data point picker.

All the fonts in a visualisation can be changed using the font picker implemented in the `diagram.controls.FontPicker` package. The font picker offers all the fonts declared in `fonts.as` (see Section 6.2.6). The font picker enables the user to select different font types and font sizes for specific text categories, as shown in Figure 6.4. The font picker component and is events are implemented in the `diagram.controls.FontPicker` package.

6.2.5 Colour Schemes

In the Liquid Diagrams framework introduced in Lessacher [2009b] colour schemes were presented. These colour schemes enabled the user to choose among different colour combinations prior to the creation of the visualisation. The chosen combination was used as the data entities colour in the visualisation. The colour schemes were only selectable in the GoogleDocs gadget options and were implemented using JavaScript.


Font Picker

Font Categories	Font Types	Sizes
Diagram title	Tahoma	14
Axis title	Georgia	12
Axis label	Georgia	10
Legend	Book Antiqua	18
Tooltip	Times New Roman	11
Chart text	Tahoma	14

Book Antiqua

Figure 6.4: The font picker allows the user to change the font type and font size of specific text categories. It also has a preview of the last chosen font type and font size at the bottom of the font picker window.

The new implementation of the ColourSchemes class enables the selection of pre-defined colour combinations independently of GoogleDocs and during runtime using the options panel of the visualisation. The implementation of this class is located in the `ColourSchemes.as` file.

The available colour schemes can be modified by editing the `colour-schemes.xml` file located in the `common` folder of the visualisations home directory. This file contains all the colour schemes and can be modified and extended using XML code. An example for a `colour-schemes.xml` can be seen in Listing 6.8. New schemes can be added by inserting a new `color` tag using the name of the scheme as its `id`. The colours of this scheme are represented by space separated hexadecimal colour codes assigned to the `name` tag of the `color` element.

The ColourSchemes class distinguishes between two different types of colour schemes. While normal colour schemes contain more than one colour, monochrome colour schemes only have one colour. Monochrome schemes are intended to be used with the parallel coordinates or star plot visualisation. Choosing a monochrome scheme turns off the legend and reduces the entity lines alpha value. By reducing the lines alpha value overlapping lines can be seen better because their alpha values are multiplied and thus the resulting line is drawn using a higher alpha value.

6.2.6 Defining Fonts

The `fonts.as` file is included in all visualisations and handles calls of the font picker and all corresponding events after clicking the `ChooseFonts...` button. This file also defines the font types available in the font picker.

To add a new font it has to be either a TrueType or OpenType font. As shown in Listing 6.9, some appropriate embed code has to be added to the `fonts.as` file. Afterwards the font can be referred in the entire project using the declared `fontName`. For the font to appear in the font picker component, it has to

```
<data>
  <schemes id="schemes">
    <color id="Dolas Break" name="#C92A18 #EC4115 #F59300 #005E79 #0A2F4A"/>
    <color id="Black" name="#000000"/>
    <color id="Blue" name="#005C81"/>
    <color id="Gore" name="#BF8266 #C2B195 #96866F #EDC389 #ADB292"/>
    <color id="Happy Mango" name="#F06060 #A8C078 #F0D878 #F04848 #F0C048"/>
    <color id="Like Fire" name="#FFDD00 #FF8000 #FF4D00 #FF0000 #AD0202"/>
    <color id="Peace" name="#EB6B34 #EB8F34 #FFBA75 #F0CCA8 #8F7256"/>
  </schemes>
</data>
```

Listing 6.8: The ColourSchemes.as file contains colour scheme definitions. Each color tag represents a different scheme. New schemes and modifications can be done by altering the xml code. The colour schemes defined here are taken from COLOURlovers [2010].

```
  [Embed(source='/font/tahoma.ttf',
    embedAsCFF='false',
    fontName='Tahoma',
    mimeType='application/x-font'
  )]
  private var font1:Class;
]
```

Listing 6.9: The code necessary to embed a new font (Tahoma) into the application. The font then can be referred to in the application using the fontName parameter.

be added to the font picker combo boxes using its `fontName`.

6.2.7 Panels and Pop-Ups

Panels are space-saving components which are used to hide feedback and additional controls until the user wants to make use of them. The implementation files of panels are located in the `diagram.panels` package. The Liquid Diagrams framework offers two basic types of panels:

- **Options Panel**
  This panel hosts several other control elements like buttons, text boxes, and combo boxes. These controls can change the look of the visualisation or export the visualisation as a vector graphic or an image. The options panel class is implemented in the `LDOptionsPanel.as` file.

- **Error Panel**
  An error panel does not contain other controls like the options panel. Instead it is used to give feedback like error messages or warnings to the user. Thus the error panel can be seen as a kind of feedback log. The implementation of the error panel class is located in the `LDErrorPanel.as` file.

For more information about panels and how they are implemented, see Section 6.3.1.

6.2.8 SVG Parser

One of the visualisations offered by the Liquid Diagrams frameworks is the heatmap. The heatmap gadget is based on SVG maps which define borders and areas of regions. To be able to perform computations on the the XML data given by the SVG maps, a SVG parser is needed.

The implementation of this parser is located in the `diagram.svg` packet and was originally written by Knapitsch et al. [2009]. Only a few adaptations were necessary to integrate the parser into the Liquid Diagrams framework. More information about the SVG parser can be found in Section 8.4.

6.2.9 Tree Structure

In the Liquid Diagrams framework some visualisations (heatmap, treemap, and voronoi treemap) are based on hierarchical datasets. To make the handling of hierarchical data more comfortable and enable features like zooming between the hierarchical levels, the dataset is stored in a special tree data structure implemented in the `TreeNode.as` file. More details about the TreeNode class and its implementation can be found in Section 8.2.

6.2.10 Voronoi

The files located in the package `diagram.voronoi` are used to implement the functions and algorithms necessary to create a voronoi diagram. To build up a voronoi structure, a special data structure taken from Okabe et al. [2000] is used. The implementation of this data structure is located in the `WingedEdgeStructure.as` file which is the most important file of the voronoi implementation, because it contains all the algorithms and computation logic.

6.3 Framework Components

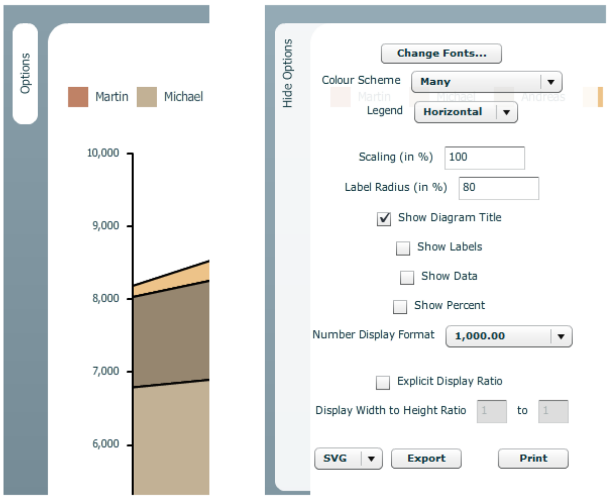
6.3.1 Side Panels

In Lessacher [2009b] all Liquid Diagrams visualisations offered an option to show an Options Panel when inserting the gadget. If the panel was shown, it was placed on the left side of the visualisation and could not be turned off. The Options Panel enabled visual aspects of the visualisation to be set during runtime, but also took up mouch of the available width (up to 300 pixels). Since it could not be turned off, it was a serious limitation in environments with low screen resolution.

To fix this issue the panels of the Liquid Diagrams framework have been revised. The options and error panels are in a space-saving state on the left side of the visualisation panel. If the panel is clicked it is opened to its full size, overlapping the visualisation as shown in Figure 6.5. When the hide panel handle is clicked, the panel closes to its space saving state again.

The LDOptionsPanel and LDErrorPanel classes are both derived from the standard Flex Canvas class. The Canvas class is a layout container which defines a rectangular region in which control elements and other container elements can be placed. This is the ideal prerequisite for the options panel, which hosts numerous control elements like buttons, combo boxes, and check boxes.

Listing 6.10 illustrates how an Options Panel can be added to an existing visualisation. First of all, a new control element of type LDOptionsPanel has to be added (line 1) in the visualisation's MXML

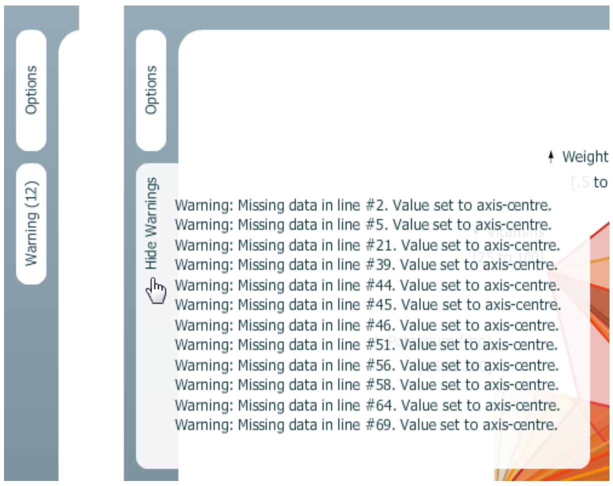


Colour Scheme: Many
Legend: Horizontal
Scaling (in %): 100
Label Radius (in %): 80
[✓] Show Diagram Title
[ ] Show Labels
[ ] Show Data
[ ] Show Percent
Number Display Format: 1,000.00
[ ] Explicit Display Ratio
Display Width to Height Ratio: 1 to 1

Figure 6.5: The left side of the image shows the options panel while in hidden state. After clicking the panel an animation which resizes the panel is triggered. After the panel reaches its full size the animation is stopped and the components are added and accessible.

```
<panels:LDOptionsPanel id="optionsPanel" left="10" top="20" backgroundColor="0xFFFFFF" clipContent="false" cornerRadius="10" borderStyle="solid" borderThickness="0">
  <mx:VBox id="controlBox" width="100%" top="20" bottom="20" horizontalAlign="center" verticalAlign="middle" visible="false" verticalGap="6">
    <mx:Button id="fonts" label="Change Fonts..." click="onFonts()"></mx:Button>
    <mx:CheckBox id="legend" label="Show Legend" change="onRedraw()"/>
    <mx:CheckBox id="percentage" label="Percent" change="onRedraw()"/>
  </mx:VBox>
</panels:LDOptionsPanel>

<mx:Script>
  <![CDATA[
    ...
    public function initializeOptionsPanel() : void {
      optionsPanel.setMaxDimensions(this.width, this.height);
      optionsPanel.assignControlBox(controlBox);
    }
    ...
  ]]>
</mx:Script>
```

Listing 6.10: A visualisation's MXML file, which uses the LDOptionsPanel class to provide an Options Panel.



Warning: Missing data in line #2. Value set to axis-centre.
Warning: Missing data in line #5. Value set to axis-centre.
Warning: Missing data in line #21. Value set to axis-centre.
Warning: Missing data in line #39. Value set to axis-centre.
Warning: Missing data in line #44. Value set to axis-centre.
Warning: Missing data in line #45. Value set to axis-centre.
Warning: Missing data in line #46. Value set to axis-centre.
Warning: Missing data in line #51. Value set to axis-centre.
Warning: Missing data in line #56. Value set to axis-centre.
Warning: Missing data in line #58. Value set to axis-centre.
Warning: Missing data in line #64. Value set to axis-centre.
Warning: Missing data in line #69. Value set to axis-centre.

Figure 6.6: The error panel is to the left of the visualisation and is used to give feedback to the user. When closed it displays the type and amount of feedback. By clicking, it is opened to its full size and the feedback is shown in full..

file. Afterwards, a grouping element of the Flex standard type VBox named `controlBox` is assigned to the Options Panel (line 2). The VBox class is a layout container which lays out its children in a single vertical column. All controls added as children of the `controlBox` element are shown in the Options Panel (lines 3-5). The next step is to initialise the Options Panel, which can be done anywhere in the code, but has to be done before the Options Panel is opened for the first time. The LDOptionsPanel class is given the maximum available width and height (line 13). Finally, the `controlBox` element is handed to the LDOptionsPanel class to enable the show and hide operations on the control elements from within the LDOptionsPanel class (line 14).

While the Options Panel hosts control elements, to change the visual appearance of the visualisation, the Error Panel is intended to give feedback to the user (see Figure 6.6). Three different levels of feedback are supported by the Error Panel: information, warnings, and errors. To add an Error Panel, an element of type LDErrorPanel has to be added to the visualisation's MXML file, just like when adding the Options Panel in line 1 of Listing 6.10. Unlike the Options Panel, there is nothing more to initialise. Unless there is feedback available, the Error Panel stays hidden. The `addNewErrorTextLine`, `addNewWarningTextLine`, and `addNewInfoTextLine` functions of the LDErrorPanel class will add a new text line of the appropriate type to the Error Panel. The Error Panel can be reset by calling the `resetErrorPanel` function of the LDErrorPanel class.

Both panels make use of the standard Flex Resize class to implement the pop-out animation, when the user clicks on the panel. Listing 6.11 illustrates how this resize effect is implemented. The panel is assigned a label which implements the `MouseEvent.CLICK` event in order to call the `onEscalateClick` function when clicked (lines 3-5). Depending on the current state (`STATE_HIDDEN` or `STATE_ESCALATED`) of the panel, the new dimensions of the panel are set (15-16 and 21-22). Using the `play` function of the Resize class, the resize effect is performed(line 26).

6.3.2 Colour Legend

In the original Liquid Diagrams visualisations [Lessacher, 2009b] and in the star plot visualisation, all entities are assigned a colour value which is used to draw and characterise the entity in the visualisation. This colour value can be changed either manually by clicking on the colour spot of the entities label in

```
public class LDOptionsPanel extends Canvas {
  ...
  optionPanelLabel = new Text();
  optionPanelLabel.text = "Options";
  optionPanelLabel.addEventListener(MouseEvent.CLICK, onEscalateClick);
  ...
}

public function onEscalateClick(event:MouseEvent) : void {
  var resizeEffect:Resize = new Resize();
  resizeEffect.target = this;

  if(state == STATE_HIDDEN) {
    optionPanelLabel.text = "Hide Options";
    resizeEffect.widthTo = 320;
    resizeEffect.heightTo = this.controlBox.height + 40;
    state = STATE_ESCALATED;
    this.setStyle("backgroundAlpha", 0.9);
  } else {
    optionPanelLabel.text = "Options";
    resizeEffect.widthTo = hiddenWidth;
    resizeEffect.heightTo = hiddenHeight;
    state = STATE_HIDDEN;
    this.controlBox.visible = false;
  }
  resizeEffect.play();
}
```

Listing 6.11: Illustration of the LDOptionsPanel.as file showing how the resize animation of the Options Panel and Error Panel is implemented.

the legend or by changing the colour scheme used in the visualisation which leads to a automatic colour assignment to each entity (more information on colour schemes can be found in Section 6.2.5).

This type of colour assignment cannot be used to colour the entities of the heatmap, treemap, and voronoi diagram visualisations, because their entities' colours depend on a specified data value of these entities. Thus another type of legend was implemented. Instead of linking a colour value to each entity, the colour is computed for each entity depending on the specified data value of the entity and the colour distribution type. There are three different colour distribution types available in the Liquid Diagrams framework:

- **Uniform Distribution**
  When set to uniform distribution, the legend displays a specified number of colour bins. Each colour bin is assigned a colour and a unique value range. The beginning and the end of the range is computed by dividing the value's range span by the number of colour bins. Each entity is then assigned the colour under which colour bin's range the entity's value falls.

  For example, in Figure 6.7b if the legend range span is from 0 to 100,000,000 resulting in a span of 100,000,000. This span divided by 10 colour bins would result in 10 bins, each of step size 10,000,000.

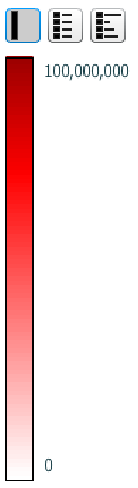
- **Continuous Distribution**
  Continuous colour distribution does not show colour bins. Instead, it shows a colour gradient resulting in more precise entity colours. Internally, the gradient is built up using 100 colour bins. Each colour bin is represented by a 2-pixel stripe in the gradient (shown in Figure 6.7a).

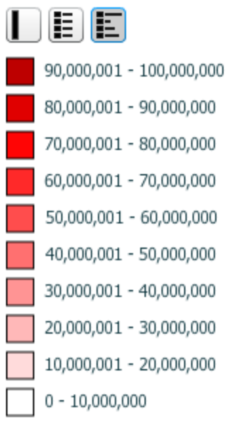
- **Quantile Distribution**
  Like the uniform distribution, quantile distribution also shows colour bins. The difference between them is how the entities are distributed among the bins. In contrast to uniform distribution, the value range of the colour bins is directly influenced by all the entities values. The number of entities is evenly spread among the number of bins. This results in nearly the same number of entities represented by each colour bin, while in some cases of uniform distribution single colour bins may not contain by any entities at all.

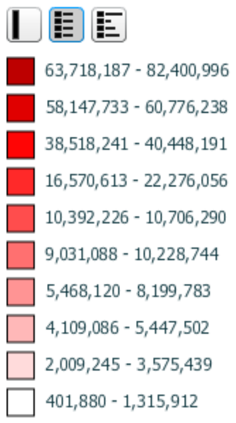
  For example, if a data set contains 30 entities and the number of colour bins is set to 10 each colour bin would host 3 entities. The value span of a bin would thereby be determined by the lowest and the highest values of the 3 represented entities. Figure 6.7c illustrates how a quantile distribution with 10 colour bins would look.

When using uniform or quantile distribution, bins are shown to display the different colours of the data. The maximum number of colour bins (classes) can be set using the `Numberofbins` field in the Options Panel. There is also a faster way to set the number of colour bins when using the quantile distribution. On clicking the quantile distribution button, a small selection window (shown in Figure 6.8) pops up. This window offers pre-defined quantile classes to choose from: including quartiles (4), quintiles (5), octiles (8), deciles (10), and duodeciles (12). The pop up window is derived from the Flex Canvas class and implemented in the `LDQuantilePopup.as` file.

The colour distribution buttons add a two new events to the ChartPanel class. The first event is fired when either continuous or uniform distribution is chosen and is called `changedColourDistribution`.

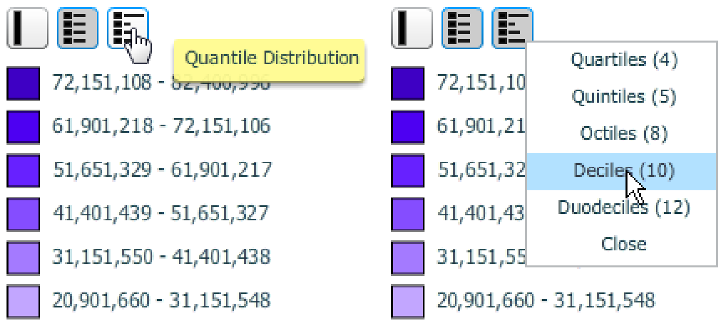



90,000,001 - 100,000,000
80,000,001 - 90,000,000
70,000,001 - 80,000,000
60,000,001 - 70,000,000
50,000,001 - 60,000,000
40,000,001 - 50,000,000
30,000,001 - 40,000,000
20,000,001 - 30,000,000
10,000,001 - 20,000,000
0 - 10,000,000


63,718,187 - 82,400,996
58,147,733 - 60,776,238
38,518,241 - 40,448,191
16,570,613 - 22,276,056
10,392,226 - 10,706,290
9,031,088 - 10,228,744
5,468,120 - 8,199,783
4,109,086 - 5,447,502
2,009,245 - 3,575,439
401,880 - 1,315,912

(a) Continious colour distribution. **(b)** Uniform colour distribution. **(c)** Quantile colour distribution.

Figure 6.7: The colour legend offers three different types of colour distribution functions: continious, uniform, and quantile.


Quantile Distribution
72,151,108 - 82,400,990
61,901,218 - 72,151,106
51,651,329 - 61,901,217
41,401,439 - 51,651,327
31,151,550 - 41,401,438
20,901,660 - 31,151,548

72,151,10
61,901,21
51,651,32
41,401,43
31,151,55
20,901,660 - 31,151,548

Quartiles (4)
Quintiles (5)
Octiles (8)
Deciles (10)
Duodeciles (12)
Close

Figure 6.8: When clicking on quantile distribution a small selection window pops up enabling the user to set the number of colour bins on the fly.

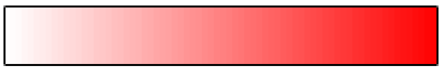
The second event `changedColourDistributionQuantile` is fired when the quantile distribution is chosen. Both events need to be caught by the ChartPanel hosting application in order to ensure consistency between the settings shown in the Options Panel and the displayed legend.

The range of each colour class is automatically computed using the minimum and maximum of all data values of a specific data column. Other than computing the classes based on the minimum and maximum there is also the possibility to manually enter the range values. The first way to change the boundary values is to set the `SetRangesManually` option in the option panel and enter the new values into the `LowestValue` and `HighestValue` text boxes. Another way to change the boundary values is to click on the value displayed next to the first or last colour bin, which will open a text box to enter a new boundary value. After the change is confirmed by pressing the return key, the new class ranges will be determined and displayed.

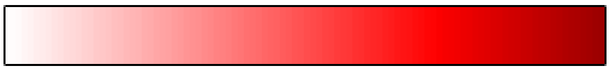
Since the implementation of the legend is located in the ChartPanel class, a notification to the ChartPanel hosting application is necessary in order to initialise a redraw according to the new boundaries. To accomplish that, two additional events have been added. The `changedLowestValue` and the `changedHighestValue` events are fired to the hosting application and need to be implemented there.

```
 1 public function setColour(color:uint, count:int) : Array {
 2   var hsbColor:ColorHSB = new ColorHSB();
 3   hsbColor = ColorHSB.rgb_to_hsb(color);
 4   var indexFactor:Number = (140 / count);
 5
 6   colors = new Array();
 7   for(var i:int = 0; i < count; i++) {
 8     hsbColor.s = Math.min(indexFactor * i, 100);
 9     var bValue:Number = Math.max((indexFactor * i) - 100, 0);
10     hsbColor.b = 100 - bValue;
11     colors[i] = ColorHSB.hsb_to_rgb(hsbColor);
12   }
13   return colors;
14 }
```

Listing 6.12: Computing the colour values for each colour bin shown in the legend.



(a) 100 different colours (each 1 pixel) based on the colour with the hexadecimal value #FF0000. Each colour posesses the same hue and brightness value. Only the saturation value of the colours has been modified (from 0 to 100).



(b) 140 colours (each 1 pixel) are also generated from the hexadecimal value #FF0000. The difference is that the 40 additonal colours have been added. These colours are a result of reducing the brightness from the value of 100 to 60.

Figure 6.9: Different colour tones based on the colour #FF0000.

The user can set the main colour by clicking on one of the colour bins in the legend. This opens the colour picker dialog to choose another colour. After the colour is chosen the new colours for the colour bins are computed.

The computation of the colours used for the bins is shown in Listing 6.12. The `setColour` function is handed the new colour value and the number of colour bins to display (line 1). First, the colour is translated into its corresponding value in the Hue, Saturation, Brightness (HSB) colour model (lines 2-3). The hue parameter determines the colour type and therefore remains the same. To obtain several other colour tones of the colour specified by the hue value, the saturation and brightness parameters are varied. As shown in Figure 6.9a, changing the saturation value only would result in not having darker tones of the colour. Therefore, the brightness value is also altered, to add darker tones of the specified colour type (see Figure 6.9b).

The brightness values of the taken colours start at 100 and are continuously reduced turning the colours more and more into black. To prevent the colour becoming too dark the brightness is only reduced to the value 60 resulting in 40 different colours (from 100 to 60). Including the 100 colours given by the saturation, there are now 140 different colours available based on a given hue (illustrated in Figure 6.10). The variable `indexFactor` in Listing 6.12 determines the span between the colours that are taken (line 4). For example if there are 12 colour bins only every eleventh colour is taken (140 divided by 12). The brightness and saturation values of the taken colours are then computed (lines 7-12) and written into an array. Finally, the array containing the computed colours is returned (line 13).

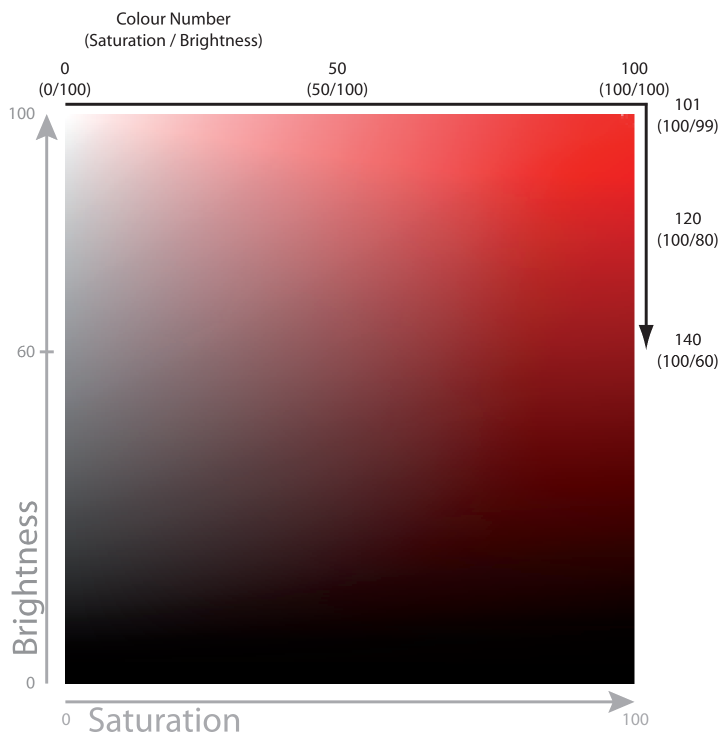


Figure 6.10: Computing the different colour tones of a specified colour value. The 140 different colour values are along the top and right edge of the colour scale (indicated by the black arrow).

The alignment of the colour legend can be set using the `Legend` combo box the Options Panel. The colour legend supports two different alignments:

- **Horizontal**
  When `horizontal` is chosen the legend is drawn on top of the diagram. Due to the limited horizontal space between the colour bins, only the first and the last bin are assigned a range label. All other bin ranges are only shown when moving the mouse over the specified bin.

- **Vertical**
  If `vertical` alignment is selected, the legend is drawn to the left of the diagram. Every bin range is drawn to the right of each colour bin. Therefore, a `vertical`legend takes more space than the `horizontal`.

The implementation of the legend is located in the `ChartPanel.as` file. Two different functions are available to draw either the vertical or horizontal colour legend: `drawVerticalColourLegend` and `drawHorizontalColourLegend`. Listing 6.13 illustrates how to include this type of colour legend into a visualisation. The colours of the legend bins are computed by calling the `setColour` function (line 5). Afterwards the legend can be created by calling `drawHorizontalColourLegend` or `drawVerticalColourLegend` (lines 7-11). As already mentioned, there are some events that need to be implemented by the visualisation application to support the change of the colour distribution. Therefore, the necessary events are registered (lines 44-45) and linked to the appropriate functions (lines 15-40).

```
<mx:Script>
 <![CDATA[
   public function drawVisualisation() : void {
     ...
     colours = chartPanel.setColour(colourType, options[COLOUR_COUNT]);

     if(options[SHOW_LEGEND] == "Horizontal") {
       chartPanel.drawHorizontalColourLegend(...);
     } else if(options[SHOW_LEGEND] == "Vertical") {
       chartPanel.drawVerticalColourLegend(...);
     }
     ...
   }

   private function changedQuantileDistribution(event:PanelEvent) : void {
     if(event.eventParameter == 0) {
       colour_text.text = options[COLOUR_COUNT] = 4;
     } else if(event.eventParameter == 1) {
       colour_text.text = options[COLOUR_COUNT] = 5;
     } else if(event.eventParameter == 2) {
       colour_text.text = options[COLOUR_COUNT] = 8;
     } else if(event.eventParameter == 3) {
       colour_text.text = options[COLOUR_COUNT] = 10;
     } else if(event.eventParameter == 4) {
       colour_text.text = options[COLOUR_COUNT] = 12;
     }
     event.eventParameter = 1;
     changedDistribution(event);
   }

   private function changedDistribution(event:PanelEvent) : void {
     if(event.eventParameter == 0) {
       legendType = "Continious";
     } else if(event.eventParameter == 1) {
       legendType = "Percentile";
     } else {
       legendType = "Uniform";
     }
     redrawVisualisation();
   }
 ]]>
 </mx:Script>

<diagram:ChartPanel id="chartPanel" width="100%" height="100%" borderStyle="solid"
     borderThickness="0" changedColourDistribution="changedDistribution(event)"
    changedDistributionQuantile="changedQuantileColorDistribution(event)"
 </diagram:ChartPanel>
```

Listing 6.13: The functions and calls necessary to include a colour legend into the heatmap visualisation.

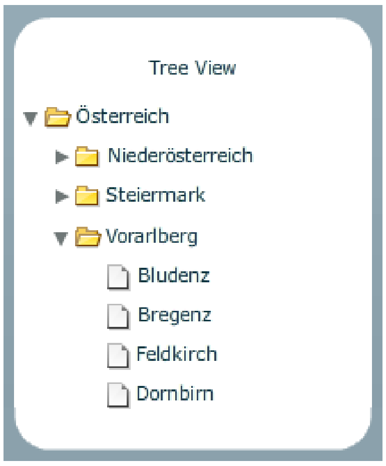


Figure 6.11: To support the understanding of hierarchical data, the treemap and the voronoi treemap offer a tree view component which can be shown on the right side of the visualisation.

6.3.3 Tree View

Sometimes, when visualising hierarchical data using a treemap or voronoi treemap, details of the hierarchical structures are not to be found at a glance. To support the recognition of hierarchical structure, an optional tree view component can be added to the right side of the visualisation. The tree view uses a list-like display, where each data entry is represented by a single node. Each node has its child elements drawn as sub-items which can be hidden or shown by collapsing or expanding the parent node. Figure 6.11 illustrates a tree view showing a simple hierarchical data set.

Since the tree view is linked to the data shown in the visualisation, interactions between the diagram and the tree view are possible. If a data entity in the diagram is clicked the branches leading to the corresponding tree view entry will be expanded and the entry is selected. On the other hand, selecting an entry in the tree view will trigger a highlight effect of the corresponding data in the visualisation.

The tree view is implemented using the Flex Tree class. As shown in Listing 6.14, the control element is added to the application (lines 1-3). To add the hierarchical data to the control element the `dataProvider` variable is assigned the root TreeNode element of the internal tree data structure (line 9). More details about the TreeNode class and its implementation can be found in Section 8.2.

6.4 Framework Functions

The Liquid Diagrams framework implements several functions common to the visualisation gadgets. This section provides a summary of the available functions and their intended use.

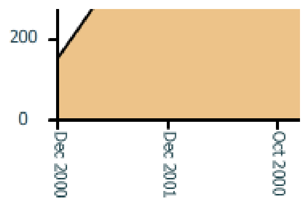
6.4.1 Individual Appearance

The ChartPanel class offers many functions to alter the visual aspects of the generated visualisation. The most important of these functions are:

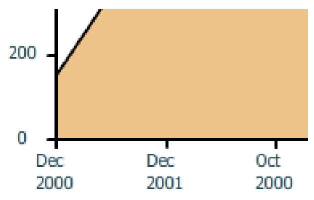
- **Titles:** Diagram title as well as axis titles, are optional and can be changed in appearance. Therefore the font, font size, position, and in specific cases even the direction of the titles can be altered.

```
<mx:VBox id="treeBox" width="200" height="100%" backgroundColor="#FFFFFF"
    paddingTop="20" paddingBottom="20" cornerRadius="20" borderStyle="solid"
    borderThickness="0">
  <mx:Tree id="tree" width="100%" height="100%" borderStyle="none" itemClick="
      treeItemClicked(event)"></mx:Tree>
</mx:VBox>

public function showTreeControl() : void {
  ...
  var treeParentNode:TreeNode = new TreeNode();
  treeParentNode.addChild(treeRoot, false);
  tree.dataProvider = treeParentNode;
  ...
}
```

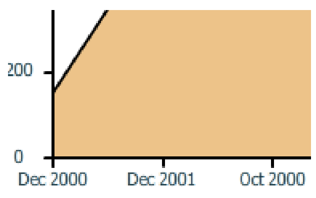
Listing 6.14: Adding and initialising the tree view component.



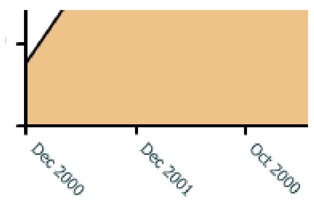
(a) Vertical label style.



(b) Line break label style.



(c) Centered label style.



(d) 45 degrees label style.

Figure 6.12: The four different x-label styles available in Liquid Diagrams.

- **Axis Labels:** When using a visualisation gadget that features x-axis and y-axis the label style used for the ticks, displayed along the axis, can be changed according to the four label styles shown in Figure 6.12.
- **Drag-and-Drop:** Some visualisations support drag-and-drop operations to rearrange axes. This can improve the recognition of patterns in the data.
- **Selection Labels:** When selecting data entities in the visualisation so called selection labels are shown along the known data points of the selected entities. Additionally the selected entities are drawn using higher line sizes and alpha values (both are configurable) to be able to compare them in a better way.
- **Line Sizes:** The line and border sizes of data entities, drawn in the visualisation, are settable.

6.4.2 Legend

The legend used in the Liquid Diagrams gadgets is not limited to a visual aspect only. Besides showing the entities colours the legend offers some functions to the user depending on the type of visualisation:

- **Selecting:** Clicking on the entities name in the legend will highlight (line size and alpha value) the entities visual representation in the visualisation. Depending on the type of visualisation, multiple selections are possible.
- **Hiding:** For better comparison results individual entities can be hidden without the need to manipulate the data set. Pressing the ALT key while clicking an entity name in the legend will hide this entity and thus remove it from the visualisation.
- **Bring data in front:** When a visualisation hosts many entities some entities may be over-drawn by other entities drawn at a later time. Each legend item of an entity can be clicked while holding down the SHIFT key to bring the data back in front.
- **Change colour:** When clicking on an entity's colour bin the colour which represents this entity in the visualisation can be set individually using the colour picker.
- **Change data points:** In some visualisations known data values are represented by data points in the visualisation. The style of the data point and its size can be set by clicking on the entity's data point displayed in the legend. This will open the data point picker to choose among the different data point styles.

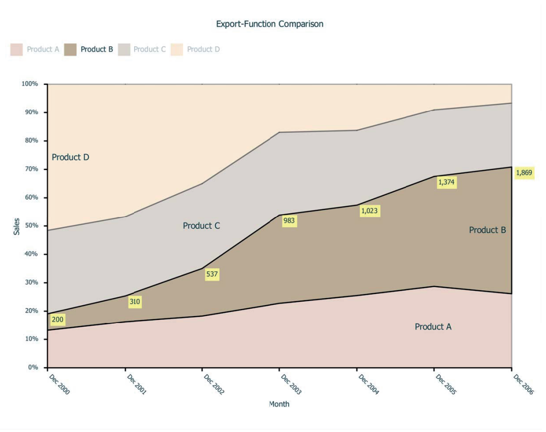
6.4.3 Diagram Export

In Lessacher [2009b] an export function based on the built-in print function was introduced. This function enables the export of a visualisation as a vector graphic (PDF) or as a bitmap. For this one of the many freely available PDF printers is required. Besides the need to have a PDF printer installed, there are several other drawbacks using this export function:

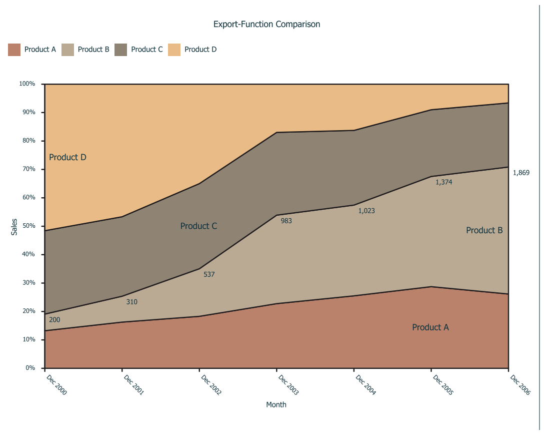
- The resulting graphic is too reliant on the version of the PDF printer. Bad PDF printers may lead to results not matching the original.
- Besides choosing the export format there are very few possibilities to influence the print function.
- The content internally needs to be resized to A4 paper format to fit on the printed page or some parts of the content will be cut off.
- Due to limitations in the used FlexPrintJob, class exported vector graphics do not contain alpha transparencies and colour effects.
- It is very complicated to process the result of the export function (PDF format) within the application.
- When choosing to export a bitmap instead of a vector graphic, the resolution of the bitmap is very low and there are no settings to influence the quality of the bitmap.

Although there are several issues when using this export function, it is still available using the `Print` button in the options panel. However, two new export functions have been added: `ExportasPNG` and `ExportasSVG`.

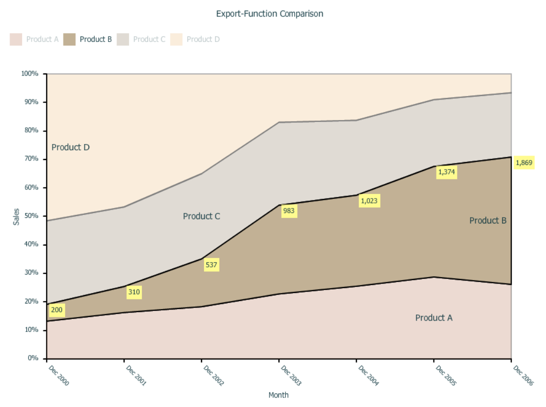
The first export function is to export the diagram using Flex's BitmapData and PNGEncoder classes [Adobe, 2010a] as a PNG image, as shown in Listing 6.15. The BitmapData object is initialised with the size of the ChartPanel which hosts the diagram (line 1). Afterwards the `draw` function of the BitmapData object is called to capture the content of the ChartPanel (line 2). Finally, the pixels of the BitmapData object are converted to a PNG-encoded ByteArray object (line 5).



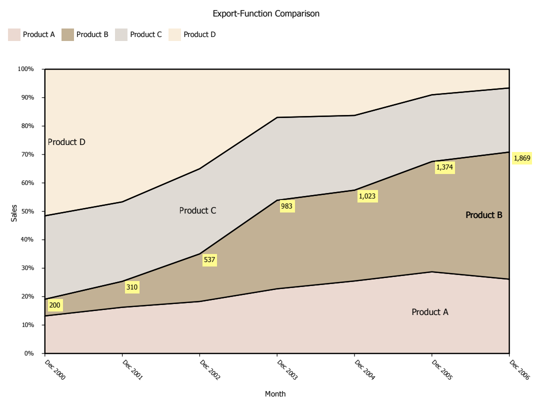
(a) Print as a bitmap. The quality is low because this is only a bitmap.



(b) Print as a vector graphic. Since the print function does not support transparencies, there are significant differences to the original.



(c) Export as PNG. The image is not freely scalable, because it is not a vector graphic, but it is an exact copy of the original with better resolution than the bitmap given by the built-in print function.



(d) Export as SVG. The only drawback of this export type is that the result might not exactly match the original, because the export is built up in parallel to the original.

Figure 6.13: The Liquid Diagrams framework offers four different ways to export a visualisation. Each has its benefits and drawbacks.

```
var bitmap:BitmapData = new BitmapData(chartPanel.width, chartPanel.height);
bitmap.draw(chartPanel);
var pngEncoder:PNGEncoder = new PNGEncoder();
imgArray:ByteArray = pngEncoder.encode(bitmap);
```

Listing 6.15: One new way to export the diagram is to export it as a PNG byte array. This code shows how the byte array is created.

```
 1  if(xRatio > yRatio) {
 2    var newHeight:Number = actualWidth * yRatio / xRatio;
 3    if(newHeight > actualHeight) {
 4      actualWidth = actualHeight * xRatio / yRatio;
 5    } else {
 6      actualHeight = newHeight;
 7    }
 8  } else {
 9    var newWidth:Number = actualHeight * xRatio / yRatio;
10    if(newWidth > actualWidth) {
11      actualHeight = actualWidth * yRatio / xRatio;
12    } else {
13      actualWidth = newWidth;
14    }
15  }
```

Listing 6.16: Computing the available space of the chart panel in regard to the display ratio.

This export function is quite simple but it generates a high quality PNG export of the diagram panel as can be seen in Figure 6.13. Since the export is a pixel-wise capture of the ChartPanel, the export exactly matches the reference visualisation and there is no need to preview the export as image. Another benefit is that the ByteArray object can be further processed by the application. However, the major drawback is that a PNG image is a raster image and is not scalable.

The second new export function creates a scalable vector graphic (SVG). The SVG export file is built up using SVG tags like `line`, `text`, `rect`, and `circle`. Each function that creates elements to build up the visualisation on the ChartPanel also creates output using SVG tags. Since the elements on the screen and in the export file are created using different functions of different programming languages the results might not exactly match. This is especially true when using special fonts, because the SVG file only proposes a font family for each text node. When the `Export` button in the options panel is clicked, all the SVG tags created along with the screen output are written into a file.

6.4.4 Display Ratio

All visualisations made with the Liquid Diagrams framework attempt to use all the space available to the gadget. Thus, using high resolution and large displays can lead to untypical and undesirable display ratios. One way to obtain a specific display ratio would be to manually resize the window hosting the gadget, but this is neither comfortable nor are the results precise. Hence, the display ratio of each visualisation is settable using the Options Panel.

To set a display ratio, the check box `ExplicitDisplayRatio` needs to be set. Afterwards the display width and height text boxes located beneath the check box are activated. After changing the display width or height, the change needs to be confirmed by pressing the return key. This will force a redraw of the diagram using the entered display ratio. Un-checking `ExplicitDisplayRatio` will discard the custom ratio and use the entire available space again.

Listing 6.16 illustrates how the available space for the `ChartPanel` is computed. First of all, it is determined if the width or the height ratio is higher (line 1). The next step is to try to keep the display size of the higher aspect ratio and to resize the smaller ratios display size to fulfil the aspect ratio (lines 6 and 13). If the resulting size exceeds the available space the size of the smaller ratio is taken and the display size of the higher ratio is adapted (lines 4 and 11).

YYYY-MM-DD	DD-MM-YYYY	MM-DD-YYYY	DDxx MMMM YYYY
YYYY/MM/DD	DD/MM/YYYY	MM/DD/YYYY	DD MMM YYYY
YYYY.MM.DD	DD.MM.YYYY	MM.DD.YYYY	MMM YYYY
YYYY-MM	YYYY/MM	YYYY.MM	YYYY
MM-YYYY	MM/YYYY	MM.YYYY	

DD... Day (e.g. 01)
M... Month (e.g. MM = 02, MMM = Feb, MMMM = February)
YYYY... Year (e.g. 1983)
xx... Ordinal Suffixes (e.g. st, nd, rd, th)

Table 6.1: Date formats available in the Liquid Diagrams framework. Values have to match one of these formats to be recognized as representing a date.

6.4.5 Date Formatting

The Liquid Diagrams framework introduced in Lessacher [2009b] supports formatting of all numbers shown in the diagram. To do so, various number format schemes that use different decimal and thousands separators can be chosen using the `NumberDisplayFormat` combo box in the options panel.

In addition to the number formatting a new function, date formatting has been added. Date formatting enables the user to choose among 19 different date formats (shown in Table 6.1). By choosing one of the formats, that format is applied to all date objects in the diagram.

To recognise that a given value is a valid date, it has to be in one of 19 date formats shown in Table 6.1 (the exception is YYYY). This is implemented using the regular expressions shown in Listing 6.17. If the value conforms to one of the date formats, the value is written into a Flex Date object and the `setDateFormat` function of the ChartPanel class is called. This function receives the date object and transforms it to the desired format using the Flex DateFormatter class. Therefore the `formatString` of the DateFormatter object is set to the desired date pattern (like shown in Table 6.1) and the `format` function is called, which returns the desired formatted date as a String.

6.4.6 Alphanumerical Data

In Lessacher [2009b] the parallel coordinates visualisation only supported numerical data. Since then the star plot visualisation has been added, which uses the same data format and is built up in nearly the same way. With this addition, the possibility to use alphanumerical (categorical) data was introduced to both, the star plot visualisation and the parallel coordinates visualisation.

To implement alphanumerical classes an array `numericColumn` was added to indicate if a column only contains numerical values. This is done in the `getDataObjects` method right before the internal data computation is carried out. This internal data computation verifies and validates each data value. If the `numericColumn` variable specifies a data column as not numeric, the data computation for this column is not carried out because then all kind of values are allowed.

After finishing the data computation, the `createAlphanumericalClasses` function, which creates different classes based on the data of a column is called. This function is illustrated in Listing 6.18. Each column which hosts non-numerical data is assigned an `alphanumericalClasses` array which holds the different data classes (line 4). All values present in the column are compared to each other and for each different value an entry is added to the `alphanumericalClasses` array (lines 5-9). Finally, the classes of

```
/^(?P<day>0[1-9]|[12][0-9]|3[01]|[1-9])(.)? (?P<month>Jan|Feb|Mar|Apr|May|Jun|Jul|
    Aug|Sep|Oct|Nov|Dec|Mai|Okt|Dez) (?P<year>\d{4})$/

/^(?P<day>0[1-9]|[12][0-9]|3[01]|[1-9])(.)? (?P<month>Jan|Feb|Mar|Apr|May|Jun|Jul|
    Aug|Sep|Oct|Nov|Dec|Mai|Okt|Dez) (?P<year>\d{4})$/

/^(?P<month>0[1-9]|1[012]|[1-9])(.|-|\/)(?P<day>0[1-9]|[12][0-9]|3[01]|[1-9])
    (.|-|\/)(?P<year>\d{4})$/

/^(?P<day>0[1-9]|[12][0-9]|3[01]|[1-9])(.|-|\/)(?P<month>0[1-9]|1[012]|[1-9])
    (.|-|\/)(?P<year>\d{4})$/

/^(?P<year>\d{4})(.|-|\/)(?P<month>0[1-9]|1[012]|[1-9])(.|-|\/)(?P<day
    >0[1-9]|[12][0-9]|3[01]|[1-9])$/

/^(?P<month>Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec|Mai|Okt|Dez)? (?P<year
    >\d{4})$/

/^(?P<year>\d{4})(.|-|\/)(?P<month>0[1-9]|1[012]|[1-9])$/

/^(?P<month>0[1-9]|1[012]|[1-9])(.|-|\/)(?P<year>\d{4})$/

/^(?P<day>0[1-9]|[12][0-9]|3[01]|[1-9])(th|st|nd|rd) (?P<month>January|February|
    March|April|May|June|July|August|September|October|November|December|Januar|
    Februar|März|April|Mai|Juni|Juli|August|September|Oktober|November|Dezember)
    (?P<year>\d{4})$/
```

Listing 6.17: The regular expressions used to check if a value is a date. If none of these regular expressions match, the value is not considered to be a date.

each column are sorted using the `sort` function of the Flex Array class (line 10). After the sort operation the data in the array is sorted in ascending order, and from now on the index of each array-element (a numerical value) represents its value.

```
public function createAlphanumericalClasses() : void {
  for(var i:int = 0; i < numericColumn.length; i++) {
    if(numericColumn[i] == false) {
      alphanumericalClasses[i] = new Array();
        for(var j:int = 1; j < data.length; j++) {
          if(notInArray(data[j][i], alphanumericalClasses[i])) {
            alphanumericalClasses[i][alphanumericalClasses[i].length]=data[j][i];
          }
        }
        alphanumericalClasses[i] = alphanumericalClasses[i].sort();
      }
  }
}

public function getClassIndex(value:String, index:Number) : Number {
  for(var i:int = 0; i < alphanumericalClasses[index].length; i++) {
    if(alphanumericalClasses[index][i] == value) {
      return i;
    }
  }
  return -1;
}
```

Listing 6.18: The creation and handling of alphanumerical data classes.

Chapter 7

Liquid Diagrams Visualisations

The Liquid Diagrams framework introduced in Lessacher [2009b], initially offered the functions to create five visualisation types (line charts, pie charts, bar charts, area charts, and parallel coordinates). All of visualisations underwent upgrades to make use of the new created components and functions, fix minor issues, and to enhance their usability.

The Liquid Diagrams framework now contains nine visualisation gadgets. Each visualisation gadget's implementation is located in a separate MXML file specified by the visualisation name. As shown in Figure 6.1, these MXML files are compiled to SWF objects, which can be included on web pages or executed using a standalone Flash player.

7.1 Line Chart

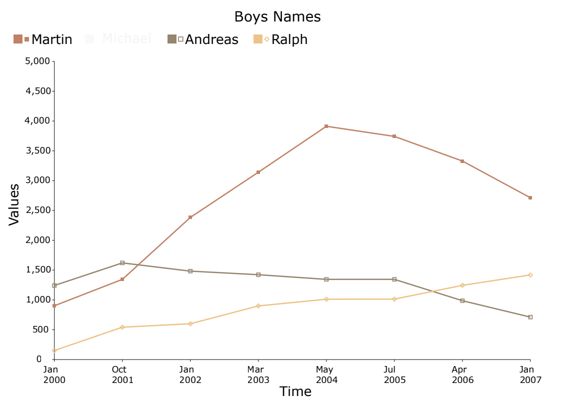
As described in Section 2.4.1, line charts are most commonly used for time-based data because the lines reveal trends over time Sample line charts can be seen in Figure 7.1. The line chart visualisation uses the axis creation functions of the ChartPanel to set up an x-axis and a y-axis. The data entities are then drawn by looping over all data points of the entities and adding them to the visualisation in relation to the values shown on the y-axis and x-axis. Afterwards all data points belonging to one entity are connected by lines, resulting in an entity characteristic line.

7.2 Pie Chart

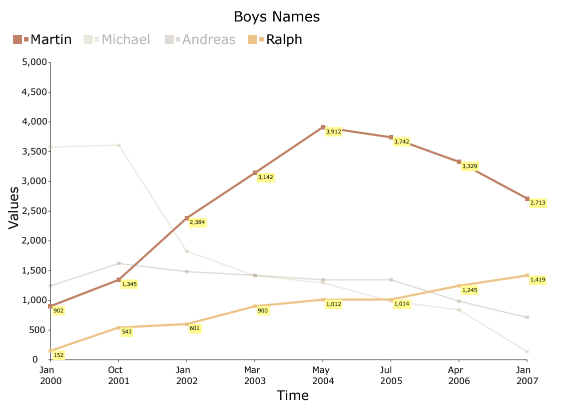
A pie chart consists of a circle composed of several sectors, each representing a different entity (see Section 2.4.2). The area of each sector represents its relative weight (significance). Examples for pie charts are illustrated in Figure 7.2.

The pie chart does not include axes of any kind. Therefore, it does not have to call axis creation functions or functions to measure label spaces. The available space for the visualisation is solely determined by the space of the legend and the borders around the visualisation.

The first step when drawing the pie is to determine which sectors are visible. Each visible slice of the pie is drawn separately using lines and Bezier curves. The algorithm to determine slice sizes is described in Section 7.6 and illustrated in Figure 7.6.

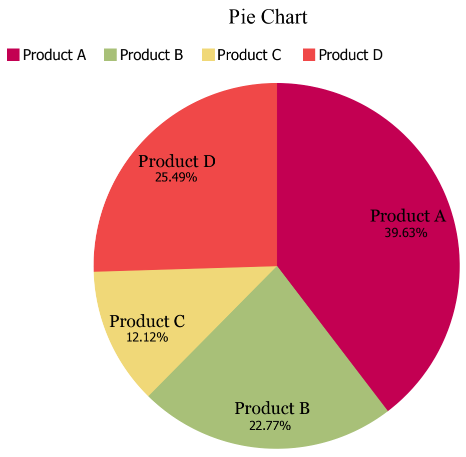


(a) A line chart using different types of data points and one hidden entity.

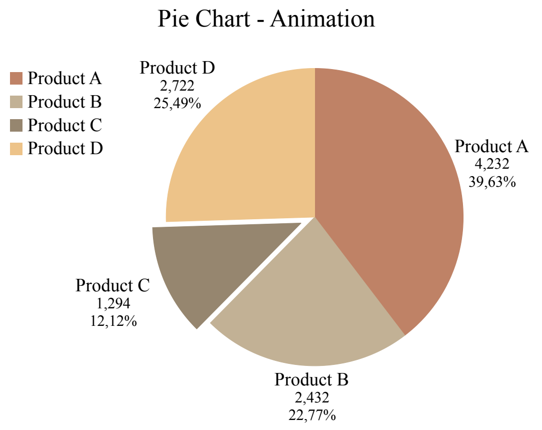


(b) This line chart shows two selected entities. Using the selection function entities can be easier compared to each other.

Figure 7.1: Liquid Diagrams line charts, showing the number of occurrences of four different names over an 8-year period (2000-2007).

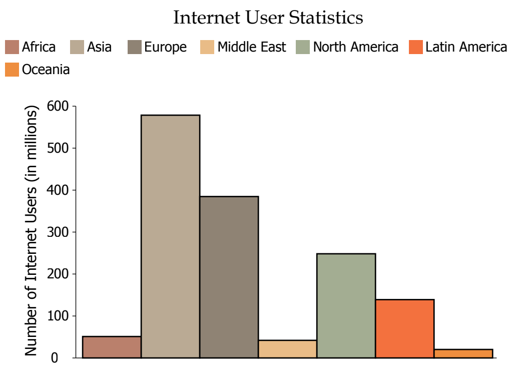


(a) This pie chart uses the option to show labels for the name and the percentage of each slice.

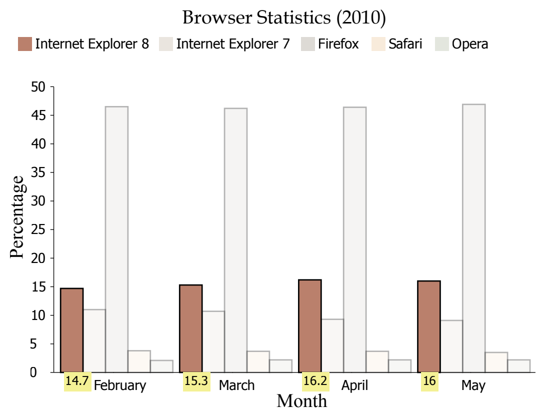


(b) This image shows one slice in an animation state. The slice is slid out after a user click. The radius of the labels was modified to also allow display of the labels outside the pie chart.

Figure 7.2: Various pie chart visualisations created with Liquid Diagrams.



(a) This bar chart shows the number of internet users by the region. The y-axis is bound to years (time) but only one year is given per entity.



(b) A bar chart displaying the usage percentage of different browsers during four different months in 2010. In this visualisation, Internet Explorer 8 is selected and thus highlighted.

Figure 7.3: Various Liquid Diagrams bar charts.

In addition to the general features of the Liquid Diagrams visualisations (see Section 6.4), the pie chart visualisation offers several individual features:

- **Pie Size**
  Per default the radius of the pie chart fits into the available space. This is symbolized by a radius value of 100 percent. However if a smaller pie size is desired it can be achieved by entering a scaling for the radius between 1 and 100 percent. Any changes made are only accepted after pressing the enter key.

- **Labelling**
  There are three different types of labels that can be shown or hidden using the Options Panel. These labels are the name of the entity, the value of the entity, and the entity's share of the total in percent.

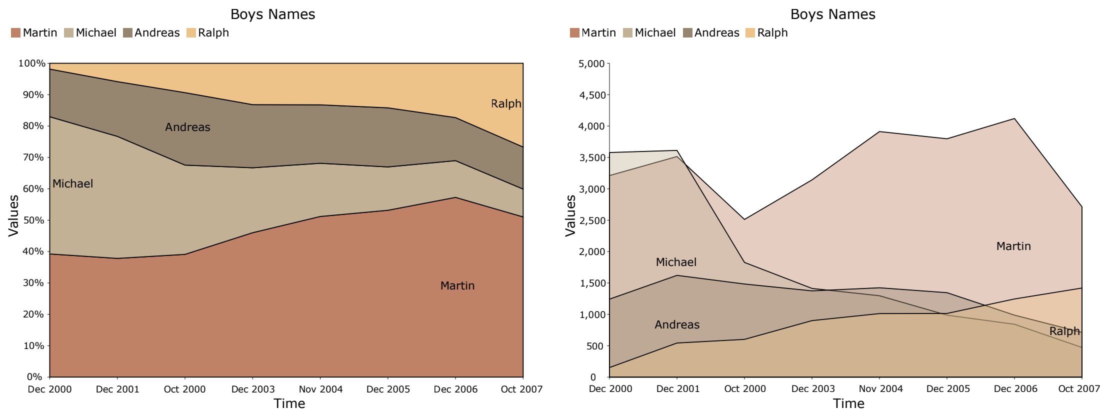
  The placement of labels can also be adjusted using the Options Panel. The text box `LabelRadius` is used to set the radius where the labels should be displayed. If a value greater than 100 is used, the labels are drawn outside of the pie.

- **Click Animation**
  The visualisation also implements a slice-slide animation, when an entity or its legend label is clicked. The first time it is clicked the slice is moved away from the centre and the second click brings the slice back to its old position. This effect is shown in Figure 7.2. The implementation of this effect is based on the Move class included in Flex's `mx.effects` package.

7.3 Bar Chart

As described in Section 2.4.3, bar charts are especially useful for comparing two or more values. An example of a Liquid Diagrams bar chart is illustrated in Figure 7.3.

The Liquid Diagrams bar chart implements common features introduced in Section 6.4 and also provides a setting for the `smallestbarwidth`. This width (in pixels) determines the smallest possible width of a single bar, representing an entity, in the visualisation. The smaller the width the more x-ticks



(a) A stacked area chart using the percentage option. **(b)** An overlay area chart. Each entity occupies an area according to its values. The transparency of these entities is increased enabling other entities in the background to be seen making it easy to compare entities.

Figure 7.4: Variants of area charts created with the Liquid Diagrams Framework.

can be shown in the visualisation. If the width is too large it can happen that not all data are shown in the visualisation and a `cutOffEvent` will be fired (see Section 6.2.3).

7.4 Area Chart

Area charts are especially useful to reveal trends in the data, because their resulting areas can easily be compared (see Secton 2.4.4). The construction of the area chart depends on the type of area chart to be drawn. In an overlay chart, the highest value is determined by finding the highest value of an entity. In a stacked area chart the highest value of the chart is the highest sum of entity values at a given tick. After the extreme points have been determined and the labels measured, the x-axis and y-axis are created.

Areas are then created for each entity. The overlay area chart is drawn like a line chart, because each entity is drawn independently of the others. The entities of the stacked area chart need to be created differently, because the entities are drawn atop of each other. To determine the y-coordinate of an entity at a specific tick, all previous entity values at this tick have to be summed up and added to the entity's value.

Having drawn all the points of an entity, the area of the entity is closed by drawing a line from the last point back to the starting point. In an overlay chart the closing line runs along the x-axis. The closing line of a stacked area chart runs along the entity line of the entity directly beneath. Once closed, the entity is filled with the desired colour.

The final step in the construction of an area chart is to add a label to each area. To accomplish this, the location of each entity's largest vertical extent is determined and the area name label is inserted at this position.

The Liquid Diagrams area chart offers some specific features:

- **Different Types of Area Charts**

```
name;mpg;cyl;disp;hp;lbs;accel;year;origin
chevrolet chevelle malibu;18;8;307;130;3504;12;1970;1
buick skylark 320;15;8;350;165;3693;11.5;1970;1
plymouth satellite;18;8;318;150;3436;11;1970;1
amc rebel sst;16;8;304;150;3433;12;1970;1
ford torino;17;8;302;140;3449;10.5;1970;1
ford galaxie 500;15;8;429;198;4341;10;1970;1
chevrolet impala;14;8;454;220;4354;9;1970;1
plymouth fury iii;14;8;440;215;4312;8.5;1970;1
pontiac catalina;14;8;455;225;4425;10;1970;1
amc ambassador dpl;15;8;390;190;3850;8.5;1970;1
...
```

Listing 7.1: The first 10 entries of a high-dimensional data set containing facts about 400 different cars. The data set was created by Ramos and Donoho [2010].

There are two different types of area chart available as shown in Figure 7.4: stacked and overlay. The type of visualisation can be dynamically switched using the Options Panel.

- **Display in Percent**
  The areas are displayed according to their percentage share of the summed up values. This leads to better use of the available space, but discards the original units.
- **Area Labels**
  This option determines whether area labels are shown or not.

7.5 Parallel Coordinates

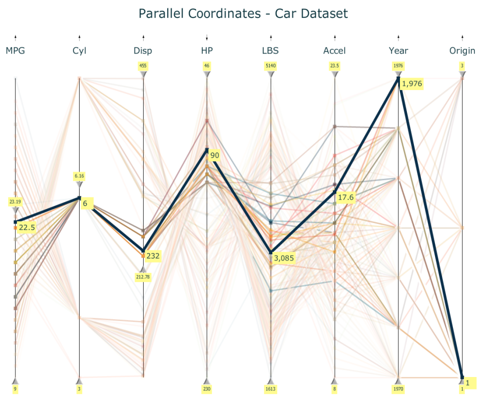
Parallel coordinates is a popular and effective way to visualise high-dimensional data (see 2.4.5). An example of a high-dimensional data set is illustrated in Listing 7.1. The visualisation is constructed by dividing up the available space into N separate axes, where N is the number of data dimensions. The highest and the lowest value of each dimension's entities are taken as the corresponding upper and lower axis limits. The next step is to take an entity, and draw its values relative to the upper and lower boundary on the corresponding axis. Afterwards, the data points are connected by lines, leading to an entity characteristic line or polyline. This line drawing process is then carried out for each entity. Examples of parallel coordinates visualisations can be seen in Figure 7.5.

To create the parallel coordinates visualisation the y-axes, representing the categories, are spread evenly among the available horizontal space. Afterwards the data points for each entity are mapped and drawn onto the y-axes. Then all the points of an entity are connected to each other with a line to create an entity characteristic line.

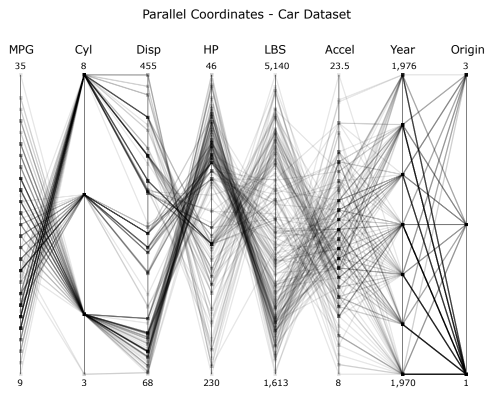
The Liquid Diagrams parallel coordinates visualisation has numerous visualisation-specific features:

- **Filter Entities**
  Entities can be filtered out by dragging the top and bottom slider thumb of each axis. When an entity drops outside the two slider thumbs of an axis, the entity is filtered out and is visually greyed out.

- **Changeable Axis Positions**
  The position of the axes can be rearranged by dragging and dropping the axis labels to a different



(a) A PNG export of a Liquid Diagrams parallel coordinates visualisation.



(b) An SVG export of a Liquid Diagrams parallel coordinates visualisation.

Figure 7.5: The Liquid Diagrams parallel coordinates visualisation showing the car dataset which contains facts about 400 different cars.

position in the visualisation. This helps to better identify patterns in the data, since dimensions can be best compared next to one another.

- **Inversion of Axes**
  Usually, the lowest value is at the bottom and the highest value is at top of an axis. This can be changed by clicking on the axis invert button located above each axis. A further click resets the axis.

- **Entity Labels**
  The label of each entity can be displayed at the corresponding line's starting point, ending point, or at both positions.

7.6 Star Plot

The first of four new visualisations that were added to the Liquid Diagrams framework is the star plot visualisation (see Section 2.4.7). It is called a star plot because when using several axes, the resulting diagrams shape looks like a star. The star plot visualisation is used to visualise high-dimensional data like the data shown in Listing 7.2. Just like the parallel coordinates visualisation, the start plot visualisation uses different axes to map the entities' values for a specific attribute. The difference between the two visualisations is the alignment of the axes. The parallel coordinates visualisation draws the axes parallel to each other along the available width of the display space. The star plot visualisation aligns its axes in the shape of a circle, with each axis origin in the centre of the display space.

Besides the general features of the Liquid Diagrams visualisations (see Section 6.3 and Section 6.4), the star plot visualisation offers several specific features:

- **Entity Colours**
  The colour of each entity drawn in the visualisation can be set a user-friendly way. To change the colour of an entity, the user just has to click the corresponding colour spot in the legend and choose a new colour from the colour chooser popup window.

```
Name;Manufacturer;Type;Calories;Protein;Fat;Sodium;Fibre;Carbo;Sugar;Shelf;
    Potassium;Vitamins;Weight;Cups
100%_Bran;N;cold;70;4;1;130;10 ,00;5 ,00;6 ,00;3;280;25;1 ,00;0 ,33
100%_Natural_Bran;Q;cold;120;3;5;15;2 ,00;8 ,00;8 ,00;3;135;0;1 ,00;
All—Bran;K;cold;70;4;1;260;9 ,00;7 ,00;5 ,00;3;320;25;1 ,00;0 ,33
All—Bran_with_Extra_Fiber;K;cold;50;4;0;140;14 ,00;8 ,00;0 ,00;3;330;25;1 ,00;0 ,50
Almond_Delight;R;cold;110;2;2;200;1 ,00;14 ,00;8 ,00;3;;25;1 ,00;0 ,75
Apple_Cinnamon_Cheerios;G;cold;110;2;2;180;1 ,50;10 ,50;10 ,00;1;70;25;1 ,00;0 ,75
Apple_Jacks;K;cold;110;2;0;125;1 ,00;11 ,00;14 ,00;2;30;25;1 ,00;1 ,00
Basic_4;G;cold;130;3;2;210;2 ,00;18 ,00;8 ,00;3;100;25;1 ,33;0 ,75
Bran_Chex;R;cold;90;2;1;200;4 ,00;15 ,00;6 ,00;1;125;25;1 ,00;0 ,67
Bran_Flakes;P;cold;90;3;0;210;5 ,00;13 ,00;5 ,00;3;190;25;1 ,00;0 ,67
...
```

Listing 7.2: The first 10 entries of a high-dimensional data set containing facts about 78 different cereals acquired from StatLib [2010].

Besides colouring each entity manually, which can be a tough task, depending on the number of entities, there is also the option to use pre-defined colour schemes. Choosing a colour scheme automatically assigns each drawn entity a pre-defined colour from the colour scheme. When using a monochrome colour scheme (colour schemes that only host one colour), the legend is removed and the alpha value of each entity is reduced to a user-settable value. By reducing the alpha value, each entity's line is displayed semi-transparently, enabling entities drawn underneath them to be seen. If several entities lines share the same line segment, this segment's alpha value would be computed by a multiplication of each entity's alpha value giving a less transparent line than a segment only occupied by a single entity. An example of a star plot visualisation using a monochrome colour scheme is shown in Figure 7.7a.

- **Rearranging Axes**
  When created for the first time the different axes are mapped to attributes depending on the order of the attributes in the data set. To enable better ways to compare the relationships between different attributes and thus gain more insight into the data set, the order of the axes can be rearranged interactively. To rearrange an axis, the label representing this axis can be dragged to a different place in the visualisation. The axis will then be moved between the two axes closest to the place the axis was dragged to.

- **Area Filling**
  To obtain a different view on the data, each shape created by the entity characteristic lines can be filled with the entity's colour. The alpha value of the filling colour can be set to achieve semi-transparent areas. The overlapping of entities' areas can be compared more easily. Figure 7.7b shows a star plot diagram using area filling.

- **Invert Axes**
  Initially, all axes are drawn mapping the attribute's lowest value to the centre and highest value to the outside of the circle. Each axis can be swapped to contain the highest value in the centre and the lowest on the outside. This can be done for each axis individually by clicking on the inverse arrow next to the axis label.

- **Axis Labels**
  Besides turning on or off the axis labels, there are several configuration possibilities regarding what the labels contain (ranges, inverse buttons) and how they are positioned along the axes.

- **Interactivity**
  The star plot visualisation supports high interactivity to enrich the exploration of the data. When

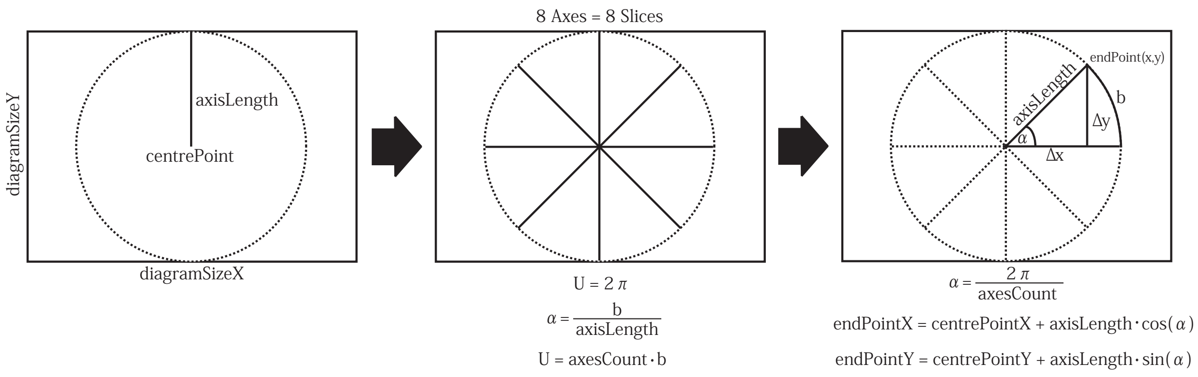


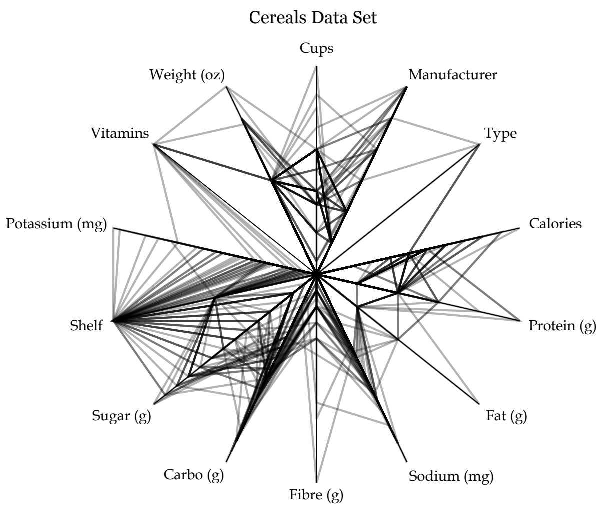
Figure 7.6: Computing the end point of each axis in a star plot.

moving the mouse over an entity, a formatted tooltip shows all information known about this entity and its data. Several click events are implemented for each entity and its corresponding label. The effect of the click event depends partially on the keyboard key held during the click event. Clicking the entity will select it and thus increase the entity's line size and reduce the alpha values of all non-selected entities. Additionally, labels will be shown where the entity's lines cross the axes, showing the data values at this point. When clicking entities while holding down the CTRL key, all clicked entities will be selected. For better comparison results, individual entities can be hidden without the need to manipulate the data set. Pressing the ALT key while clicking an entity will hide this entity and thus remove it from the visualisation. The legend item of the entity will still be shown using a grey, alpha reduced colour. Clicking this item once more while holding down the ALT key will bring back the entity. When a visualisation contains many entities, some entities may be obscured by other entities drawn at a later time. Any entity or the legend item of an entity can be clicked while holding down the SHIFT key to bring this entity to the front.

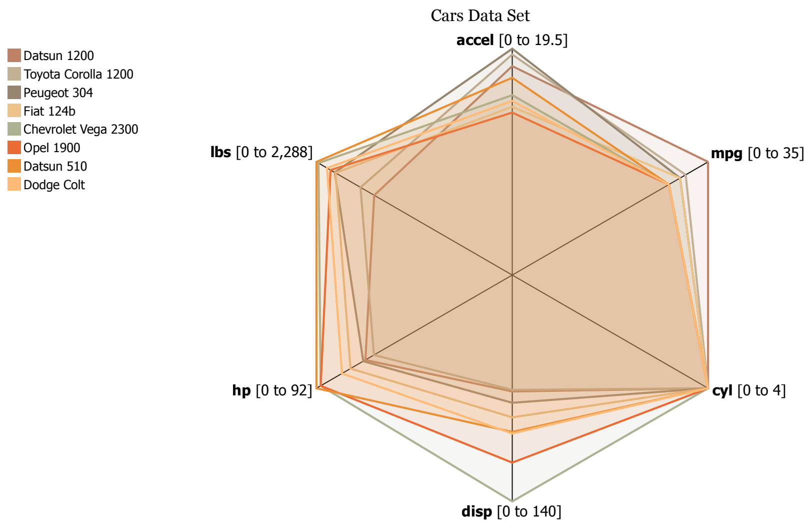
- **Filtering**
  In order to consider only data entities matching to certain properties, the Liquid Diagrams framework offers a filtering function for each axis. Two slider thumbs are initially drawn at the maximum and the minimum value of the axis. Dragging the sliders along the axis filters the entities and thus fades out entities that are not contained within the slider positions. Filtered out entities are not entirely hidden: they are still visible using a very low alpha value (0.1). Filtered out entities do not trigger interactions like mouse-over and selection effects.

The first task when implementing a star plot visualisation is to determine how the axes should be displayed and how their length and position are computed. Listing 7.3 illustrates how this is handled in the Liquid Diagrams framework. The length of the axis is determined by the smaller length of either the width or the height (line 1). Afterwards the space required for the axes labels is subtracted from the length, resulting in the final axes length (line 2). The start point of each axis is given by the origin of the diagram (line 4). The origin is computed in a previous step by dividing the available width and height by 2. The end points of the axes are computed by dividing the circle around the centre into N equally sized wedges, where N is the number of axes (dimensions in the data). Each point where a wedge intersects a circle of radius of axis length becomes the end point of an axis (lines 6-13). This is also illustrated in Figure 7.6

To complete the visualisation the values of data entities need to be mapped to their position on each axis. Listing 7.4 illustrates how the position of a given value on an axis is computed using Flex's `Point` class.



(a) A star plot visualisation showing the cereal data set using a monochrome colour scheme. Line segments occupied by more than one entity are drawn more opaquely than line segments occupied by a single entity.



(b) A star plot visualisation using area filling to support the recognition of shapes and overlappings.

Figure 7.7: The star plot visualisation is a highly interactive visualisation suitable for multidimensional data sets.

```
axisLength = (diagramSizeY < diagramSizeX) ? (diagramSizeY / 2) : (diagramSizeX /
    2);
axisLength = ((axisLength * radius) / 100) - ((diagramSizeY < diagramSizeX) ? (
    labelHeight) : (labelWidth));

var centrePoint:Point = new Point(diagramOriginX, diagramOriginY);

var alpha:Number = (Math.PI * 2) / axesCount;
for(i = 0; i < axesCount; i++) {
  var angle = alpha * (i + 1);

  endPoint[i] = new Point();
  endPoint[i].x = centrePoint.x + axisLength * Math.cos(angle);
  endPoint[i].y = centrePoint.y - axisLength * Math.sin(angle);
}
```

Listing 7.3: Computing the length and position of the axes displayed in a star plot visualisation are computed.

```
var axisRange:Number = maximum - minimum;
var percentage:Number = (value - minimum) / range;
var location:Point = Point.interpolate(axisEnd, axisStart, percentage);
}
```

Listing 7.4: Calculating the position of a point along a star plot axis given a percentage.

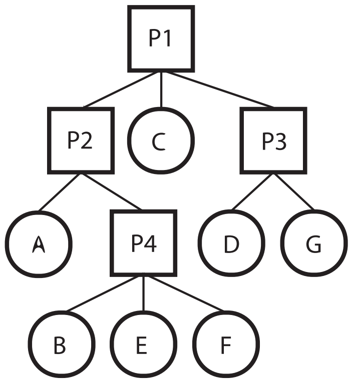
7.7 Tree Map

First introduced in 1991 a treemap [Johnson and Shneiderman, 1991] is a space-filling visualisation for large hierarchical data sets (see Section 2.4.8. Since then treemaps have become very popular and are used for many tasks, including the display of disk usage, financial analyses, and sports data.

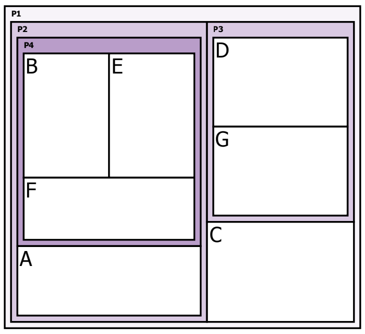
A treemap models each item in a hierarchy as a rectangle. Hierarchical structures are presented by nesting the rectangles, resulting in all the rectangles at one hierarchy level sharing the space of their common parent rectangle. Figure 7.8a shows the hierarchical structure of a set of data nodes using a graph representation. The mapping of the nodes to a treemap is shown in Figure 7.8b. If the data in the data set is weighted, the weight of each node can be reflected in the treemap by sizing rectangles in proportion to it. Nodes with higher weights occupy larger areas than those with smaller weights. After assigning a weight to each of the nodes in Figure 7.8a, the newly generated treemap would look like Figure 7.8c. In addition to sizing the data according to their weight, a treemap can include an additional data dimension using colour coding.

While Shneiderman [1992] introduced the original slice-and-dice algorithm to lay out the rectangles, several new algorithms have been introduced since then. Each algorithm gives different drawing results regarding sizing and distribution of the entities' rectangles. According to Shneiderman [2008] the optimal layout algorithm uses balanced square nodes with an aspect ratio close to 1 while preserving the order of the input data and reflecting changes to the data set. Unfortunately these factors contradict one another. Shneiderman [2008] hosts a list of known layout algorithms and rates their performance according to their aspect ratio, ordering, and stability:

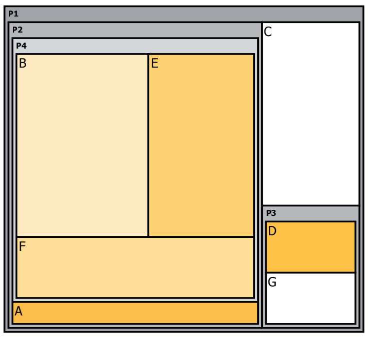
- **Slice-and-Dice**
  Slice-and-dice was the first algorithm used in treemap visualisations as presented in the original



(a) A graph showing the hierarchical structure of the data set. Each leaf node possesses the same priority (weight).



(b) The data shown in (a) shown as a treemap. If the parents' borders were turned off each element would occupy the same space because all have equal weights.



(c) By adding weights to the nodes, the space occupied by each node is calculated proportionately. An additional attribute of the data is shown using colour.

Figure 7.8: Mapping hierarchically structured data into a treemap visualisation.

treemap paper [Johnson and Shneiderman, 1991]. It uses parallel lines to alternately horizontally and vertically divide a rectangle into smaller rectangles, representing the rectangle's children. The algorithm is easy to implement and retains ordering, but produces very bad (high) aspect ratios.

- **Ordered**
  According to Shneiderman and Wattenberg [2001] which introduced it, this algorithm uses layout algorithms that change relatively smoothly under dynamic updates. The pivot-by-middle and pivot-by-size algorithms roughly preserve the ordering of the index of the items, which will fall in a left-to-right and top-to-bottom direction in the layout. The aspect ratio is low but not optimal.

- **Squarified**
  This algorithm, introduced in Bruls et al. [1999] results in the best aspect ratio results, but at the cost of a loss of the ordering present in the data set. Another drawback is that changing the data may result in dramatic changes in the layout.

- **Cluster**
  This algorithm's layout results are very similar to the results of the squarified algorithm and thus shares its benefits (ratio) and drawbacks (order, update). The cluster algorithm was introduced by Martin Wattenberg in Wattenberg [1999].

- **Strip**
  The strip treemap algorithm introduced in Bederson et al. [2002] is a modification of the squarified layout algorithm. It works by processing input rectangles in order, and laying them out in horizontal or vertical strips of varying thicknesses. The results of the algorithm are comparable to the results of the ordered treemap algorithms, but with better readability.

The Liquid Diagrams treemap visualisation provides many functions to explore a hierarchical data set and modify the resulting visualisation. The result can be exported using the export functions introduced in Section 6.4.3. The visualisation also uses the colour legend described in Section 6.3.2 and thus enables switching between different colour distributions. If enabled, a tree view component is shown on the right side of the visualisation. This component enables quick access and navigation to specific nodes (see Section 6.3.3). Besides these features, the Liquid Diagrams treemap visualisation offers:

- **Layout Algorithm**
  The treemap offers two different layout algorithms to choose from. The first is the slice-and-dice algorithm, which preserves the order of the nodes according to the input data. The drawback is that it typically has bad aspect ratios and thus generates many thin strips. The second algorithm, the squarified treemap algorithm, has the best aspect ratios among all layout algorithms, but does not preserve the ordering present in the data set. Figure 7.9 illustrates the result of both available layout algorithms.

- **Nesting**
  Nesting was introduced in Johnson and Shneiderman [1991] to enhance the recognition for hierarchical structures. Each non-leaf node is given a border to show which child nodes belong to this node. In the Liquid Diagrams treemap visualisation, nesting is turned on by default, but can be switched off to preserve the children's original rectangle sizes.

- **Zoom**
  Due to the hierarchical structure of the data shown in the treemap, the data is stored in a tree structure. Within this tree structure, each parent (non-leaf node) can be chosen to be displayed as the top element. This can be done by either clicking on the parent node's border in the visualisation or by using the tree view control element located to the right of the visualisation. Depending on the hierarchical position of the clicked node with respect to the currently displayed top node, either a zoom in or zoom out will be carried out.

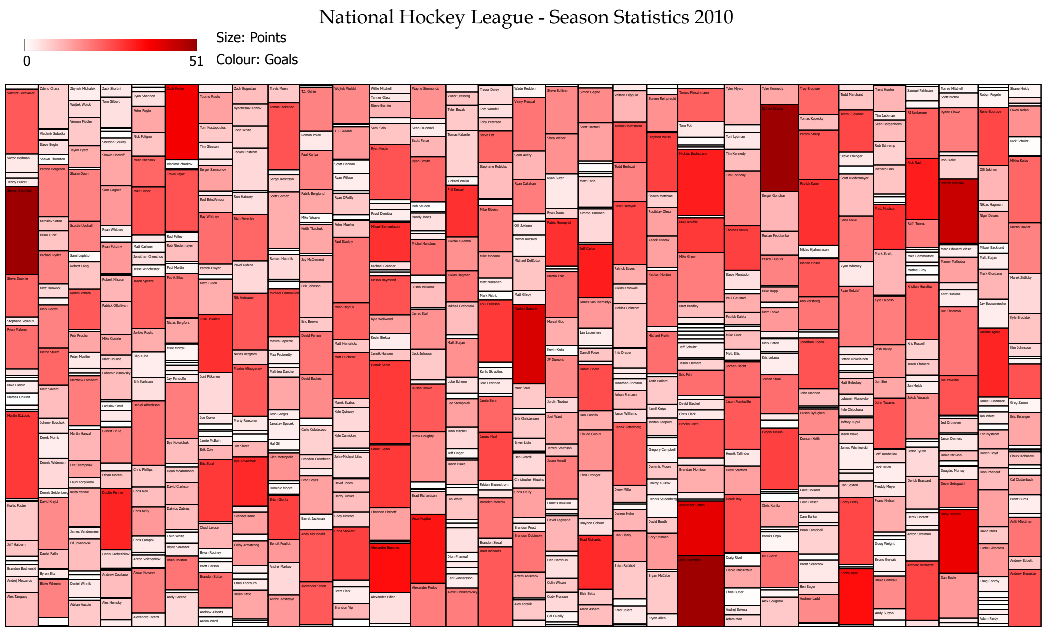
- **Choosing Size and Colour Attributes**
  The attributes to determine the rectangles' size and colour can be changed interactively using the combo boxes in the legend or by accessing the combo boxes in the Options Panel.

- **Border Sizes**
  Border sizes can be changed interactively to enhance the visibility of the hierarchical structure in the data set.

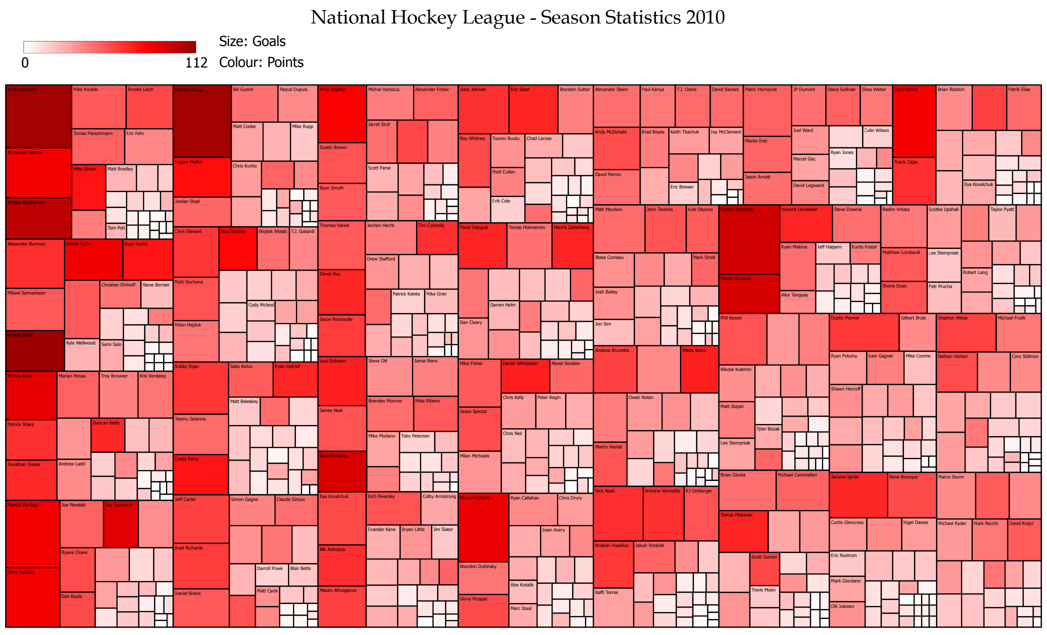
The first task when implementing treemap visualisation was to decide how to handle the data inside the gadget. Since the data is hierarchical, a custom tree class is used to build up a tree structure directly after receiving the data. More about the `TreeNode` class and how the data is processed can be found in Section 8.2. After the data is stored in the `TreeNode` class, the layout algorithm computes how to lay out the rectangles in the available space. The Liquid Diagrams framework implements two different layout algorithms - slice-and-dice and squarified.

The implementation of the slice-and-dice algorithm is shown in Listing 7.5. The recursive function `drawSliceDiceTreeMap` is called to draw all the child notes of the `TreeNode` element handed to the function. Thus it loops over all child elements and computes their width, height, x-position, and y-position in respect to the drawing alignment (lines 4-14). The drawing alignment is determined by the available space of the parent. If the width is larger than the height, the children are drawn vertically. If the currently processed child element does not have children of its own, it is drawn and thus added to the diagram (line 19). If the current node has children, the element is not drawn, but instead the `drawSliceDiceTreeMap` function is called to draw the children of this node in its computed space (line 17).

The second implemented layout algorithm is more complex than the slice-and-dice algorithm. The squarified treemap algorithm was introduced in Bruls et al. [1999] and is the best treemap layout algorithm with regard to the aspect ratios. This is because the algorithm is designed to reduce aspect ratios to as close to 1 as possible. The `drawSquarifiedTreeMap` function shown in Listing 7.6 is a recursive function which implements the squarified treemap algorithm. The function is handed the parent element, its width, height, x-position, and y-position as well as the sum of all child elements values (line 1). The



(a) A treemap layout created by the slice-and-dice algorithm. The order of the elements in the data set is preserved, but aspect ratios vary widely and some nodes are represented by very thin strips.



(b) The squarified treemap algorithm was used to generate this treemap layout. While the order is not preserved, the aspect ratios are much better.

Figure 7.9: Two variants of the treemap visualisation. The data set shown in the visualisation was assembled by gathering the statistics of 900 players from the web site of the National Hockey League [NHL, 2010].

```
public function drawSliceDiceTreeMap(treeNode:TreeNode, restSize:Number, mapWidth:
    Number, mapHeight:Number, x:Number, y:Number) : void {
  var currentX:Number = x, currentY:Number = y;
  for(var i:int = 0; i < treeNode.children.length; i ++) {
    if(drawVertically(width, height)) {
      width = (mapWidth / restSize) * squareElements[i].node.values[square_size.
          selectedIndex];
      height = mapHeight;
      currentX = currentX + width;
      currentY = y;
    } else {
      height = currentHeight = (mapHeight / restSize) * squareElements[i].node.
          values[square_size.selectedIndex];
      width = mapWidth;
      currentX = x;
      currentY = currentY + height;
    }

    if((treeNode.children.source[i]).children != null) {
      drawSliceDiceTreeMap(treeNode.children.source[i], treeNode.children.source[i
          ].values[square_size.selectedIndex], width, height, currentX, currentY);
    } else {
      // draw element
    }
  }
}
}
```

Listing 7.5: The slice-and-dice layout algorithm for treemaps.

goal of the function is to lay out all of the parent's children in the available space with the lowest aspect ratios possible. In order to work correctly, the child nodes need to be sorted in descending order of their size (line 2). Then a decision is made whether to draw the following rectangles horizontally or vertically (line 35). Afterwards the width, height, and aspect ratio of the first rectangle is computed (lines 35-71). The aspect ratio is computed by taking the maximum of either (width/height) or (height/width). After computing the aspect ratio of the placed rectangle, a check is made whether the aspect ratio has improved (grew closer to 1) or got worse (line 7). If the aspect ratio improved, the next rectangle is computed (lines 35-71) and the aspect ratio is checked again. This continues as long as the aspect ratio improves or until all rectangles are computed. If the aspect ratio is worse than before, all computed rectangles but the last (which caused the bad aspect ratio) are finally placed at their computed position (lines 8-18). Afterwards, the `drawSquarifiedTreeMap` function is called with all the remaining child notes (including the one that caused the bad aspect ratio) to be laid out in the remaining parent space (lines 20-26). If a placed child element itself possesses child nodes, the `drawSquarifiedTreemap` function is called using this element as parent to compute the layout of its children, which are then placed into the available space instead of the parent. Due to its complexity an example of how the squarified treemap algorithm works can be found in Section 8.1.

7.8 Heat Map

As introduced in Section 2.4.9 thematic heatmaps or so-called choropleth maps are used to visualise geographical data. Geographical maps of regions are coloured in a way that each entity (for example, a country) is given a colour reflecting the value assigned to it in the data set. The colouring concept is very

```
public function drawSquarifiedTreeMap(treeNode:TreeNode, index:int, restSize:
    Number, mapWidth:Number, mapHeight:Number, x:Number, y:Number) : void {
  treeNode.sortChildren(square_size.selectedIndex);
  var currentAspectRatio = 100, oldAspectRatio = 100, alreadyPlaced = 0;;
  for(var i:int = 0; i < treeNode.children.length + 1; i ++) {
    if((currentAspectRatio>oldAspectRatio)||((index+i)==treeNode.children.length))
        {
      for(var j=0;j<(i-1) || ((index+i)>=treeNode.children.length && j==0);j++) {
        alreadyPlaced = alreadyPlaced + element[j].node.value;
          // draw the element or call drawSquarifiedTreeMap (if children)
      }
      if((index + i) <= treeNode.children.length) {
        if(drawVertically(width, height)) {
          drawSquarifiedTreeMap(treeNode,index+(i-1),restSize-alreadyPlaced,width-
              element[0].oldWidth,height,x+(element[0].oldWidth),y);
        } else {
          drawSquarifiedTreeMap(treeNode,index+(i-1),restSize-alreadyPlaced,width,
              height-(element[0].oldHeight),x,y+(element[0].oldHeight));
        }
      }
      return;
    }
    oldAspectRatio = currentAspectRatio;
    var aspectRatios:Array = new Array(), sizeToPlace = 0;
    element[i] = new SquareElement(i, treeNode.children.source[index + i]);
    if(drawVertically(width, height)) {
      element[i].width = (mapWidth/restSize) * element[i].node.value;
      for(j = 0; j <= i; j ++) {
        sizeToPlace = sizeToPlace + element[j].width;
      }
      for(j = 0; j <= i; j ++) {
        element[j].oldWidth = element[j].newWidth;
        element[j].oldHeight = element[j].newHeight;
        element[j].newHeight = element[j].width/(sizeToPlace/mapHeight);
        element[j].newWidth = sizeToPlace;
        aspectRatios.push(element[j].getMaxAspectRatio(sizeToPlace, element[j].
            newHeight));
      }
    } else {
      element[i].height = (mapHeight / restSize) * element[i].node.value;
      for(j = 0; j <= i; j ++) {
        sizeToPlace = sizeToPlace + element[j].height;
      }
      for(j = 0; j <= i; j ++) {
        element[j].oldHeight = element[j].newHeight;
        element[j].oldWidth = element[j].newWidth;
        element[j].newWidth = element[j].height/(sizeToPlace/mapWidth);
        element[j].newHeight = sizeToPlace;
        aspectRatios.push(element[j].getMaxAspectRatio(element[j].newWidth,
            sizeToPlace));
      }
    }
    aspectRatios.sort(Array.DESCENDING | Array.NUMERIC);
    currentAspectRatio = aspectRatios[0];
  }
}
```

Listing 7.6: A recursive function implementing the squarified treemap algorithm. An example of how it is applied can be found in Section 8.1.

similar to the colouring concept used in treemaps. Heatmaps have grown very popular over the last few years and there are several existing software solutions (see Section 4).

In contrast to most existing software solutions, the Liquid Diagrams heatmap visualisation offers functions to export heatmaps as scalable vector graphics. In addition, the heatmap visualisation provides:

- **SVG Maps**
  Since the drawn regions are completely based on scalable vector graphics, the complete content is resizable. If desired, the content can be scaled to be displayed in the whole available space. Upto now, 13 maps including a world map (shown in Figure 7.11), all six continents, and a few European countries are available.

- **Colours**
  The attribute which is mapped to colour can easily be changed interactively by selecting another attribute frome those supplied with the data set from the colour combo box. Each attribute is assigned a predefined colour value. It is also possible to change the assigned colour, by simply clicking on one of the colour bins in the legend, which opens a colour chooser to pick a new colour. The colour distribution among the entities is also changeable by clicking on the appropriate distribution button in the legend. Figure 7.10 shows different heatmap visualisations using different colour distributions.

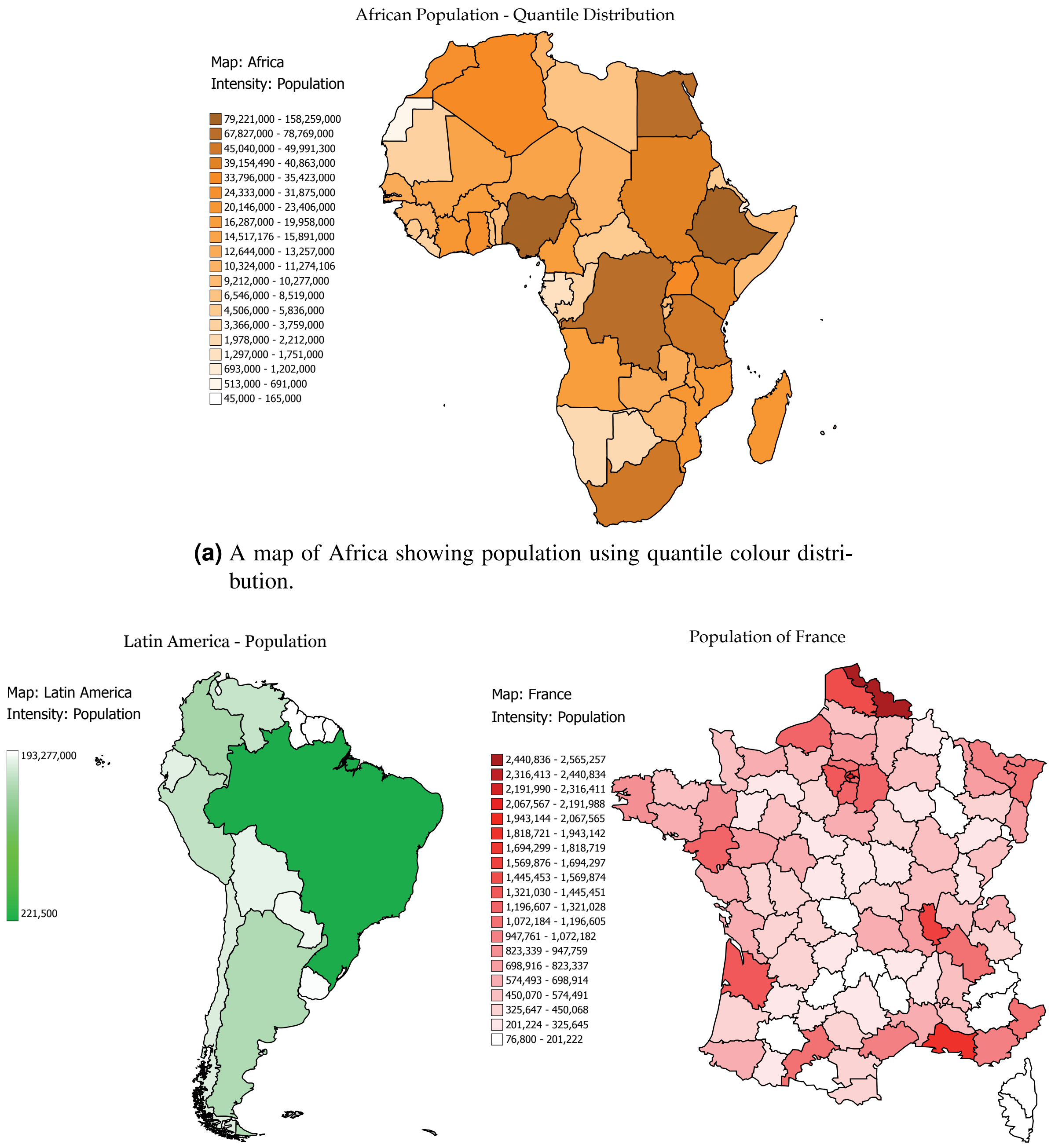
- **Selective Content**
  There is an option to only draw specific regions of those contained in a map. For example, if users want to visualise only the Benelux countries using the Europe map, they can do this by providing data only for the Benelux countries and turning on the appropriate option in the Options Panel.

- **Zooming**
  Depending on the input data, the heatmap visualisation automatically chooses an appropriate map (common region) to display all the country and region codes present in the data file. If the data and an appropriate map for a country is present, this country can be clicked on the common map to zoom to it. Thus the data of the chosen region is shown on the specific detail map. For example, if data values of Austrian and French provinces are given, the Europe map will be used to initially display all the values. By clicking on Austria, a switch to the Austria map occurs, displaying the data supplied for the Austrian provinces.

- **Interaction**
  When exploring the visualisation, detailed tooltips composed from the provided data sets are displayed using mouse-over effects to enhance the understanding of the displayed content. Also, zooming effects are accessible using mouse clicks.

The Liquid Diagrams heatmap visualisation uses a custom XML file (`data.xml`) as lookup table to define which countries and regions are available to be visualised. Until now there are 228 country nodes and 6 regions present in the lookup table. Each country is uniquely defined using its ISO-3166 ALPHA-2 code [ISO, 2010] as id. This standard was published in 1974 by the International Organization for Standardization and assigns each country an unique two-letter country code. Along with this code, each country node is assigned the full text name supplied by [ISO, 1998] as well as the path of the individual SVG map of this country and the names of regions containing this country. Besides countries, regions are also included in the lookup file. Regions can be defined as specific locations that supply a map which includes one or more countries present on this map. For example, all the continents are present as regions in the lookup table. Each region tag consists of a unique id, a name, the names of parent regions hosting this region and a link to the SVG map of this region. Listing 7.7 shows a snippet of the `data.xml` file.

To visualise data using a heatmap visualisation, the user needs to enter the data according to a specific format. The ISO-3166 ALPHA-2 country and / or region code [ISO, 2010] must be the first entry



(a) A map of Africa showing population using quantile colour distribution.

(b) A map of population in Latin America using continuous colour distribution.

(c) A map of the French départements, showing their population using quantile colour distribution.

Figure 7.10: The Liquid Diagrams heatmap gadget enables the creation of interactive choropleth map visualisations.

```
<data>
  <regions id="regions">
    <region id="Europe" parent="World" name="Europe" map="europe.svg"/>
    <region id="Asia" parent="World" name="Asia" map="asia.svg"/>
    <region id="Oceania" parent="World" name="Oceania" map="oceania.svg"/>
    <region id="Latin America" parent="World" name="Latin America" map="lamerica.
        svg"/>
    <region id="North America" parent="World" name="North America" map="namerica.
        svg"/>
    <region id="Africa" parent="World" name="Africa" map="africa.svg"/>
  </regions>
  <countries id="countries">
    <country id="AT" parent="Europe" name="Austria" map="austria.svg"/>
    <country id="DE" parent="Europe" name="Germany" map="germany.svg"/>
      <country id="KZ" parent="Europe,Asia" name="Kazakhstan" map=".svg"/>
    <country id="CH" parent="Europe" name="Switzerland" map="switzerland.svg"/>
  </countries>
</data>
```

Listing 7.7: Part of the data.xml XML file regions and countries available in the heatmap visualisation.

of each data entity. Using the codes, all the entities in the spreadsheet are looked up in the `data.xml` file and an appropriate map to cover all countries is selected. If several countries are present, each entity's parents are looked up and a common parent region is computed. This is implemented using the TreeNode class to build up a hierarchical tree structure based on the lookup table and the provided data set (more information about the TreeNode class can be found in Section 8.2).

In order to create a heatmap, the Liquid Diagrams framework uses scalable vector graphic (SVG) maps containing the shape information about regions. As in Section 3.7, an SVG file is built up using XML tags to describe elements like shapes, texts and images. The SVG maps used in the heatmap visualisation define region shapes using the SVG path element. An example for such an SVG file is given in Listing 7.8. Each path node contains an id element identifying the region. This id comprises the map id and the ISO-3166 ALPHA-2 code of the country which the path belongs to. Besides the id, there is also an optional name element containing the full text name of the entity and most important, the information about how to draw the shape. A detailed example of data definitions, SVG maps, and how new SVG maps are added to the visualisation can be found in Section 8.3.

After determining a common parent the visualisation application reads the contents of the appropriate SVG map. To draw the appropriate shapes, a slightly modified version of a SVG parser written by Knapitsch et al. [2009] is used. This parser reads the path definitions and translates them to corresponding Flex commands which are used to draw the shapes. More information on the SVG parser and its supported SVG tags can be found in Section 8.4. Based on the `DrawCompleteMap` setting in the Options Panel, either all shape definitions or only the definitions of entities present in the data file are read from the SVG map and drawn onto a Panel element. This panel element is then sized according to the available diagram space and added to the ChartPanel giving the final visualisation.

The components and steps necessary to create a heatmap visualisation (illustrated in Figure 7.12) are:

1. The data set is handed to the heatmap gadget.
2. The gadget extracts the ISO country or region codes from the data set and retrieves information about the corresponding countries or regions in the lookup file.

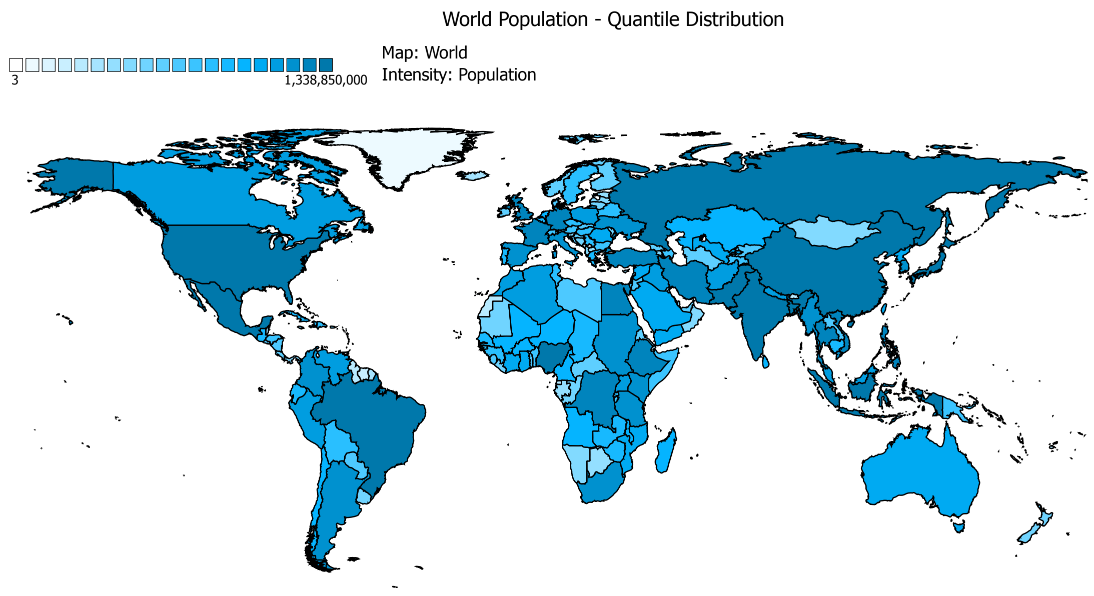


Figure 7.11: A world map of population by country, using the quantile colour distribution.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG
    /1.1/DTD/svg11.dtd">
<svg
  version="1.1"
  id="Africa"
  xmlns="http://www.w3.org/2000/svg"
  width="1425"
  height="1400">

<path
  id="Africa_YT"
  name="Mayotte"
  d="M1206.787,976.599c1.022-1.438,2.275-4.51-0.188-5.475C1206
      .03,972.382,1205.821,975.407,1206.787,976.599z"/>

<path
  id="Africa_SH"
  name="Saint Helena"
  d="M320.129,1057.867c0.731-1.09,0.82-2.216,0.265-3.378C319
      .01,1055.423,318.917,1056.795,320.129,1057.867z"/>
</svg>
```

Listing 7.8: Part of the SVG map of Africa, showing the country outlines of Mayotte and Saint Helena.

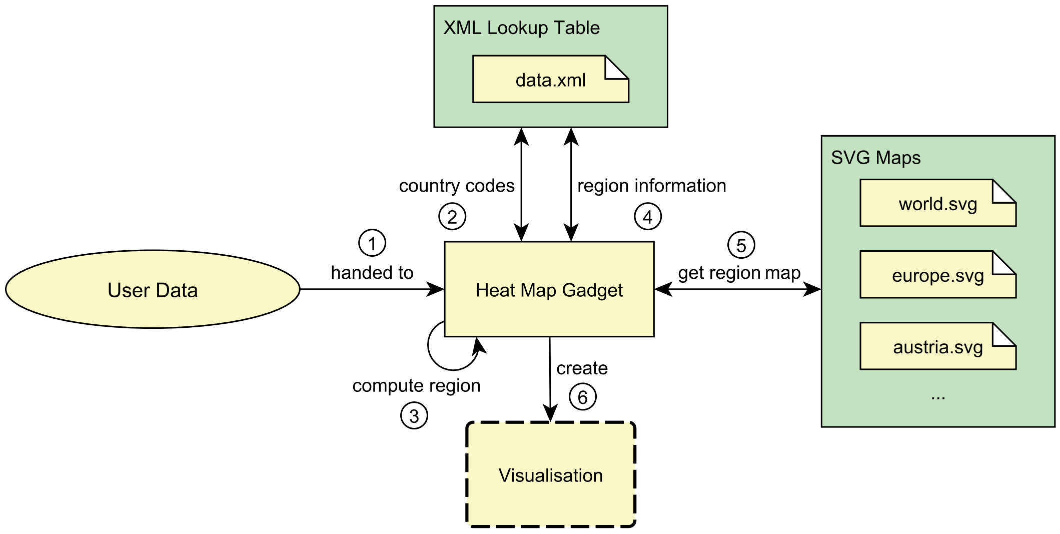


Figure 7.12: The components and steps necessary to create a heatmap visualisation.

3. Using the country information provided by the lookup file, a hierarchical data structure is created (TreeNode and used to compute the common region of all data entities present in the data set.
4. The lookup file is consulted once more to find the path of the common region's SVG map.
5. The SVG map of the common region is acquired and parsed.
6. The parsed information is used to draw the shapes of the regions and thus finish the creation process.

7.9 Voronoi Tree Map

Voronoi tesselations (see Section 2.4.10), named after the Russian mathematician George Voronoi, divide the available space among a number of given locations (called sites), according to the nearest-neighbour rule. Each site p is assigned the region closest to p (illustrated in Figure 7.13) [Aichholzer and Aurenhammer, 2002].

In the notation of Okabe et al. [2000], assume there are a finite number of n points labelled $p_1, ..., p_n$ with the location vectors $(x_1, ..., x_n)$ in the Euclidean plane with $2 \leqslant n \leqslant \infty$. The points are distinct in the sense that $x_i \neq x_j$ for $i \neq j, i, j \in I_n = 1, ..., n$. The Euclidean distance between a point p with the coordinates (x_1, x_2) or a location vector x and a point $p_i \in P$ is given by

$$d(p, p_i) = \parallel x - x_i \parallel = \sqrt{(x_1 - x_{i1})^2 + (x_2 - x_{i2})^2}$$

If p_i is the nearest point from p or p_i is one of the nearest points from p, the relation $\parallel x - x_i \parallel \leqslant \parallel x - x_j \parallel$ for $j \neq i, i, j \in I_n$ is given. The region

$$V(p_i) = x | \parallel x - x_i \parallel \leqslant \parallel x - x_j \parallel forj \neq i, j \in I_n$$

is called the planar ordinary Voronoi polygon associated with p_i, and the set given by

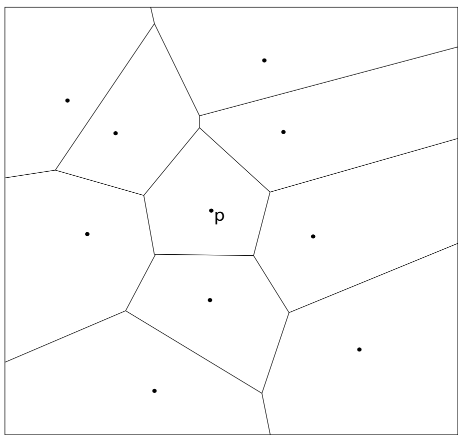


Figure 7.13: A Voronoi diagram. Ten sites (generators) divide the available space into 10 regions.

$$V = V(p_1), ..., V(p_n)$$

is called the Voronoi diagram generated of P.

Since the points on a bisector $b(p_i, p_j)$ are equi-distant to the generator points p_i and p_j the bisector divides the plane into two half planes. This division concept is used in Voronoi treemaps. In Voronoi treemaps each entity is represented by a polygon whose area is sized according to a chosen data attribute. Each entity is assigned a point (site) and the bisectors between the points are constructed. An ordinary Voronoi tessellation draws the bisector equi-distant to the sites and does not take account of any weighting.

The Liquid Diagrams Voronoi treemap gadget uses weighted Voronoi diagrams to create the visualisation. In weighted Voronoi diagrams, each site $\{p_1, ..., p_n\}$ is assigned a weight $\{w_1, ..., w_n\}$. The weights of both sites influence the position of the bisector between between them. There are four types of weighted Voronoi diagrams: multiplicatively weighted, additively weighted, additively power weighted, compoundly weighted. Until now only one type of weighted Voronoi diagram is implemented and used in the Liquid Diagrams Voronoi gadget - the additively weighted power diagram. According to Okabe et al. [2000] the power diagram is characterised by

$$d(p, p_i; w_i) = \parallel x - x_i \parallel^2 - w_i$$

which is called the additively weighted power distance. The bisector of two points p_i and p_j is a straight line passing through the point x_{ij} given by

$$x_{ij} = \frac{\parallel x_j \parallel^2 - \parallel x_i \parallel^2 + w_i - w_j}{2 \cdot \parallel x_j - x_i \parallel^2} \cdot (x_j - x_i) \tag{7.1}$$

The formula shown in (7.1) is used in the Voronoi gadget to determine the bisector of two given sites (generators). A detailed example how the bisector of two sites is computed can be seen in Section 8.5.3.

With additively weighted power Voronoi diagrams the Liquid Diagrams Voronoi gadget is able create polygon regions whose areas are determined by the weight factor applied to the polygon's generator point (site). Simply assigning each generator a weight according to its value, will not result in the desired

Voronoi diagram in which each entity's size is proportional to its value. To achieve polygon sizes which are proportional to their assigned data attribute, an iterative process of generating weighted Voronoi diagrams, adjusting the weights, and moving the generators (sites) to the centre of the polygon as described in Andrews et al. [2002] and Balzer and Deussen [2005] has to be carried out. The desired result is illustrated in Figure 7.14.

The Liquid Diagrams Voronoi treemap gadget uses the SVG export functions, colour legend, and tree view component introduced in Section 6.3 and Section 6.4 In addition the Liquid Diagrams Voronoi treemap visualisation offers the following functions and features:

- **Shapes**
  The visualisation gadget lets the user choose the main shape of the whole visualisation. The user can choose among space-filling rectangles, triangles, and regular polygons. When choosing regular polygons, the user is able to enter the order (number of sites) of the polygon. An example of a polygon of order 35 can be seen in Figure 7.14a.

- **Change Display Attributes**
  The data attributes corresponding to size and colour can be changed interactively by selecting another data attribute from either the size or the colour combo box located in the legend or in the Options Panel.

- **Visual Changes**
  The width of borders ca be changed in order to enhance the recognition of hierarchical structures. The gadget implements user interactions like mouse-over tooltips to obtain more information about specific polygons. There is also an option to turn on polygon labelling. Examples of diagrams with larger border values and polygon labelling can be seen in Figure 7.14c and Figure 2.12b.

- **Logarithmic Scale**
  In order to enhance the visibility of small value entities, the gadget offers the option to use a logarithmic scale. This can easily be activated by clicking on the `LogarithmicScale` check box in the legend panel.

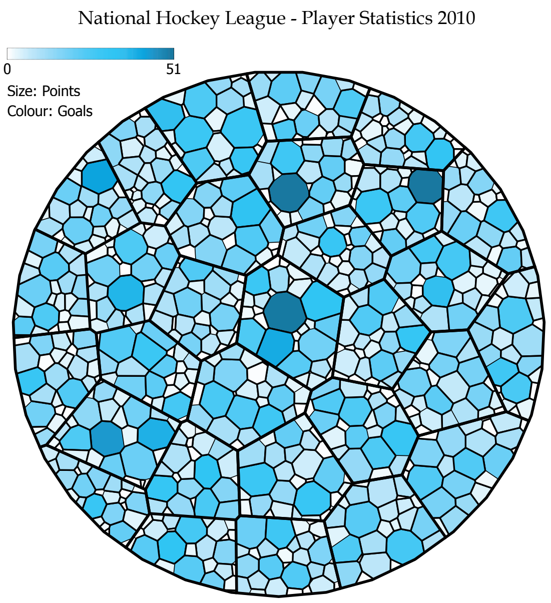
- **Zoom**
  The Voronoi gadget implements the tree view component, enabling the user to click on parent nodes in the tree view to zoom to the clicked element. If a leaf item is clicked the specific leaf is assigned an effect in the visualisation.

- **Computation Settings** The Options Panel hosts several text fields to configure the computation algorithm. Manipulating these settings influences the computation time and the precision of the resulting diagram.

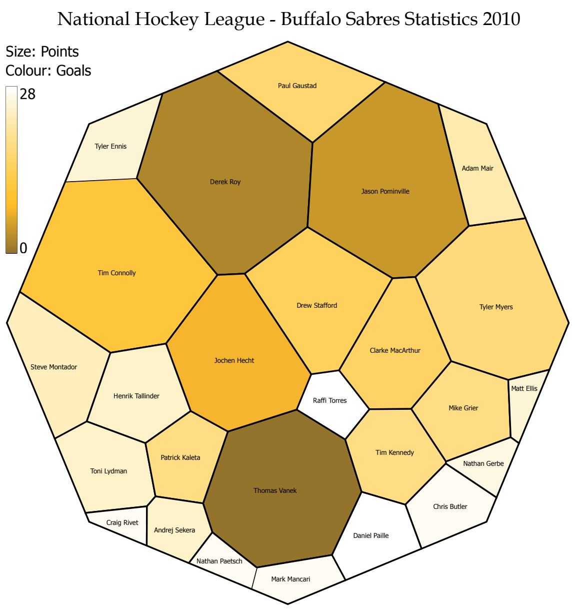
The implementation of the Voronoi treemap gadget is based on a winged-edge data structure introduced in Baumgart [1975]. The winged-edge data structure keeps track of information about each polygon and enables quick traversal between faces, edges, and vertices. Detailed information about the winged-edge data structure can be found in Section 8.5.1.

There are several possibilities to compute the Voronoi diagram for n points. Aurenhammer and Klein [1999] describes three different approaches to compute a Voronoi diagram as follows:

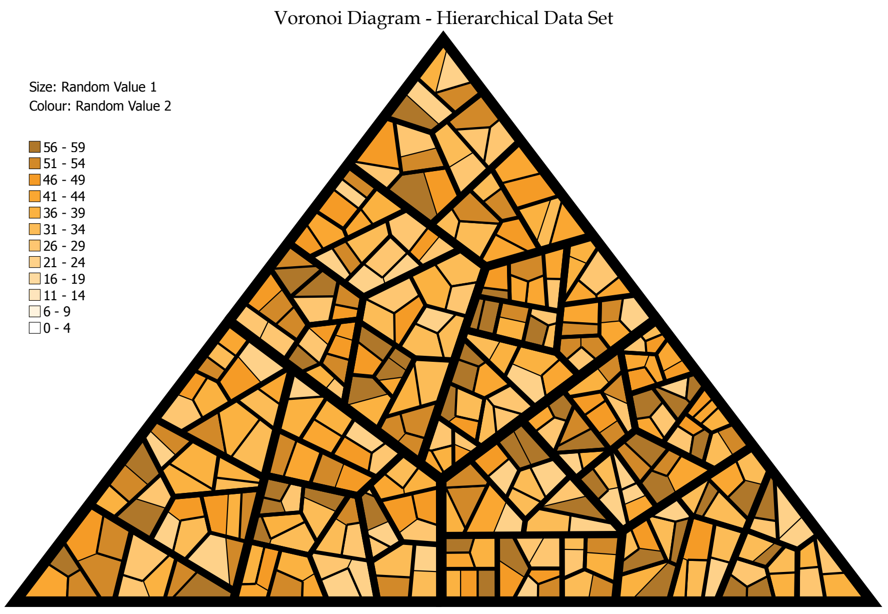
- **Incremental Construction**
  Incremental construction builds up the voronoi diagram by incremental insertion to obtain $V(S)$



(a) A Voronoi Treemap showing the point and goal statistics of over 900 hockey players playing in the National Hockey League.



(b) The player statistics of the Buffalo Sabers displayed in a regular polygon shaped Voronoi Treemap. This view was reached after using the zoom function on a parent polygon of the Voronoi diagram in (a).



(c) A Voronoi Treemap hosting a randomly generated data set. The data set consists of 553 entities with a depth-level of 6.

Figure 7.14: The Liquid Diagrams Voronoi treemap visualisation.

from $V(S)$ $\{p\})$. The benefit of this construction method is that vertices only appearing in intermediate diagrams do not need to be stored or constructed. Incremental construction takes an average runtime of $O(n)$ for well distributed sets of sites.

- **Divide and Conquer**
  The divide and conquer algorithm splits the set of point sites, S, into same sized subsets L and R. Afterwards, the two Voronoi diagrams $V(L)$ and $V(R)$ are computed recursively. The sets $V(L)$ and $V(R)$ are then merged to obtain $V(S)$. The benefit of the divide and conquer algorithm is its construction time of $O(n \cdot logn)$.

- **Sweep**
  In the sweep algorithm a vertical line is moved along the plane and the intersections of the n line segments are computed. Some modifications are made to the sweep-line algorithm in order to apply it to Voronoi diagrams. The plane sweep algorithm also constructs Voronoi diagrams in $O(n \cdot logn)$ time.

In the Liquid Diagrams Voronoi visualisation incremental construction is used to build up the Voronoi structure. The initial polygon is created based on the available space and the chosen shape. Then a number of n random points are generated one after another with n being the number of data entities.

This is illustrated in Listing 7.9. First of all, a minimum distance between the generated points is computed. This is based on the size of the available area (line 3). Then the parent polygon (initial shape) boundaries are computed (line 4). Afterwards a random point is generated within the boundaries of the parent polygon. After checking that the point is not within the minimum distance (line 8-12) and inside the polygon (line 13-17) (see Section 8.5.2) the point is accepted. Then a weight is generated for this point based on the minimum distance and the corresponding data value (lines 19-25). Afterwards the point is inserted into the existing Voronoi diagram (line 26) (see Section 8.5.3).

After adding all points to the Voronoi diagram the initial Voronoi diagram is computed. The iterative process to approximate the Voronoi area sizes, to reflect the value distribution of the data set, is then started. This process is illustrated in Listing 7.10. In line 3 the area sizes are compared to the desired area sizes. If the actual areas are not within a user-defined error tolerance, either the generators are moved to the polygons centre (line 8) or the weights of each polygon are adjusted according to the difference between actual and desired area (line 10) (see Formula (7.2)). With either the weight or the generator positions changed, the Voronoi diagram is computed again and the process starts once again.

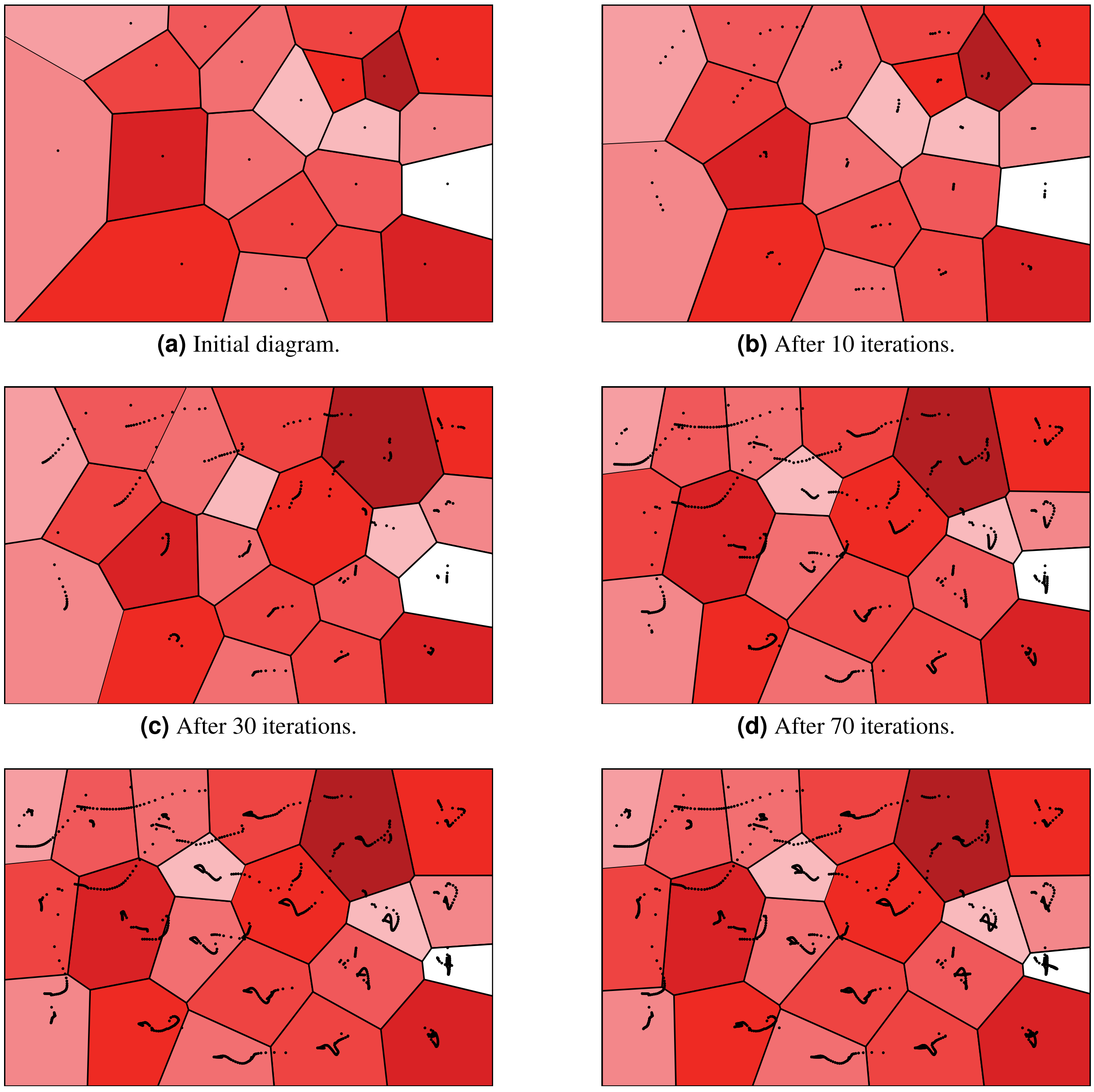
The adjustment of weights is based on the algorithm shown in Balzer and Deussen [2005]. The formula to compute the adjusted weight is

$$adjustedWeight = actualWeight \cdot (1 + \frac{desiredArea - actualArea)}{desiredArea}) \tag{7.2}$$

The iterative process stops if the areas are within the user defined error tolerance and the last computed Voronoi diagram is accepted. An illustration of this process can be seen in Figure 7.15.

```
public function createGenerators(area:Number, parentVoronoiStructure:
    WingedEdgeStructure, parentPolygonNumber:int) : String {
  voronoiGenerators = new Array();
  var minimumDistance:Number = Math.sqrt(area / generatorCount) * 25 / 100;
  var boundary = parentVoronoiStructure.getPolygonBoundary(parentPolygonNumber);
  for(i = 0; i < generatorCount;) {
    var rejected:Boolean = false;
    voronoiGenerators[i] = new Point(lowestX + (Math.random() * boundary.right),
        lowestY + (Math.random() * boundary.top));
    for(var j:int = 0; j < voronoiGenerators.length - 1 && !rejected; j++) {
      if(Point.distance(voronoiGenerators[i], voronoiGenerators[j]) <
          minimumDistance) {
        rejected = true;
      }
    }
    if(parentVoronoiStructure != null && rejected == false) {
      if(!parentVoronoiStructure.isPointInsidePolygon(voronoiGenerators[i],
          parentPolygonNumber)) {
        rejected = true;
      }
    }
    if(!rejected) {
      if(minimumDistance > 100) {
        absoluteWeights[i] = percentualWeights[i] * 1200;
      } else if(minimumDistance > 30) {
        absoluteWeights[i] = percentualWeights[i] * 100;
      } else {
        absoluteWeights[i] = percentualWeights[i] / 100;
      }
      parentVoronoiStructure.addPoint(voronoiGenerators[i], weight);
      i++;
    }
  }
}
```

Listing 7.9: The iterative creation of generator points assigned to a data entity and inserted into the existing Voronoi diagram.



(a) Initial diagram.

(b) After 10 iterations.

(c) After 30 iterations.

(d) After 70 iterations.

(e) After 135 iterations.

(f) The final diagram after 194 iterations.

Figure 7.15: The iteration process to approximate each polygon's area to a desired value reflecting the assigned entities data value. The points illustrate the position of the polygons generator point and the move process each generator underwent. The darker the fill colour of a polygon, the higher its corresponding data value.

```
var centerNext:Boolean = true;
while(!finished && iteration < maximumBaseLevelIterations) {
  var result:Array = voronoiGenerators[instanceNumber].checkPolygonAreaSizes(area,
       voronoiStructure[instanceNumber], errorTolerance);
  var actualAreas:Array = result[1], desiredAreas:Array = result[2];

  if(result[0] == false) {
    if(centerNext) {
      voronoiGenerators[instanceNumber].moveGeneratorsToCenter(parentNode,
          voronoiStructure[instanceNumber]);
    } else {
      voronoiGenerators[instanceNumber].adjustPolygonWeights(actualAreas,
          desiredAreas);
    }
    centerNext = !centerNext;

    createVoronoiDiagram(instanceNumber, parentNode, voronoiGenerators[
        instanceNumber], parentVoronoiStructureNumber, parentPolygonNumber);
    iteration++;
  }
}
```

Listing 7.10: The iterative process carried out to approximate the Voronoi area sizes to reflect the value distribution of the data set.

Chapter 8

Selected Details of the Implementation

“ The whole is more than the sum of its parts. ”

[Aristotle, Greek Philosopher (384 BC - 322 BC).]

8.1 Squarified Tree Map Example

This section shows an example how the squarified treemap algorithm computes the layout of the treemap. The implementation of the algorithm was shown in Listing 7.6 and a brief summary of the algorithm was given in Section 7.7.

8.1.1 Starting Situation

A data array containing 7 data values [5,6,2,8,1,2,6] is handed to the layout algorithm function. The available space for the parent element was set to a width of 600 pixels and a height of 500. The result of each step is illustrated in Figure 8.1.

8.1.2 Step 1

On the first execution the given data array is sorted to contain the data in descending order [8,6,6,5,2,2,1]. Since no rectangles have yet been drawn, the whole space of 600x500 is available. The drawing direction is chosen by looking at the width and height. The width (600) exceeds the height (500) and thus the rectangles will be drawn vertically until the next recursive call. Until now there were no rectangles drawn and thus there is still a value sum of 30 to be drawn (8+6+6+5+2+2+1).

Then the next value (8) is taken from the array. The width of the rectangle is computed by looking at the rectangle's value in respect to the available width (Equation 8.1). Thus there is currently only one rectangle examined the width of the rectangle is also the summed up width of the examined rectangles (Equation 8.2). Then the height of the rectangle is computed in respect to the available height (Equation 8.3). The last computation is to calculate the aspect ratio of the newly created rectangle. This is done by taking the maximum of the rectangle's (with/height) and (height/width) (Equation 8.4).

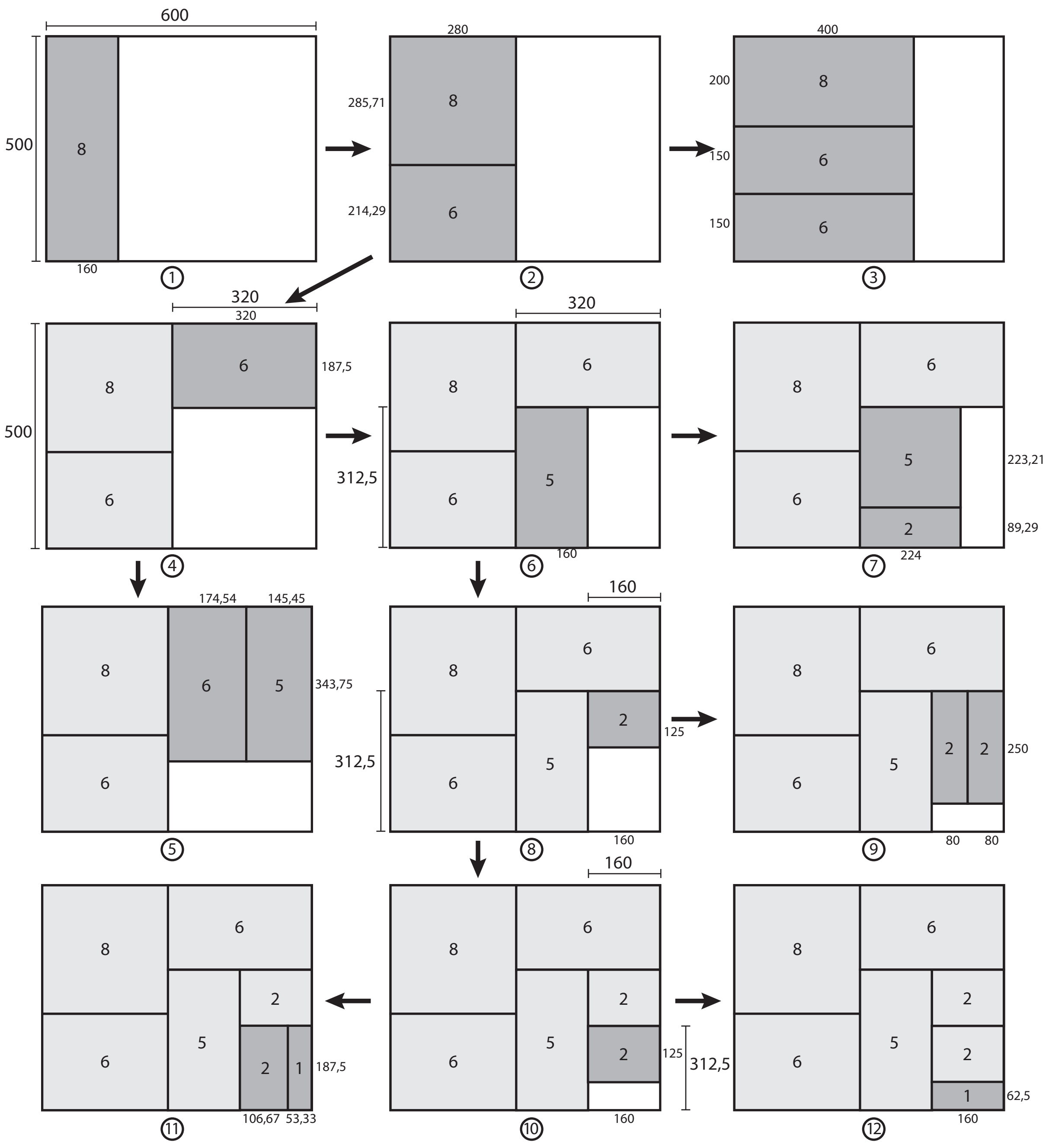


Figure 8.1: The squarified treemap layout algorithm. The numbers below the images refer to the step number. Darker colours indicate that this rectangle underwent changes in the current step. Lighter colours indicate rectangles that have been already drawn in a previous step and will not be changed any more.

$$\begin{aligned}
rectWidth &= widthLeft/valuesLeft \cdot value = 600/30 * 8 = 160 && (8.1)\\
width &= rectWidth = 160 && (8.2)\\
height &= rectWidth/(width/mapHeight) = 500 && (8.3)\\
AspectRatio &= max(160/500), (500/160)) = 3,125 && (8.4)
\end{aligned}$$

8.1.3 Step 2

Since there is currently no old aspect ratio value, the aspect ratio cannot get worse. Since no rectangles are placed until now, there is still an area of 600x500 available and a value sum of 30 to place.

The algorithm carries on by taking the next value from the array [6] and computes its width in the current layout (Equation 8.5). Since there are now two rectangles present the width increases because the summed up values of the rectangles take a larger share of the whole value than just the first rectangle alone (Equation 8.6). The height of each rectangle is then computed in regard to their individual width and the available height space (Equation 8.7). Afterwards, the aspect ratios of both rectangles are computed and the highest value taken into account (Equation 8.8).

$$\begin{aligned}
rectWidth &= widthLeft/valuesLeft \cdot value = 600/30 * 6 = 120 && (8.5)\\
width &= \sum rectWidth = 120 + 160 = 280 && (8.6)\\
height[8] &= rectWidth/(width/mapHeight) = 285,71 && (8.7)\\
height[6] &= rectWidth/(width/mapHeight) = 214,29 &&\\
AspectRatio[8] &= max(280/285,71), (285,71/280)) = 1,020 && (8.8)\\
AspectRatio[6] &= max(280/214,29), (214,29/280)) = 1,307 &&
\end{aligned}$$

8.1.4 Step 3

At the beginning of the third step there are two aspect ratios present for the first time and thus a comparison to the previous step can be done. Since the aspect ratio of the first step was 3,125 and the ratio of the second was 1,307 the addition of the second rectangle was an improvement for the aspect ratio. The second step is confirmed and the algorithm carries on by taking the next value.

After computing the value's width (Equation 8.9) it is added to the width of the other two rectangles already present to give the total width (Equation 8.10). Afterwards the individual heights are computed (Equation 8.11) which are used to determine the aspect ratios (Equation 8.12).

$$\begin{aligned}
rectWidth &= widthLeft/valuesLeft \cdot value = 600/30 * 6 = 120 && (8.9)\\
width &= \sum rectWidth = 400 && (8.10)\\
height[8] &= rectWidth/(width/mapHeight) = 200 && (8.11)\\
height[6] &= height = rectWidth/(width/mapHeight) = 150 &&\\
height[6] &= height = rectWidth/(width/mapHeight) = 150 &&\\
AspectRatio[8] &= max(400/200), (200/400)) = 2 && (8.12)\\
AspectRatio[6] &= max(400/150), (150/400)) = 2,667 &&\\
AspectRatio[6] &= max(400/150), (150/400)) = 2,667 &&
\end{aligned}$$

8.1.5 Step 4

By comparing the aspect ratio of Step 3 and Step 2 an increase in the ratio from 1,307 to 2,667 can be noticed. This means the aspect ratio got worse and the last step needs to be undone.

All steps up to Step 3 (not included) are now confirmed and finally the first and second rectangle (Step 1 and Step 2) are drawn. When taking the implementation of the algorithm shown in Listing 7.6 into account this step would cause a recursive call of the `drawSquarifiedTreeMap` with the changed conditions. After drawing the two rectangles, the available space is reduced to 320x500 and the sum of the values to place is reduced to 16 (6+5+2+2+1). Since the height is now larger than the width, the next rectangles are added horizontally.

The next value is taken from the array [6]. Since the algorithm is now drawing horizontally, the height is computed (Equation 8.13). There are currently no other rectangles taken into account so the height of the single rectangle is also the final height (Equation 8.14). Afterwards the width (Equation 8.15) and the aspect ratio (Equation 8.16) are calculated.

$$rectHeight = heightLeft/valuesLeft \cdot value = 500/16 * 6 = 187,5 \tag{8.13}$$

$$height = \sum rectHeight = 187,5 \tag{8.14}$$

$$width = rectHeight/(height/mapWidth) = 320 \tag{8.15}$$

$$AspectRatio = max(320/187,5),(187,5/320)) = 1,707 \tag{8.16}$$

8.1.6 Step 5

Because the aspect ratio of Step 4 is the first ratio since the first two rectangles were drawn, there is no other ratio to compare it to. Thus the next value [5] is removed from the array and its individual height (Equation 8.17) is computed. Since there are two rectangles the overall height is determined by both of them (Equation 8.18). The individual widths (Equation 8.19) are then used to compute the new aspect ratios of the rectangles (Equation 8.20).

$$rectHeight = heightLeft/valuesLeft \cdot value = 500/16 * 5 = 156,25 \tag{8.17}$$

$$height = \sum rectHeight = 343,75 \tag{8.18}$$

$$\begin{aligned} width[6] &= rectHeight/(height/mapWidth) = 174,54 \qquad (8.19)\\ width[5] &= rectHeight/(height/mapWidth) = 145,45 \end{aligned}$$

$$\begin{aligned} AspectRatio[6] &= max(174,54/343,75),(343,75/174,54)) = 1,964 \qquad (8.20)\\ AspectRatio[5] &= max(145,45/343,75),(343,75/145,45)) = 2,363 \end{aligned}$$

8.1.7 Step 6

The aspect ratio got worse since Step 4, so the 5th step is undone and the rectangle computed in Step 4 is drawn. Another recursion step is taken giving a new available space of 320x312,5 and leading to a vertical alignment. The sum of values to place is reduced to 10.

The next value is taken from the array [5] and its individual width (Equation 8.21) and height (Equation 8.22) is computed. Using this width and height the aspect ratio can be determined (Equation 8.23).

$$rectWidth = widthLeft/valuesLeft \cdot value = 320/10 * 5 = 160 \tag{8.21}$$
$$height = rectWidth/(width/mapHeight) = 312,5 \tag{8.22}$$
$$AspectRatio = max(160/312,5),(312,5/160)) = 1,953 \tag{8.23}$$

8.1.8 Step 7

Since only one rectangle was added since the last recursion step, no comparable ratios are available. The next value is acquired [2] and because the algorithm is still drawing vertically, the individual width of this rectangle is computed (Equation 8.24) and added to the width of the first rectangle (Equation 8.25). Again, the individual heights (Equation 8.26) and widths are used to compute the aspect ratios (Equation 8.27).

$$rectWidth = widthLeft/valuesLeft \cdot value = 320/10 * 2 = 64 \tag{8.24}$$
$$width = \sum rectWidth = 224 \tag{8.25}$$
$$\begin{aligned} height[5] &= rectWidth/(width/mapHeight) = 223,21 && (8.26)\\ height[2] &= rectWidth/(width/mapHeight) = 89,29 \\ AspectRatio[5] &= max(224/223,21),(223,21/224)) = 1,004 && (8.27)\\ AspectRatio[2] &= max(224/89,29),(89,29/224)) = 2,509 \end{aligned}$$

8.1.9 Step 8 and Step 9

Once more, the aspect ratio got worse and so Step 7 is undone. The rectangle created in Step 6 is drawn, resulting in a new available space of 160x312,5. There is only a value sum of 5 (2+2+1) left to be drawn. Since the height exceeds the width, the next rectangles are drawn horizontally. The space assigned to the next value [2] and its aspect ratio is computed (Equation 8.28). Afterwards Step 9 is carried out (no comparable ratios), which adds the next value [2]. The heights are summed up (Equation 8.30) to lead to new aspect ratios (Equation 8.31).

$$\begin{aligned} rectHeight &= heightLeft/valuesLeft \cdot value = 312,5/5 * 2 = 125 && (8.28)\\ width &= rectHeight/(height/mapWidth) = 160 \\ AspectRatio &= max(160/125),(125/160)) = 1,28 \\ & && (8.29)\\ rectHeight &= heightLeft/valuesLeft \cdot value = 312,5/5 * 2 = 125 && (8.30)\\ height &= \sum rectHeight = 250 \\ width[2] &= rectHeight/(height/mapWidth) = 80 \\ width[2] &= rectHeight/(height/mapWidth) = 80 \\ AspectRatio[2] &= max(80/250),(250/80)) = 3,125 && (8.31)\\ AspectRatio[2] &= max(80/250),(250/80)) = 3,125 \end{aligned}$$

8.1.10 Step 10 and Step 11

Again, the aspect ratio got worse (from 1,28 to 3,124), leading to the negation of Step 9. The rectangle created in Step 8 is confirmed and thus drawn, leading to a new recursion step with the available space of 160x187,5. There are only two values left to be drawn with a sum of values of 3.

After computing the aspect ratio of the first rectangle (Equation 8.32) the next step, Step 10 is carried out. This step could be the last step because it adds a rectangle presenting the last value [1] to the diagram. But instead of just drawing this rectangle the computed aspect ratios (Equation 8.34) of the two steps are compared once more.

$$\begin{aligned} rectHeight &= heightLeft/valuesLeft \cdot value = 187,5/3 * 2 = 125 \\ width &= rectHeight/(height/mapWidth) = 160 \\ AspectRatio &= max(160/125),(125/160)) = 1,28 \end{aligned} \tag{8.32}$$

$$\tag{8.33}$$

$$\begin{aligned} rectHeight &= heightLeft/valuesLeft \cdot value = 187,5/3 * 1 = 62,5 \\ height &=? rectHeight = 187,5 \\ width[2] &= rectHeight/(height/mapWidth) = 106,67 \\ width[1] &= rectHeight/(height/mapWidth) = 53,33 \\ AspectRatio[2] &= max(106,67/187,5),(187,5/106,67)) = 1,758 \\ AspectRatio[1] & max(53,33/187,5),(187,5/53,33)) = 3,516 \end{aligned} \tag{8.34}$$

8.1.11 Step 12

Even though the last value would have fitted into the left space the aspect ratios of Step 10 and Step 11 were compared and indicated a higher aspect ratio. Thus only Step 10 is drawn and a final 12th step is necessary to finish the treemap visualisation. The space available for the last value [1] is 160x62,5 leading to a vertical alignment. For the last time the width and height of the rectangle are computed (Equation 8.35) and finally the last element is drawn too, leading to the final treemap layout.

$$\begin{aligned} rectWidth &= widthLeft/valuesLeft \cdot value = 160/1 * 1 = 160 \\ width &= \sum rectWidth = 160 \\ height &= rectWidth/(width/mapHeight) = 62,5 \\ AspectRatios &= max(160/62,5),(62,5/160)) = 2,56 \end{aligned} \tag{8.35}$$

8.2 Tree Structure

In order to process hierarchically structured data, data from the user is written into a custom tree data structure called TreeNode. The implementation of the TreeNode class is located in the `TreeNode.as` file.

Each TreeNode element represents exactly one data entity and possesses the following attributes:

- **id**
  The id is an unique intern identifier used to determine between the nodes. The id is automatically

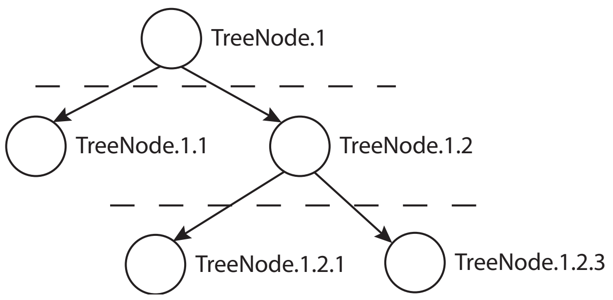


Figure 8.2: The nodes of a three-tier TreeNode structure and how their unique id attributes would look. Each id is built up using a string, its parents id number, and a number identifying the node among the other children of this parent.

assigned and reflects the position of the node at the time it was inserted. Thereby the id is composed of the string ”TreeNode.” followed by the parent identification number (not included if it is the root node) and the own identification number. An example of how this would look like in a three layered TreeNode structure is shown in Figure 8.2.

- **name**
  This attribute represents the name of the node. In data sets used for the heatmap visualisation this attribute indicates the ISO code of the element. The name attribute does not have to be unique.
- **label**
  The label attribute is only used in combination with the heatmap visualisation and contains the name of the country or region (if present in the data set).
- **text**
  This attribute is used to store information about the node. This information is either given by the data set or can be set during runtime. The information is then included in the tooltip generated by a mouse-over effect.
- **values**
  The values attribute is a numerical array holding all data values assigned to this entity.
- **sortIndex**
  Defines the index of the values column which is taken when sorting the data using the sort function. This is needed for the squarified treemap layout algorithm of the treemap visualisation.
- **parent**
  This attribute contains a link to the parent node.
- **children**
  This is an array containing a link to all child nodes of this node.

The TreeNode class provides many functions. These include functions to search for names, attributes, and extreme values in specific depths as well as accessing values such as specific column data of a given depth. There is also a function to determine the common parent of nodes. This function set is needed in order to determine which map file is to be used for a given data set of a heatmap visualisation. Using the sort function, child nodes of a given node can be sorted according to a given column of their data values.

The treemap, heatmap, and voronoi diagram visualisations use the TreeNode class to handle their data. They all share a common input format to which the data handed to the gadget must conform:

- The first row contains the header names.
- Each further row of the array represents one data entity.
- The first element is the name of the entity.
- The second element is the parent of the entity.

Under these assumptions, the tree structure is built up. The implementation is shown in Listing 8.1. A copy of all rows is made and written into the `lines` array. This lines array represents all entities still to be inserted into the tree structure. Then a node item, which will serve as common root item is created (lines 5-6). Afterwards, all elements having an empty parent attribute are added as children to the root element (lines 8-23). During this process, all lines of the nodes which have been added to the root element are removed from the `lines` array (lines 15-19). All entities that are still present in the `lines` array possess a parent and thus need to be assigned to the appropriate parent. This is done by looping over the array elements looking for the entity's parent attribute in the tree structure (lines 26-41). If the parent is present in the tree structure, the node is assigned as a child and removed from the list (line 30). If the node is not present in the tree the entity is skipped to be processed again after the other elements of the array have been processed (line 32). This loop operates as long as entities remain to assigned to their appropriate parents. Once a loop cycle does not assign any nodes, the loop cycle status switches to `idle`. Then a last loop cycle is carried out, in which the parent nodes of all elements still present in the `lines` array are created and assigned to the root node. Afterwards, the elements in the `lines` array are assigned to their added parents, resulting in the completion of the tree structure.

8.3 Adding New SVG Maps

Since the country and region maps used in the heatmap visualisation are based on SVG maps, it is easy to add new maps or change existing maps. This example illustrates what steps have to be taken to add a new map.

The first task is to create or acquire a map of the region to be added. For this example, a Carinthia map taken from Wikimedia [2010] should be added to the maps available in the heatmap visualisation gadget. To ensure that the map loads properly into the gadget, the SVG code has to be revised and changed if it does not conform to the following conventions:

- **Namespace**
  Ensure that the namespace of the document is set to `xmlns="http://www.w3.org/2000/svg`. If the namespace is not set, the SVG tags will not be recognised properly by browsers.
- **Map ID**
  In order to integrate the map into the whole map hierarchy, the `id` of the map document has to be set carefully. The `id` of the map must reflect the ISO-3166 ALPHA-2 code [ISO, 2010] of the region. If the region code is determined by more than one code, the codes have to be separated by underscores. Thus, for the provinces of Carinthia map, the `id` would be `AT_2` because Austria has the ISO code AT and Carinthia has the code `2`.
- **Width and Height**
  The map `width` and `height` attributes are read by the gadget and determine the initial size of the panel used to draw the shapes. The `width` and `height` values should be chosen such that each path command draws within the boundary. Both values have to be numeric only (no ”px” extension).

```
public function handleData() : void {
  var lines:Array = ObjectUtil.copy(values) as Array;
  var idle:Boolean = false, valueLength = values[0][3].length;

  treeRoot = new TreeNode();
  treeRoot.initializeNode("root", "", null, null, valueLength);

  for(var i:int = 0, j:int = 1; i < lines.length;) {
    if(lines[i][1] == '') {
      var treeNode:TreeNode = new TreeNode();
      treeNode.initializeNode(lines[i][0], lines[i][2], treeRoot, lines[i][3],
          valueLength);
      treeRoot.addChild(treeNode, options[COMPUTE_CATEGORY_VALUES]);
      j++;

      if(i+1 != lines.length) {
        lines[i] = lines.pop();
      } else {
        lines.pop();
      }
    } else {
      i++;
    }
  }

  var addParentNextTime:Boolean = false;
  while(idle == false) {
    idle = true;

    for(i = 0, j = 0; i < lines.length;) {
      if(findCategoryName(i, lines[i], lines, treeRoot, addParentNextTime) !=
          false){
        idle = false;
      } else {
        i ++;
      }
    }
    if(idle == true && addParentNextTime == false) {
      addParentNextTime = true;
      idle = false;
    }
  }
}
```

Listing 8.1: Mapping the data entries of an array to a tree structure reflecting the data's hierarchical structure.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
  xmlns="http://www.w3.org/2000/svg"
  width="590"
  height="280"
  id="AT_2">

<path
  id="AT_2_KL"
  name="Klagenfurt Land"
  d="M 437.79299,150.14561 L ..."/>
  ...
<path
  id="AT_2_FE"
  name="Feldkirchen"
  d="M 338.73804,93.377282 L  ..."/>
</svg>
```

Listing 8.2: The adapted SVG code of the Carinthia map acquired from Wikimedia [2010].

```
<country id="AT_2" parent="AT" name="Carinthia" map="carinthia.svg"/>
```

Listing 8.3: The newly created map needs to be registered in the country lookup table.

- **Path ID**
  Each path node needs to have an `id` parameter identifying the path as a region. This `id` value should be assembled by the map id, an underscore, and the ISO region code (if an ISO code exists). For example a region on the Carinthia map would have the `id` `AT_2_SP`.

- **Path Name**
  The `name` attribute of a path node is not required, but if entered, the `name` attribute will be used in the region tooltip if the user data does not supply the region's full name.

- **Path**
  The `d` attribute of the path is the path's data, holding the instructions to draw the path. These instructions are parsed by the SVG parser (see Section 8.4). In order to colour the region in the diagram, the path object must be closed, or the fill operation will not work properly.

- **Transitions**
  Transitions manipulate the position and scale of the path drawn using the instructions. Since only the instructions are read by the parser, transitions will not be applied to the instructions when they are used in the gadget. Therefore, all transitions have to be removed and applied to the `d` attribute.

After adapting the SVG code of the Carinthia map accordingly, it looks like the illustration in Listing 8.2. The next step is to register the map in the country lookup table located in the `data.xml` file. The country is registered with its ISO code, parent, full name, and the file path to the map, as shown in Listing 8.3.

Now that the map is created and registered in the lookup table, it should be possible to use it in the gadget. When handing data to the gadget, it has to be ensured to hand over the right parent and country id (in this case the parent id is AT-02 and the country id is the region). In the case of the Carinthia map, the add process was successful and the result can be seen in Figure 8.3.

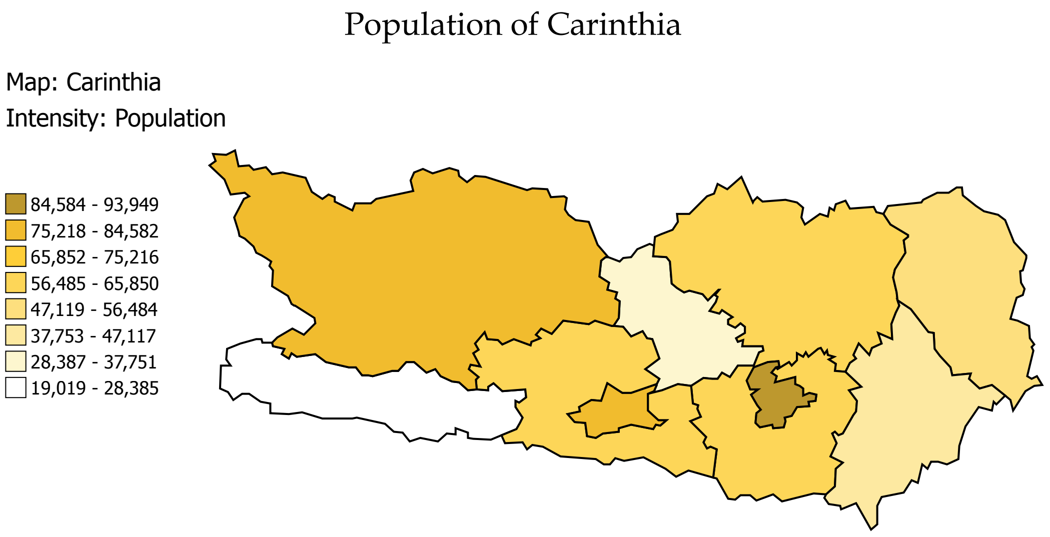


Figure 8.3: The newly added Carinthia map displaying the province's population by district.

8.4 SVG Parser

The SVG parser was originally written by Knapitsch et al. [2009] and was slightly modified to fit into the Liquid Diagrams framework. The implementation is located in the package `diagram.svg` which includes two files `SVGPath.as` and `SVGPathDrawer.as`. The purpose of the SVG parser is to read all the instructions located in path elements of SVG files and translate these instructions into corresponding Flex commands. This commands are then executed to draw the shapes described by the instructions.

The following SVG instruction tags are implemented by the SVG parser. Small letter commands have the same function as capital letters, but their position values are relative to the current position, while capital positions are absolute. The description of each command is taken from [W3C, 2010d].

- **M m**
  The `M` instruction moves to a position and starts a new sub-path. If there are more than two coordinates following the instruction, these coordinates are interpreted as line to commands.

- **L l**
  Draws a line from the current position to the position given by the parameters. More than two coordinates result in the drawing of a multiple line segments.
- **H h**
  Draws a horizontal line starting from the current point.
- **V v**
  The `V` instruction draws a vertical line to the given y value.
- **Z z**
  Close the current path by drawing a straight line to the start point of the path.
- **C c**
  Draws a Bézier curve to a given coordinate. The instruction takes two additional coordinates which are used as the curve's control points.

- **S s**
  This instruction draws a cubic Bézier curve to a point. Only one control point supplied, the second is computed.
- **Q q**
  The `Q` instruction draws a quadratic Bézier curve to a point, using one additional coordinate pair as control point.
- **T t**
  Draws a quadratic Bézier curve to a point.

8.5 Voronoi Construction Details

8.5.1 Winged-Edge Data Structure

The winged-edge data structure was introduced by Baumgart [1975]. The data structure is used to store graph information like edges, vertices, and polygons of the graph and thus enables fast and easy access when needed. As described in Okabe et al. [2000], the graph is closed in order to use it for the Voronoi diagram. This closure is achieved using a closed shape that enwraps all polygon generators.

The winged-edge data structure used in the Liquid Diagrams framework stores the following information to construct Voronoi diagrams:

- **polygonNumber**
  This parameter indicates how many polygons are present in the Voronoi diagram (further referred to as n). The number of polygons reflects the number of data entities, because each entity is presented by a polygon.
- **edgeNumber**
  The `edgeNumber` represents the number of edges present in the diagram. Each edge is directed and thus points from its start vertex to its end vertex. Edges around the start vertex are called predecessors and edges around the end vertex are called successors.
- **vertexNumber**
  The number of vertices. A vertex is a point where two or three edges intersect each other. A vertex can either be a start point or an end point of an edge.
- **polygonGeneratorX, polygonGeneratorY** : [n]
  Each polygon has a generator point (site) which is used as the centre of the polygon. Depending on the weight of the polygon, this might not exactly be its computational centre. The `polygonGeneratorX` and `polygonGeneratorY` variables represent the x-position and the y-position of a polygon's generator point.
- **polygonGeneratorW** : [n]
  This value indicates whether a polygon is present in the in the Voronoi diagram or not. This can happen due to larger polygons forcing smaller polygons to be outside of the boundary.
- **polygonGeneratorWeight** : [n]
  The `polygonGeneratorWeight` parameter is used to store the weight associated with a polygon, or more precisely which is present at the polygon's generator point.
- **polygonDataID** : [n]
  This variable identifies the data entity represented by the polygon.

- **edgeAroundPolygon** : [n]
  Each polygon has at least three edges surrounding the polygon and thus defining the polygon's borders. One of these boundary edges is stored for each polygon. Based on this information, it is easy to obtain all other surrounding edges of a polygon.

- **edgeAroundVertex** : [n]
  A vertex can have up to three edges starting or ending at the vertices position. One of these edges is stored for each vertex.

- **edgeRightPolygon, edgeLeftPolygon** : [n]
  Each edge separates two regions and thus has two bordering polygons. The number of these polygons is stored in the `edgeLeftPolygon` and `edgeRightPolygon` variables, depending on the start vertex and end vertex (= direction). If the edge is on the boundary of the Voronoi diagram, one of these variables is set to `-1`.

- **edgeStartVertex, edgeEndVertex** : [n]
  Each edge is a line between its start vertex and its end vertex. The `edgeStartVertex` and `edgeEndVertex` variables are used to store the number of the edge's start vertex and end vertex.

- **edgeCWPredecessor, edgeCCWPredecessor** : [n]
  The clockwise and counter-clockwise predecessors of an edge are the two other edges present at an edge's start vertex. If there is only one other edge present at a vertex, one of the predecessor variables is set to `-1` (the missing one).

- **edgeCWSuccessor, edgeCCWSuccessor** : [n]
  The successor edges are the two other edges located at the end point of an edge. Boundary edges may have only one successor. In this case, the other successor (the one that is missing) is set to `-1`.

- **edgeW** : [n]
  The `edgeW` variable indicates whether an edge is located at the boundary of the Voronoi diagram or not. If it is a boundary edge, the value is set to `0`.

- **vertexX, vertexY** : [n]
  The `vertexX` and `vertexY` variables are used to store the x-position and y-position of a vertex.

- **vertexW** : [n]
  This value indicates if the point is located at the boundary of the Voronoi diagram. If it is an ordinary point inside the diagram, its value is `1`. If it is located on the boundary and a shape-defining vertex, its value is set to `2`. Shape defining vertices cannot be removed thus the diagram (parent polygon) shape would change if the point is removed. If it is a removable point at the boundary of the diagram, its value is set to `0`.

An example for a winged-edge data structure representing the diagram shown in Figure 8.4 can be found in Table 8.1.

8.5.2 Determining Point Positions

When randomly generating the positions for the polygon generators, it is necessary to determine whether a point is inside the polygon of the parent or not. To find out whether a point is inside a polygon or not, the Liquid Diagram framework uses the ideo of a ray casting algorithm [Sutherland et al., 1974] presented at Finley [2010]. The algorithm has weaknesses if the point is located directly on the boundary of an edge, but this is no problem in our case since such a point is rejected anyway, because points should be inside of the parent shape. Figure 8.5 shows an example polygon and several points inside and outside

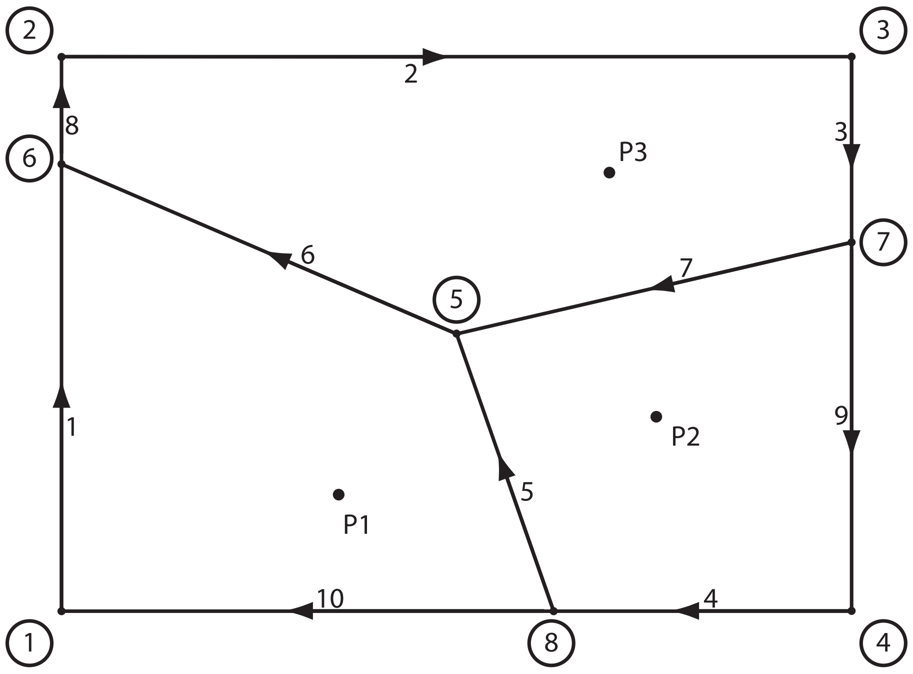


Figure 8.4: The simple Voronoi diagram shown here is the reference for the winged-edge data structure shown in Table 8.1.

edgeNumber	1	2	3	4	5	6	7	8	9	10
edgeW	0	0	0	0	1	1	1	0	0	0
edgeRightPolygon	1	3	3	2	2	3	3	3	2	1
edgeLeftPolygon	-1	-1	-1	-1	1	1	2	-1	-1	-1
edgeStartVertex	1	2	3	4	8	5	7	6	6	8
edgeEndVertex	6	3	7	8	5	6	5	2	4	1
edgeCWPredecessor	10	8	2	9	4	7	3	6	7	5
edgeCCWPredecessor	-1	-1	-1	-1	10	5	9	1	3	4
edgeCWSuccessor	8	-1	9	10	6	1	5	-1	-1	-1
edgeCCWSuccessor	6	3	7	5	7	8	6	2	4	1
polygonNumber	1	2	3							
polygonGeneratorW	1	1	1							
edgeAroundPolygon	1	4	2							
vertexNumber	1	2	3	4	5	6	7	8		
vertexW	2	2	2	2	1	0	0	0		
edgeAroundVertex	1	2	3	4	5	6	7	8		

Table 8.1: The winged-edge data structure according to Figure 8.4

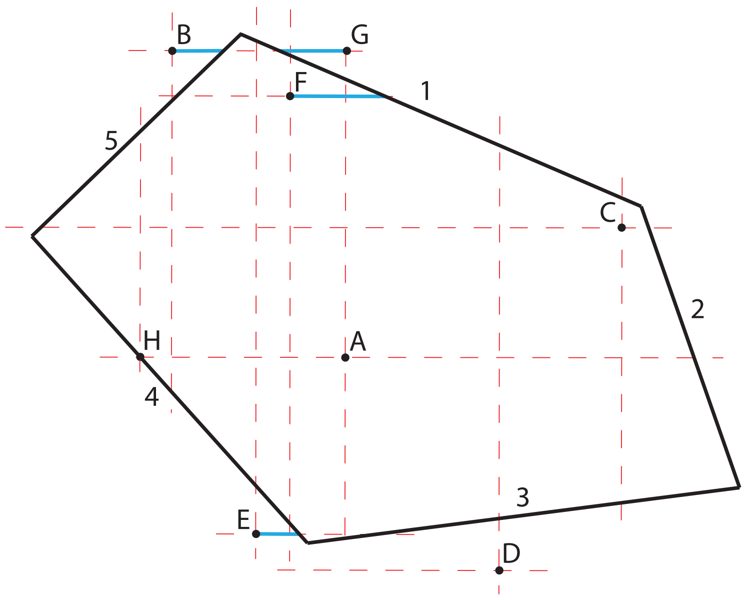


Figure 8.5: For each point labelled from A to H, a computation is carried out in Table 8.2 based on the algorithm shown in Listing 8.4 to determine if the point is inside the polygon. The red dotted lines are only for orientation purposes. The blue lines are the lines leading to the intersection point necessary to compute the position of the point.

of the polygon. Table 8.2 illustrates how the algorithm decides if the polygon is inside or outside of the polygon.

The implementation of the algorithm is illustrated in Listing 8.4. All boundary edges of a polygon are determined and stored in an array (line 2). Afterwards, each boundary edge is checked for whether the y-coordinate of the point is within the y-coordinates of the edge's start vertex and end vertex (lines 6-7). If the y-coordinate of the point is not within the vertices, this edge is skipped and nothing happens. If the y-coordinate of the point is within the y-coordinates of the start and end vertex, three situations can arise:

- The x-coordinate of the point is higher than the x-coordinates of the start vertex and the end vertex. The edge is located to the left of the point and thus is added to the `beforeList` (line 9).
- The x-coordinate of the point is lower than both x-coordinates of the edge.
  The edge is located to the right of the point and is added to the `afterList` (line 11).
- The x-coordinate is between the x-coordinate of the edge's start and end vertex.
  To determine if the edge is added to the `beforeList` or the `afterList` a line parallel to the x-axis is drawn from the point to the edge until the line intersects it. Afterwards, the x-coordinate of the intersection point is computed (line 13-14). If the resulting x-coordinate is higher than the x-coordinate of the point, the edge is added to the `afterList` (line 17). If it is lower, the edge is added to the `beforelist` (line 19). If the x-coordinate of the intersection point is equal to the x-coordinate of the point, the edge is rejected because the point is located on the boundary edge (line 21).

After running through all boundary edges, it is determined whether an odd number of edges is in both the `beforeList` and the `afterList`. If this is the case, the point is inside of the polygon and it is accepted (line 28).

```
public function isPointInsidePolygon(point:Point, polygon:int) : Boolean {
  var edges:Array = getedges(polygon);
  var beforeList:Array = new Array(), afterList:Array = new Array();

  for(var i:int = 0; i < edges.length; i++) {
    if((vertexY[edgeStartVertex[edges[i]]] <= point.y && vertexY[edgeEndVertex[
        edges[i]]] >= point.y) ||
      (vertexY[edgeStartVertex[edges[i]]] >= point.y && vertexY[edgeEndVertex[
          edges[i]]] <= point.y)) {
      if(vertexX[edgeStartVertex[edges[i]]] < point.x && vertexX[edgeEndVertex[
          edges[i]]] < point.x) {
        beforeList[beforeList.length] = edges[i];
      } else if(vertexX[edgeStartVertex[edges[i]]] > point.x && vertexX[
          edgeEndVertex[edges[i]]] > point.x) {
        afterList[afterList.length] = edges[i];
      } else {
        var s:Number = (point.y - vertexY[edgeStartVertex[edges[i]]]) / (vertexY[
            edgeEndVertex[edges[i]]] - vertexY[edgeStartVertex[edges[i]]]);
        var x:Number = vertexX[edgeStartVertex[edges[i]]] + s * (vertexX[
            edgeEndVertex[edges[i]]] - vertexX[edgeStartVertex[edges[i]]]);

        if(x > point.x) {
          afterList[afterList.length] = edges[i];
        } else if(x < point.x) {
          beforeList[beforeList.length] = edges[i];
        } else {
          return false;
        }
      }
    }
  }

  if((beforeList.length % 2 == 1) && (afterList.length % 2 == 1)) {
    return true;
  } else {
    return false;
  }
}
```

Listing 8.4: Determining whether a point is inside a polygon or not.

	1	2	3	4	5	Before	After	Accept
A	-	After	-	Before	-	1	1	Yes
B	After	-	-	-	After	0	2	No
C	-	After	-	-	Before	1	1	Yes
D	-	-	-	-	-	0	0	No
E	-	-	After	After	-	0	2	No
F	After	-	-	-	Before	1	1	Yes
G	Before	-	-	-	Before	0	2	No
H	-	After	-	Equal	-	0	1	No

Table 8.2: The computation results according to Figure 8.5.

8.5.3 Inserting an Element

Since the Voronoi Diagram is built up using the incremental method, the generators are added to the diagram (parent polygon) one by one. This is implemented in the `boundaryGrowing` function of the WingedEdgeStructure class located in the `WingedEdgeStructure.as` file. The `boundaryGrowing` implementation differentiates three different situations, based on how many generators are already present in the parent polygon. The first time the function is called, no other generators are present. Therefore, the generator is placed at its randomly generated point inside the parent polygon and all of the parents' borders are assigned to the generator.

The second situation appears when adding the second generator. In this case the area of the parent polygon is split between the first and the second generator. The new generator is added at its generated position. Afterwards, the bisector point between the two points is calculated based on the formula:

$$x_{ij} = \frac{\| x_j \|^2 - \| x_i \|^2 + w_i - w_j}{2 \cdot \| x_j - x_i \|^2} \cdot (x_j - x_i) \tag{8.36}$$

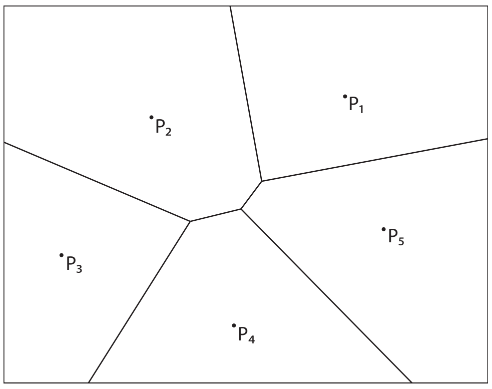
Given the bisector point, a perpendicular line between the two generators running through the bisector point is drawn. This line is the bisector edge separating the two generators areas.

The third situation emerges when adding the third generator, as well as every time a generator is added later. This situation also illustrated in Figure 8.6. The generator is added at the generated position and the distances to the previously placed generators are measured. The generator closest to the new generator (further referred to as P_1) is chosen to be handled first. The bisector point between the new generator and P_1 is computed according to Formula (8.36). Then the perpendicular bisector line between the two generators is drawn to separate them.

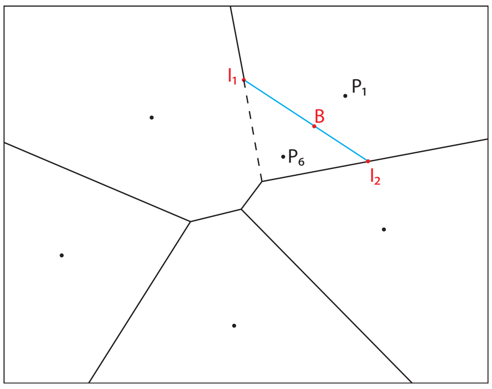
Afterwards, a boundary growing process is performed for the new generator as follows. The two intersection points of the newly drawn bisector edge and the edges surrounding P_1 are examined beginning with the intersection point at the end of the line. If the intersected edge is no boundary edge it separates two regions with one of them being the region of P_1. The generator of the other region (P_2) is handled next.

This leads to the creation of a perpendicular bisector edge between the new added generator and P_2. By looking at the intersection points of the bisector edge and the edges surrounding P_2 the next generator to consider is eventually found (P_3). This is done until either the first bisector edge is reached again, or no further generator to be handled can be found.

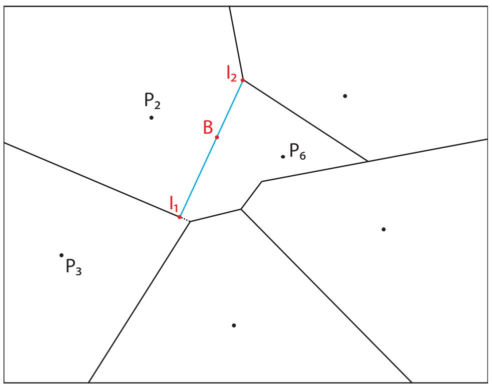
If no further generator is found because a boundary edge of the parent polygon is intersected, the boundary growing process is carried out starting from the intersection point at the start vertex of the first created bisector edge.



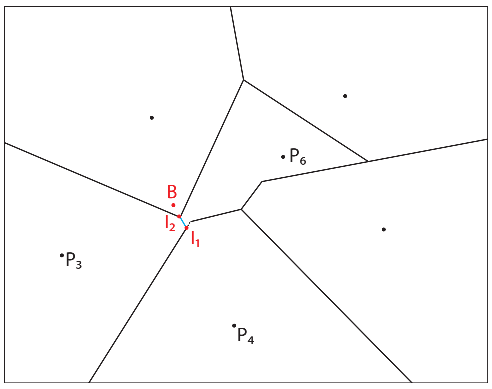
(a) The initial Voronoi diagram hosts 5 generators $\{P_1, .., P_5\}$.



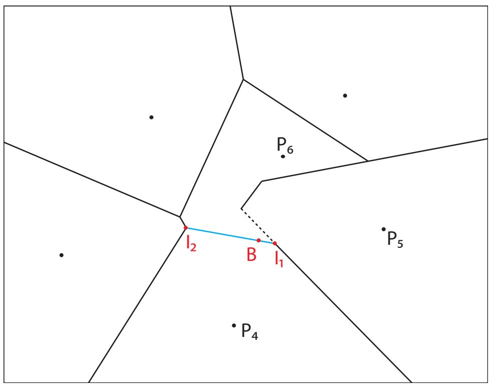
(b) A new generator P_6 is added to the diagram (closest to P_1). A perpendicular bisector line between P_6 an P_1 is drawn (blue line). The neighbouring polygon of the intersected line at I_1 is handled next.



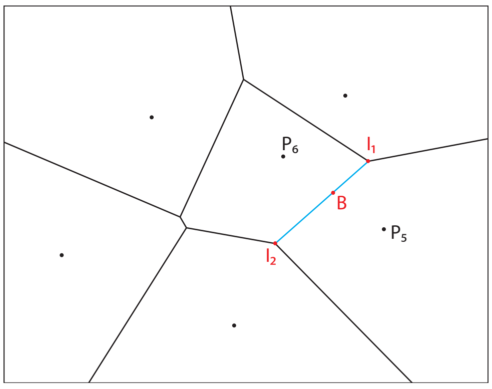
(c) A bisector edge between P_6 and P_2 is created. The intersection point I_1 is examined leading to generator P_3 being handled next.



(d) The bisector edge between P_6 and P_3 is created and boundary growing will be carried out for P_4 next (the not handled neighbour polygon of I_1).



(e) The bisector edge between P_6 and P_4 is drawn. P_5 will be handled next.



(f) After drawing the bisector edge between P_6 and P_5 the first created bisector edge is reached. This stops the boundary growing process and thus the insertion of P_6 is finished.

Figure 8.6: The boundary growing process carried out after adding a new generator to a voronoi diagram which already hosts at least two other generators.

Chapter 9

Outlook

Information visualisation will continue to extend its popularity, because the amount of data and its complexity is growing ever larger and with it the need for solutions to explore and understand this data. Online visualisation solutions are becoming easier and more comfortable to handle, and are offering more features and social experiences to attract the attention of ordinary internet users.

Although the Liquid Diagrams framework and its gadgets offer many possibilities to create aesthetically pleasing and highly interactive visualisations, there are several possibilities to further enhance it:

- **Web Site**
  The most relevant improvement would be to create a web site to host both, the gadgets and user data. This would remove the dependency on GoogleDocs and ensure long-time operability. Besides that it will eliminate all restrictions and limitations given by the GoogleDocs interface. Users could also share and collaborate on visualisations, in a manner similar to Many Eyes [IBM, 2010b].

- **Time Control Element**
  Another major improvement would be to add a control element to each gadget to enable interactive exploration of changes in the data set over a period of time (similar to the time control available in the Gapminder visualisation - see Section 4.3.9).

- **Container Around Visualisation**
  By adding an additional container which hosts the drawn entities new features like zooming, panning, scrolling, and rotating the content would be available.

- **Individual Entity Setup**
  This improvement would include individual treatment for each entity through an Options Dialog or when right clicking an entity. For example, it might be possible to rename the entity, change its size, colour, appearance, label, and many more options.

- **Adobe AIR Support**
  By implementing the gadgets using Adobe Air, additional features for offline usage would become available. This includes support for data bases and better client file access possibilities.

- **New Layout Algorithms**
  A possibility to upgrade the treemap and voronoi treemap visualisations would be to implement additional layout algorithms. This would include the strip and ordered layout algorithm for the treemap and the additively weighted voronoi tessellation for the voronoi treemap visualisation.

- **Improved Selection Behaviour**
  The implementation of improved entity selection methods, like angular brushing for parallel coordinates introduced in Hauser et al. [2002].

Appendix A

User Guide

To use the Liquid Diagrams visualisations as gadgets in combination with Google Spreadsheets, several steps have to be performed.

A.1 Gadget Installation

To install Liquid Diagrams for use with Google Spreadsheets, several files have to be partially modified and copied to a reachable web space. These files include common files used by every visualisation as well as gadget-specific files. The following steps have to be performed to install Liquid Diagrams:

1. Select some web space which is reachable without access restrictions.

   For example: http://www.lessacher.info

2. Create a parent folder in this web space which will hold all visualisation gadgets.

   For example: http://www.lessacher.info/ivis/

3. In this parent folder, create a folder called `common`.

   For example: http://www.lessacher.info/ivis/common/

4. In the file `includes.js` located in the `common` folder, the host string (located in line 2) needs to be changed to the parent folder of the chosen web space. The file is then closed and saved.

   For example: var host = 'http://www.lessacher.info/ivis/';

5. The files `utils.js`, `includes.js`, and `colour-schemes.js` are simply copied to the common folder on the web space.

   For example: http://www.lessacher.info/ivis/common/utils.js

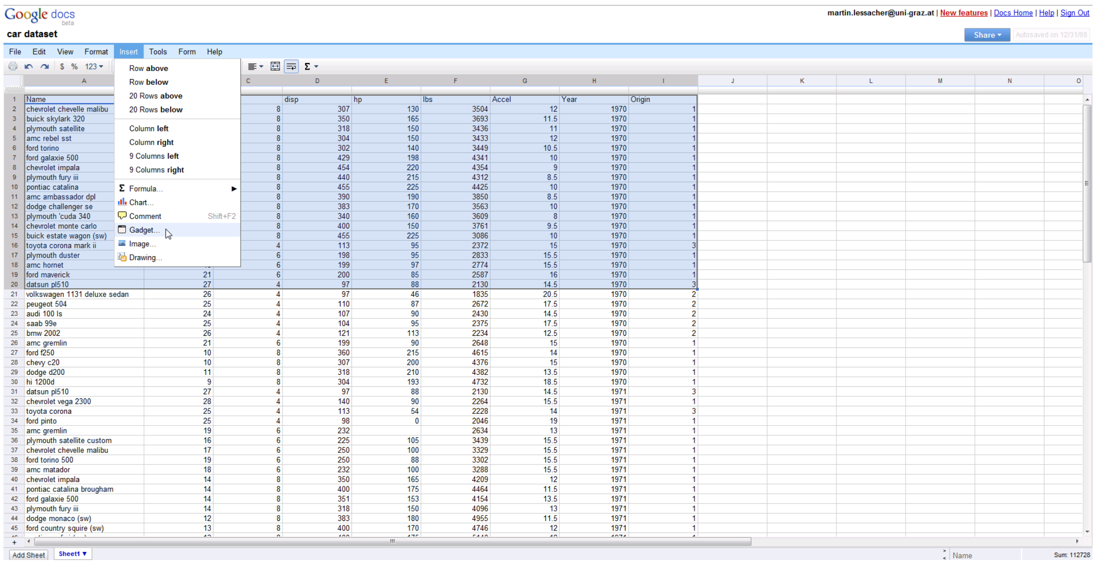
6. For each visualisation a folder is created in parent directory on the web space.

   For example: http://www.lessacher.info/ivis/linechart/

7. A replacement needs to be done in the local XML files of each visualisation. In the XML file, the lines need to be found which include the `includes.js` file. These lines need to be changed to contain the correct web space.

   For example: Find http://www.host.com/parent/common/includes.js and replace it with

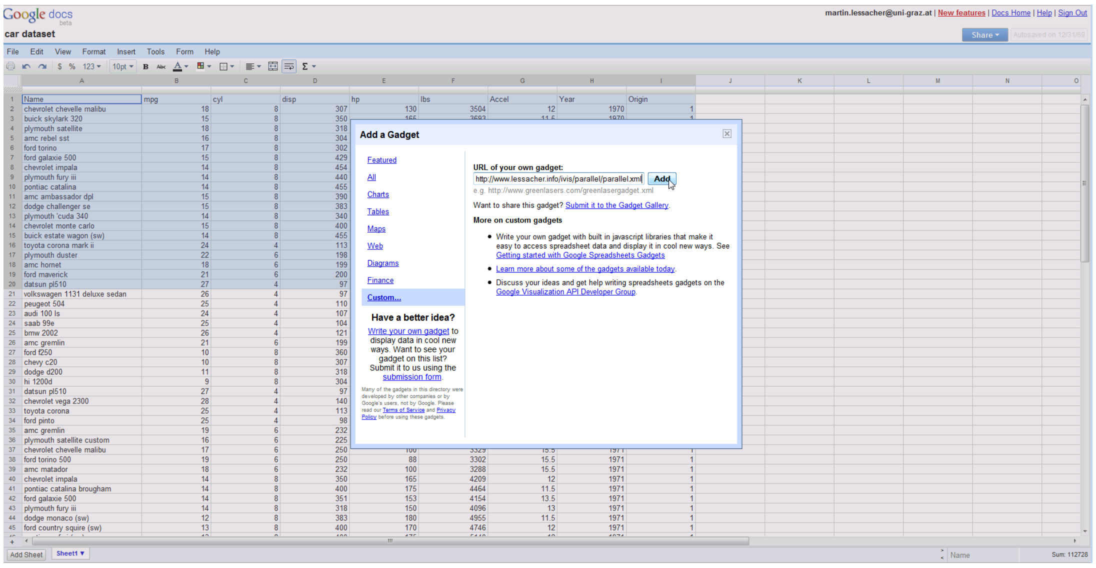
   http://www.lessacher.info/ivis/common/includes.js.



car dataset

	Name	mpg	cyl	disp	hp	lbs	Accel	Year	Origin
2	chevrolet chevelle malibu	18	8	307	130	3504	12	1970	1
3	buick skylark 320	15	8	350	165	3693	11.5	1970	1
4	plymouth satellite	18	8	318	150	3436	11	1970	1
5	amc rebel sst	16	8	304	150	3433	12	1970	1
6	ford torino	17	8	302	140	3449	10.5	1970	1
7	ford galaxie 500	15	8	429	198	4341	10	1970	1
8	chevrolet impala	14	8	454	220	4354	9	1970	1
9	plymouth fury iii	14	8	440	215	4312	8.5	1970	1
10	pontiac catalina	14	8	455	225	4425	10	1970	1
11	amc ambassador dpl	15	8	390	190	3850	8.5	1970	1
12	dodge challenger se	15	8	383	170	3563	10	1970	1
13	plymouth 'cuda 340	14	8	340	160	3609	8	1970	1
14	chevrolet monte carlo	15	8	400	150	3761	9.5	1970	1
15	buick estate wagon (sw)	14	8	455	225	3086	10	1970	1
16	toyota corona mark ii	24	4	113	95	2372	15	1970	3
17	plymouth duster	22	6	198	95	2833	15.5	1970	1
18	amc hornet	18	6	199	97	2774	15.5	1970	1
19	ford maverick	21	6	200	85	2587	16	1970	1
20	datsun pl510	27	4	97	88	2130	14.5	1970	3
21	volkswagen 1131 deluxe sedan	26	4	97	46	1835	20.5	1970	2
22	peugeot 504	25	4	110	87	2672	17.5	1970	2
23	audi 100 ls	24	4	107	90	2430	14.5	1970	2
24	saab 99e	25	4	104	95	2375	17.5	1970	2
25	bmw 2002	26	4	121	113	2234	12.5	1970	2
26	amc gremlin	21	6	199	90	2648	15	1970	1
27	ford f250	10	8	360	215	4615	14	1970	1
28	chevy c20	10	8	307	200	4376	15	1970	1
29	dodge d200	11	8	318	210	4382	13.5	1970	1
30	hi 1200d	9	8	304	193	4732	18.5	1970	1
31	datsun pl510	27	4	97	88	2130	14.5	1971	3
32	chevrolet vega 2300	28	4	140	90	2264	15.5	1971	1
33	toyota corona	25	4	113	54	2228	14	1971	3
34	ford pinto	25	4	98	0	2046	19	1971	1
35	amc gremlin	19	6	232		2634	13	1971	1
36	plymouth satellite custom	16	6	225	105	3439	15.5	1971	1
37	chevrolet chevelle malibu	17	6	250	100	3329	15.5	1971	1
38	ford torino 500	19	6	250	88	3302	15.5	1971	1
39	amc matador	18	6	232	100	3288	15.5	1971	1
40	chevrolet impala	14	8	350	165	4209	12	1971	1
41	pontiac catalina brougham	14	8	400	175	4464	11.5	1971	1
42	ford galaxie 500	14	8	351	153	4154	13.5	1971	1
43	plymouth fury iii	14	8	318	150	4096	13	1971	1
44	dodge monaco (sw)	12	8	383	180	4955	11.5	1971	1
45	ford country squire (sw)	13	8	400	170	4746	12	1971	1

(a)



Add a Gadget

URL of your own gadget:
http://www.lessacher.info/ivis/parallel/parallel.xml
e.g. http://www.greenlasers.com/greenlasergadget.xml

Want to share this gadget? Submit it to the Gadget Gallery.

More on custom gadgets

- Write your own gadget with built in javascript libraries that make it easy to access spreadsheet data and display it in cool new ways. See Getting started with Google Spreadsheets Gadgets
- Learn more about some of the gadgets available today.
- Discuss your ideas and get help writing spreadsheets gadgets on the Google Visualization API Developer Group.

Have a better idea?

Write your own gadget to display data in cool new ways. Want to see your gadget on this list? Submit it to us using the submission form.

Many of the gadgets in this directory were developed by other companies or by Google's users, not by Google. Please read our Terms of Service and Privacy Policy before using these gadgets.

(b)

Figure A.1: After having chosen to insert a gadget, the URL of the gadget XML file is entered.

Visualisation	Chart Data Format	Pie Data Format	Multidimensional Data Format	Hierarchical Data Format	Geo Data Format
Line Chart	X				
Bar Chart	X				
Area Chart	X				
Pie Chart		X			
Parallel Coordinates			X		
Starplot			X		
Treemap				X	
Voronoi Treemap				X	
Heatmap					X

Table A.1: The data format expected by each visualisation gadget.

8. A further replacement of the web space path needs to be done in the visualisation specific JavaScript file. *For example: Find http://www.host.info/parent/linechart/linechart.js*

   and replace it with http://www.lessacher.info/ivis/linechart/linechart.js

A.2 Insertion

After installation, Liquid Diagrams can be used to visualise the data contained in a Google Spreadsheet. A visualisation is inserted as a gadget into a Google Spreadsheet. The data to be visualised is first selected in the spreadsheet. Afterwards, `Gadget...` is selected from the `Insert` menu. In the pop up window, `Custom...` is chosen and the URL of the desired visualisation's XML file (for example http://www.lessacher.info/ivis/linechart/linechart.xml) is entered. The insert process is illustrated in Figure A.1.

After clicking the `Add` button, a window pops up showing the settings defined in the XML file. These are the initial settings for the visualisation. After clicking the `Apply` button at the bottom of the settings window, the visualisation is drawn into the gadget container, as shown in Figure A.2.

A.3 Data Formats

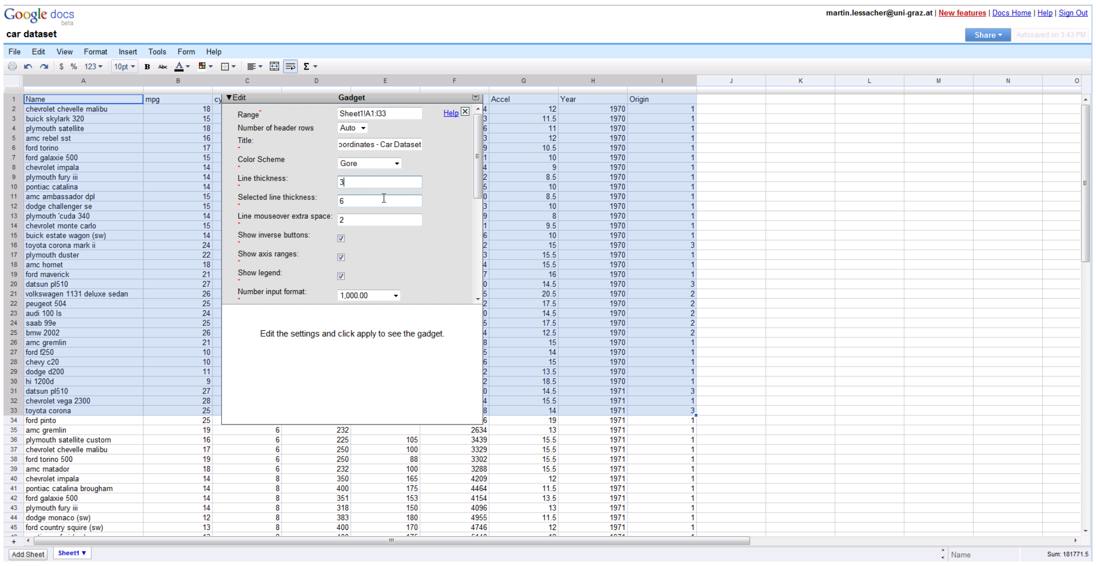
In order to visualise the data properly, the data in the Google Spreadsheet needs to be entered according to a special data format for each visualisation gadget. Liquid Diagrams distinguishes between five different data formats. Table A.1 gives a brief summarises data format each visualisation uses.

A.3.1 Chart Data Format

This data format is the standard data format used by the basic visualisation gadgets (line chart, bar chart, and area chart). Table A.2 illustrates an example for this data format. The first line contains the names of the entities while the first column features the values to be shown on the x-axis (e.g. time). This leads to columns representing the entities while the lines hold the entities data of specific x-axis ticks.

A.3.2 Pie Data Format

The pie data format is used in the pie chart visualisation and is illustrated in Table A.3. The first column features the names of the different entities and the second column hosts the values of these entities.


Google docs beta

car dataset

martin.lessacher@uni-graz.at | New features | Docs Home | Help | Sign Out

	Name	mpg	cy				Accel	Year	Origin
2	chevrolet chevelle malibu	18				4	12	1970	1
3	buick skylark 320	15				3	11.5	1970	1
4	plymouth satellite	18				6	11	1970	1
5	amc rebel sst	16				3	12	1970	1
6	ford torino	17				9	10.5	1970	1
7	ford galaxie 500	15				1	10	1970	1
8	chevrolet impala	14				4	9	1970	1
9	plymouth fury iii	14				2	8.5	1970	1
10	pontiac catalina	14				5	10	1970	1
11	amc ambassador dpl	15				0	8.5	1970	1
12	dodge challenger se	15				3	10	1970	1
13	plymouth 'cuda 340	14				9	8	1970	1
14	chevrolet monte carlo	15				1	9.5	1970	1
15	buick estate wagon (sw)	14				6	10	1970	1
16	toyota corona mark ii	24				2	15	1970	3
17	plymouth duster	22				3	15.5	1970	1
18	amc hornet	18				4	15.5	1970	1
19	ford maverick	21				7	16	1970	1
20	datsun pl510	27				0	14.5	1970	3
21	volkswagen 1131 deluxe sedan	26				5	20.5	1970	2
22	peugeot 504	25				2	17.5	1970	2
23	audi 100 ls	24				0	14.5	1970	2
24	saab 99e	25				5	17.5	1970	2
25	bmw 2002	26				4	12.5	1970	2
26	amc gremlin	21				8	15	1970	1
27	ford f250	10				5	14	1970	1
28	chevy c20	10				6	15	1970	1
29	dodge d200	11				2	13.5	1970	1
30	hi 1200d	9				2	18.5	1970	1
31	datsun pl510	27				0	14.5	1971	3
32	chevrolet vega 2300	28				4	15.5	1971	1
33	toyota corona	25				8	14	1971	3
34	ford pinto	25				6	19	1971	1
35	amc gremlin	19	6	232		2634	13	1971	1
36	plymouth satellite custom	16	6	225	105	3439	15.5	1971	1
37	chevrolet chevelle malibu	17	6	250	100	3329	15.5	1971	1
38	ford torino 500	19	6	250	88	3302	15.5	1971	1
39	amc matador	18	6	232	100	3288	15.5	1971	1
40	chevrolet impala	14	8	350	165	4209	12	1971	1
41	pontiac catalina brougham	14	8	400	175	4464	11.5	1971	1
42	ford galaxie 500	14	8	351	153	4154	13.5	1971	1
43	plymouth fury iii	14	8	318	150	4096	13	1971	1
44	dodge monaco (sw)	12	8	383	180	4955	11.5	1971	1
45	ford country squire (sw)	13	8	400	170	4746	12	1971	1

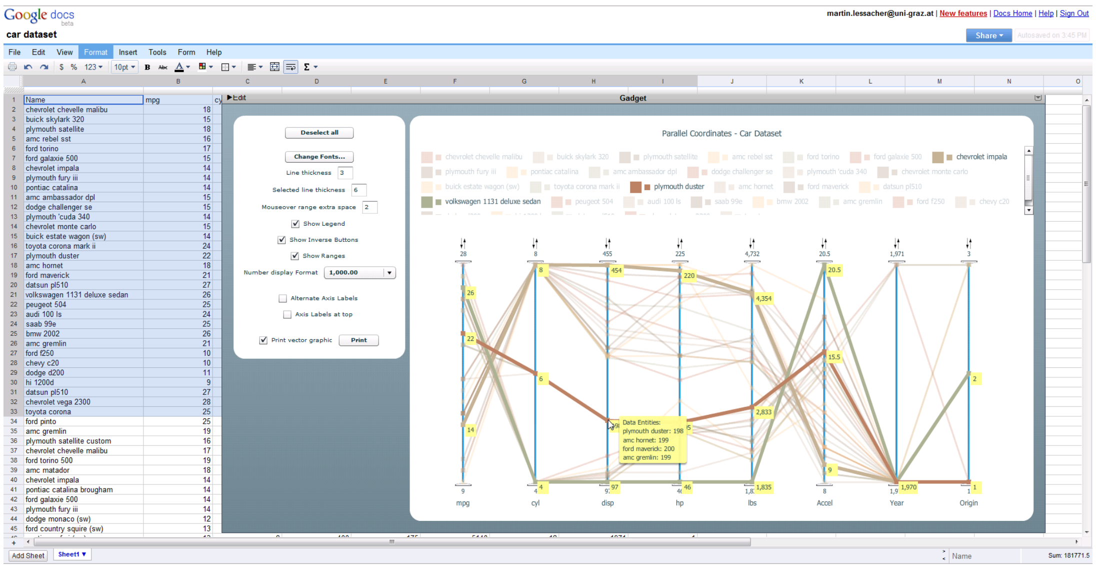
▼Edit Gadget

Range	Sheet1!A1:I33
Number of header rows	Auto
Title:	oordinates - Car Dataset
Color Scheme	Gore
Line thickness:	3
Selected line thickness:	6
Line mouseover extra space:	2
Show inverse buttons:	✓
Show axis ranges:	✓
Show legend:	✓
Number input format:	1,000.00

Edit the settings and click apply to see the gadget.

Sum: 181771.5

(a)


Google docs beta

car dataset

martin.lessacher@uni-graz.at | New features | Docs Home | Help | Sign Out

	Name	mpg
2	chevrolet chevelle malibu	18
3	buick skylark 320	15
4	plymouth satellite	18
5	amc rebel sst	16
6	ford torino	17
7	ford galaxie 500	15
8	chevrolet impala	14
9	plymouth fury iii	14
10	pontiac catalina	14
11	amc ambassador dpl	15
12	dodge challenger se	15
13	plymouth 'cuda 340	14
14	chevrolet monte carlo	15
15	buick estate wagon (sw)	14
16	toyota corona mark ii	24
17	plymouth duster	22
18	amc hornet	18
19	ford maverick	21
20	datsun pl510	27
21	volkswagen 1131 deluxe sedan	26
22	peugeot 504	25
23	audi 100 ls	24
24	saab 99e	25
25	bmw 2002	26
26	amc gremlin	21
27	ford f250	10
28	chevy c20	10
29	dodge d200	11
30	hi 1200d	9
31	datsun pl510	27
32	chevrolet vega 2300	28
33	toyota corona	25
34	ford pinto	25
35	amc gremlin	19
36	plymouth satellite custom	16
37	chevrolet chevelle malibu	17
38	ford torino 500	19
39	amc matador	18
40	chevrolet impala	14
41	pontiac catalina brougham	14
42	ford galaxie 500	14
43	plymouth fury iii	14
44	dodge monaco (sw)	12
45	ford country squire (sw)	13

▶Edit Gadget

Line thickness	3
Selected line thickness	6
Mouseover range extra space	2
Show Legend	✓
Show Inverse Buttons	✓
Show Ranges	✓
Number display Format	1,000.00
Alternate Axis Labels	☐
Axis Labels at top	☐
Print vector graphic	✓

Parallel Coordinates - Car Dataset

Data Entities:
plymouth duster: 198
amc hornet: 199
ford maverick: 200
amc gremlin: 199

Sum: 181771.5

(b)

Figure A.2: Before the visualisation is drawn, the Google Gadget parameters need to be set. After clicking Apply, the visualisation is drawn.

Month	Internet Explorer 8	Internet Explorer 7	Mozilla Firefox	Safari	Opera
February	14.7	11	46.5	3.8	2.1
March	15.3	9.1	46.2	3.7	2.2
April	16.2	9.3	46.4	3.7	2.2
May	16	9.1	46.6	3.5	2.2

Table A.2: The chart data format, used by line chart, bar chart, and area chart. An export of a line chart using this data can be seen in Figure 2.3c.

Browser	February
Internet Explorer	25.7
Mozilla Firefox	46.5
Safari	3.8
Opera	2.1

Table A.3: The pie data format, used in the pie chart visualisation.

Car	MPG	Cyl	HP
Dodge Monaco	12	8	383
Toyota Corolla 1200	31	4	71
Peugeot 304	30	4	79
Fiat 124b	30	4	88

Table A.4: The multidimensional data format is used in the parallel coordinates and star plot visualisations.

A.3.3 Multidimensional Data Format

The multidimensional data format is used in the parallel coordinates and the star plot visualisations. An example can be seen in Table A.4. The first row contains the data dimensions (attributes, axis names) and the first column features the entity names. Generally, rows represent entities and columns represent attributes.

A.3.4 Hierarchical Data Format

The treemap visualisation and the Voronoi treemap visualisation expect hierarchically structured data, which has to be entered according to this data format. Table A.5 shows an example for this data format. Each row, except the first, represents a single data entity. The first row is the header row and has to be according to a special format in order to recognise optional data fields. The first column represents the name of the entity, no matter what the label of the column is.

The second column is always used to specify the parent of this entity. If the parent column of an entity is left blank, the entity is used as one of the root (top level) nodes. If the value entered into the parent field of an entity does not match any other entity in the data set, an entity having the name of this value is created as a top level node and the entity is assigned to it as a child node.

Beginning with the third column, all further columns are considered value columns which can be chosen in the gadgets combo boxes as colour or size defining properties. The label of the column in the header line is used as the name displayed in the combo boxes.

A.3.5 Geo Data Format

This data format is used in the heatmap (Choropleth) visualisation. As in the hierarchical data format, each row except the first represents a single entity. Due to optional data fields there are two possibilities to enter entities in a correct way. The first possibility is to define a single entity by two data fields. In this case the first column represents the ISO-3166 ALPHA-2 code (see Section 7.8) of the parent country and the second column is used to specify the entity using the ISO-3166 ALPHA-2 code for the region. In this case the label of the second column has to be `Name` in order to interpret the format in a correct way (see Table A.6).

Distinct	Province	Area	Population
Graz	Styria	12,748	253,994
Spittal	Carinthia	276,408	79,759
Villach	Carinthia	13,490	58,949

Table A.5: The hierarchical data format is used in the treemap visualisation and the Voronoi treemap visualisation.

Country	Name	Population	Area
DE	BE	3,416,000	892
DE	BR	2,535,000	29,479
DE	BY	12,520,000	70,552

Table A.6: In this geo data format each entity is identified by the first two columns.

The second possibility is to define an entity by using only the first column. The first column must contain the ISO-3166 ALPHA-2 country code or an unique country name present in the XML lookup table or if the entity represents a province of a country the ISO-3166 ALPHA-2 code of the country followed by a dash (-) and the ISO-3166 ALPHA-2 code of the province (see Table A.7).

Depending on the optional field all columns starting at the third, or forth column are data columns. These columns are used as the colour defining attribute in the heatmap visualisation.

Country	Population	Internet Users
Austria	8,214,160	5,143,600
Switzerland	7,623,438	5,739,300
DE-BE	3,416,000	892
DE-BY	12,520,000	70,552

Table A.7: In this geo data format each entity is identified by the first column only. Country names, specified in the lookup table, as well as composite country codes are used.

Bibliography

Adobe [2010a]. *Actionscript 3.0 Reference*. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/mx/graphics/codec/PNGEncoder.html. (Cited on page 62.)

Adobe [2010b]. *Adobe AIR - Deliver Rich Internet Applications on the Desktop*. http://wwwimages.adobe.com/www.adobe.com/products/air/pdfs/air_flex_datasheet.pdf. (Cited on pages 17 and 18.)

Adobe [2010c]. *Adobe Flex*. http://www.adobe.com/de/products/flex/. (Cited on page 35.)

Adobe [2010d]. *Adobe Illustrator*. http://www.adobe.com/products/illustrator/. (Cited on page 19.)

Adobe [2010e]. *Adobe SVG Viewer*. http://www.adobe.com/svg/viewer/install/. (Cited on page 21.)

Adobe [2010f]. *Flash Player Statistics*. http://www.adobe.com/products/player_census/flashplayer/. (Cited on pages 35 and 37.)

Adobe [2010g]. *SWF searchability FAQ*. http://www.adobe.com/devnet/flashplayer/articles/swf_searchability.html. (Cited on page 18.)

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 pages 13 and 88.)

AlwaysOn Technologies [2010]. *Cloud Browse*. http://www.alwaysontechnologies.com/. (Cited on page 35.)

Andrews, Keith [2006]. *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 ix.)

Andrews, Keith [2009]. *Information Visualisation, Course Notes*. http://courses.iicm.tugraz.at/ivis/ivis.pdf. (Cited on pages 4 and 5.)

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(3/4), pages 166–181. doi:10.1057/palgrave.ivs.9500023. (Cited on pages 13 and 90.)

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

Aurenhammer, Franz and Rolf Klein [1999]. *Voronoi Diagrams*. In Sack, Jörg-Rüdiger and Jorge Urrutia (Editors), *Handbook of Computational Geometry*, pages 201–290. Elsevier. ISBN 0444825371. http://www.pi6.fernuni-hagen.de/publ/tr198.pdf. (Cited on page 90.)

Balzer, Michael and Oliver Deussen [2005]. *Voronoi Treemaps*. In *Proc. 2005 IEEE Symposium on Information Visualization (InfoVis 2005)*, page 7. IEEE Computer Society. ISBN 078039464x. doi:10.1109/INFOVIS.2005.40. http://www-hagen.informatik.uni-kl.de/~kerren/courses/lecture/ws06/infovis/papers/VoronoiTreemapInfoVis2005.pdf. (Cited on pages 13, 90 and 92.)

Balzer, Michael, Oliver Deussen, and Claus Lewerentz [2005]. *Voronoi Treemaps for the Visualization of Software Metrics*. In *Proc. 2005 ACM symposium on Software Visualization (SoftVis '05)*, pages 165–172. ACM. ISBN 1595930736. doi:10.1145/1056018.1056041. http://www.cse.unt.edu/~tarau/wwwroot/tarau/teaching/OLD_COURSES/SoftEng/PapersToRead/p165-balzer.pdf. (Cited on page 13.)

Bauer, Alois, Sophie Steinparz, Richard Aßmair, and John Feiner [2009]. *Parallel Coordinates - A Tutorial*. Graz University of Technology. Project Report. (Cited on page 43.)

Baumgart, Bruce Guenther [1975]. *A Polyhedron Representation for Computer Vision*. In *Proc. 1975 National Computer Conference and Exposition (AFIPS '75)*, pages 589–596. ACM. doi:10.1145/1499949.1500071. (Cited on pages 90 and 108.)

Bederson, Benjamin B., Ben Shneiderman, and Martin Wattenberg [2002]. *Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies*. *ACM Transactions on Graphics*, 21(4), pages 833–854. doi:10.1145/571647.571649. http://hcil.cs.umd.edu/trs/2001-18/2001-18.pdf. (Cited on page 79.)

Bertin, Jacques [1981]. *Graphics and Graphic Information-Processing*. Walter de Gruyter. ISBN 3110069016. (Cited on page 3.)

Boy, Oriya and Raja Ranjan Senapati [2010]. *Applet 2 Application*. http://sourceforge.net/projects/applet2app/. (Cited on page 16.)

Brinton, Willard Cope [1914]. *Graphic Methods for Presenting Facts*. The Engineering Magazine Company. ISBN 9781432526337. http://www.archive.org/download/graphicmethodsfo00brinrich/graphicmethodsfo00brinrich.pdf. (Cited on page 3.)

Brown, Millward [2010]. *Methodology for Adobe Plug-In Technology Study*. http://www.adobe.com/products/player_census/methodology/. (Cited on page 37.)

Bruls, Mark, Kees Huizing, and Jarke van Wijk [1999]. *Squarified Treemaps*. In *Proc. 1999 Joint Eurographics and IEEE TCVG Symposium on Visualization (VisSym '99)*, pages 33–42. Press. http://www.win.tue.nl/~vanwijk/stm.pdf. (Cited on pages 79 and 80.)

Card, Stuart K., Jock D. Mackinlay, and Ben Shneiderman [1999]. *Using Vision to Think*. In *Readings in Information Visualization: Using Vision to Think*, pages 579–581. Morgan Kaufmann. ISBN 1558605339. (Cited on pages 3, 4 and 5.)

Chi, Ed Huai-Hsin [1999]. *A Framework for Information Visualization Spreadsheets*. PhD Thesis, University of Minnesota. http://www-users.cs.umn.edu/~echi/phd/chi-thesis.pdf. (Cited on page 4.)

Coenraets, Christophe [2003]. *An Overview of MXML: The Flex Markup Language*. http://www.adobe.com/devnet/flex/articles/paradigm.html. (Cited on page 38.)

COLOURlovers [2010]. *COLOURlovers*. http://colourlovers.com. (Cited on page 50.)

Corel [2010]. *Corel Draw Graphics Suite*. http://www.corel.com/servlet/Satellite/de/de/Product/1191272117978#tabview=tab0. (Cited on page 19.)

Davis, Michele and Jon Phillips [2008]. *Flex 3: A Beginners Guide*. McGraw-Hill Osborne Media. ISBN 0071544186. (Cited on pages 16 and 35.)

Descartes, René [1644]. *Le Monde de Mr Descartes, ou Le Trait de la Lumière*. (Cited on page 13.)

d'Ocagne, Maurice [1885]. *Coordonnées parallèles et axiales: Méthode de transformation géométrique et procédé nouveau de calcul graphique déduits de la considération des coordonnées parallèlles*. Cornell University Library. ISBN 1429700971. (Cited on page 8.)

Dupin, Charles [1826]. *Carte figurative de l'instruction populaire de la France*. (Cited on pages 11 and 12.)

Finley, Darel Rex [2010]. *Point-In-Polygon Algorithm - Determining Whether A Point Is Inside A Complex Polygon*. http://alienryderflex.com/polygon/. (Cited on page 109.)

Friendly, Michael [2008]. *Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization*. http://www.math.yorku.ca/SCS/Gallery/milestone/milestone.pdf. (Cited on page 8.)

Friendly, Michael and Daniel J. Denis [2010]. *Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization*. http://www.math.yorku.ca/SCS/Gallery/milestone/milestone.pdf. (Cited on page 12.)

Friendly, Michael and Leland Wilkinson [2009]. *The History of the Cluster Heat Map*. In *The American Statistician*, *2*, volume 63, pages 179–184. doi:10.1198/tas.2009.0033. (Cited on page 11.)

Gapminder [2010a]. *Data in Gapminder World*. http://www.gapminder.org/data/. (Cited on page 31.)

Gapminder [2010b]. *The Gapminder Foundation*. http://www.gapminder.org/. (Cited on page 31.)

GG [2010]. *NameVoyager: Baby Names Wizard*. http://www.babynamewizard.com/voyager/. Generation Grownup. (Cited on page 6.)

Google [2010a]. *Gadgets API*. http://code.google.com/intl/de-DE/apis/gadgets/. (Cited on page 29.)

Google [2010b]. *Google Chart API*. http://code.google.com/intl/de-DE/apis/chart/. (Cited on pages 26 and 27.)

Google [2010c]. *Google Chart API FAQ*. http://code.google.com/intl/de-DE/apis/chart/faq.html. (Cited on page 26.)

Google [2010d]. *Google Docs*. http://docs.google.com/. (Cited on page 28.)

Google [2010e]. *Google Visualization API*. http://code.google.com/intl/de-DE/apis/visualization/. (Cited on page 29.)

Google [2010f]. *Visualisation Playground*. http://code.google.com/apis/ajax/playground/. (Cited on page 30.)

Guerry, André-Michel and Adriano Balbi [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. (Cited on page 12.)

Hauser, Helwig, Florian Ledermann, and Helmut Doleisch [2002]. *Angular Brushing of Extended Parallel Coordinates*. In *proc. 2002 IEEE Symposium on Information Visualization (InfoVis 2002)*, page 127. IEEE Computer Society. ISBN 076951751X. doi:10.1109/INFVIS.2002.1173157. `http://www.mediavirus.org/parvis/parvis_full.pdf`. (Cited on page 116.)

Huckaby, Tim [2010]. *Silverlight Fully Trusted Out of Browser*. `http://www.windowsitpro.com/article/silverlight-development/Silverlight-Fully-Trusted-Out-of-Browser.aspx`. (Cited on page 19.)

IBM [2010a]. *Fernanda B. Viégas*. `http://www.research.ibm.com/visual/fernanda.html`. (Cited on page 25.)

IBM [2010b]. *Many Eyes*. `http://manyeyes.alphaworks.ibm.com/manyeyes/page/About.html`. (Cited on pages 24 and 115.)

iCharts [2010a]. *iCharts blog*. `http://www.icharts.net`. (Cited on page 30.)

iCharts [2010b]. *iCharts Business*. `http://ichartsbusiness.com`. (Cited on page 30.)

Inkscape [2010]. *Inkscape*. `http://www.inkscape.org`. (Cited on page 19.)

Inselberg, Alfred [1985]. *The plane with parallel coordinates*. *The Visual Computer*, 1(2), pages 69–91. doi:10.1007/BF01898350. (Cited on page 8.)

Inselberg, Alfred [2009]. *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications*. Springer. ISBN 0387215077. (Cited on page 8.)

ISO [1998]. *ISO 3166-2:1998, Codes for the representation of names of countries and their subdivisions – Part 2: Country subdivision code*. ANSI, 118 pages. (Cited on page 84.)

ISO [2010]. *English Country Names and Code Elements*. `http://www.iso.org/iso/english_country_names_and_code_elements`. (Cited on pages 84 and 104.)

Jern, Mikael, Tobias Aström, and Sara Johansson [2008]. *GeoAnalytics Tools Applied to Large Geospatial Datasets*. In *Proc. 2008 International Conference on Information Visualisation (InfoVis 2008)*, pages 362–372. IEEE Computer Society. ISBN 9780769532684. doi:10.1109/IV.2008.27. (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. 1991 IEEE Conference on Visualization (Vis '91)*, pages 284 – 291. ISBN 0818622458. doi:10.1109/VISUAL.1991.175815. `http://drum.lib.umd.edu/bitstream/1903/370/2/CS-TR-2657.pdf`. (Cited on pages 78, 79 and 80.)

Knapitsch, Ferdinand, Robert Lanner, and Michael Kober [2009]. *Geoheatmap Google Gadget*. Graz University of Technology. Project Report. (Cited on pages 51, 86 and 107.)

Kosara, Robert [2007a]. *InfoVis 2007: InfoVis for the Masses*. `http://eagereyes.org/blog/infovis-2007-infovis-for-the-masses.html`. (Cited on page 6.)

Kosara, Robert [2007b]. *Review: Swivel vs. Many Eyes*. `http://eagereyes.org/VisCrit/Swivel-vs-Many-Eyes.html`. (Cited on pages 16, 17 and 26.)

Kosara, Robert [2010]. *The End of Verifiable.com.* `http://eagereyes.org/blog/2010/end-of-verifiable-com/`. (Cited on page 27.)

Lessacher, Martin [2009a]. *Information Visualisation Gadgets: A Survey.* Graz University of Technology. Seminar Report. (Cited on pages 1 and 23.)

Lessacher, Martin [2009b]. *Liquid Diagrams: Visual Information Gadgets in Flex.* Graz University of Technology. Project Report. (Cited on pages 1, 39, 43, 46, 48, 51, 53, 62, 65 and 69.)

Loua, Toussaint [1873]. *Atlas statistique de la population de Paris.* J. Dejey. `http://gallica.bnf.fr/ark:/12148/bpt6k81402n.image.f1`. (Cited on pages 11 and 12.)

Lyman, Peter and Hal Varian [2003]. *How Much Information?* University of California at Berkeley. `http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/`. (Cited on page 1.)

Mayr, Georg von [1877]. *Die Gesetzmässigkeit im Gesellschaftsleben, Statistische Studien.* Oldenbourg. `http://www.archive.org/download/diegesetzmssig00mayruoft/diegesetzmssig00mayruoft.pdf`. (Cited on page 9.)

Mono [2010]. *Moonlight.* `http://www.mono-project.com/Moonlight/`. (Cited on page 18.)

NCVA [2010a]. *eXplorer for Advanced Statistical Visualization.* `http://ncva.itn.liu.se/explorer?l=en`. National Centre for Visual Analytics. (Cited on page 32.)

NCVA [2010b]. *What Does OECD eXplorer Enable You to Do? An Introduction to its Main Features.* `http://www.oecd.org/dataoecd/55/47/44084514.pdf`. National Centre for Visual Analytics. (Cited on page 32.)

New York Times [2010]. *Visualization Lab.* `http://vizlab.nytimes.com/datasets/`. (Cited on page 25.)

NHL [2010]. *Player Stats.* `http://www.nhl.com/ice/playerstats.htm#?navid=nav-sts-indiv`. National Hockey League. (Cited on page 81.)

nirajswami [2007]. *Silverlight vs Flash - An Analysis Report.* `http://silverlight.net/forums/t/3015.aspx`. (Cited on page 18.)

Nuzha, Vasiliy [2010]. *Korax ColorPicker Control.* `http://kss.korax.ru/flex/cp/index.html`. (Cited on page 48.)

OECD [2010]. *OECD eXplorer: Interactive Maps for Regional Statistics.* `http://www.oecd.org/document/30/0,3343,en_2649_34413_42402025_1_1_1_1,00.html`. Organisation for Economic CO-Operation and Development. (Cited on page 31.)

Okabe, Atsuyuki, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu [2000]. *Spatial Tessellations - Concepts and Applications of Voronoi Diagrams.* Second Edition. Wiley. ISBN 0471986356. (Cited on pages 11, 13, 51, 88, 89 and 108.)

Oracle [2010a]. *JavaFX.* `http://javafx.com/about/at-a-glance.jsp`. (Cited on page 17.)

Oracle [2010b]. *JDK 6 Documentation.* `http://java.sun.com/javase/6/docs/`. (Cited on page 17.)

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. (Cited on pages 3, 4, 6 and 7.)

Ramos, Ernesto and David Donoho [2010]. *Car Data Set*. `http://stat-computing.org/dataexpo/1983.html`. (Cited on page 73.)

Refsnes Data [2010]. *Web Statistics and Trends*. `http://www.w3schools.com/browsers/browsers_stats.asp`. (Cited on page 7.)

Schiller, Jeff [2010a]. *Codedread*. `http://www.codedread.com`. (Cited on page 21.)

Schiller, Jeff [2010b]. *SVG 1.1 Browser Support*. `http://www.codedread.com/svg-support.php`. (Cited on page 21.)

SELFHTML e.V. [2010]. *Einführung in JavaScript und DOM*. `http://de.selfhtml.org/javascript/intro.htm#standards_varianten_versionen`. (Cited on page 16.)

Shneiderman, Ben [1992]. *Tree Visualization with Tree-Maps: 2-D Space-Filling Approach*. *ACM Transactions on Graphics*, 11(1), pages 92–99. ISSN 07300301. doi:10.1145/102377.115768. (Cited on page 78.)

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)*, page 336. IEEE Computer Society. ISBN 081867508X. doi:10.1109/VL.1996.545307. `http://www.cs.ubc.ca/~tmm/courses/cs533c-02/readings/shneiderman96eyes.pdf`. (Cited on pages 1, 3, 5 and 24.)

Shneiderman, Ben [2008]. *Treemaps for Space-Constrained Visualization of Hierarchies*. `http://www.cs.umd.edu/hcil/treemap-history/`. (Cited on pages 9 and 78.)

Shneiderman, Ben and Martin Wattenberg [2001]. *Ordered Treemap Layouts*. In *Proc. 2001 IEEE Symposium on Information Visualization (InfoVis 2001)*, page 73. IEEE Computer Society. ISBN 0769513425. doi:10.1109/INFVIS.2001.963283. (Cited on page 79.)

Spence, Robert [2007]. *Information Visualization: Design for Interaction*. Second Edition. Pearson. ISBN 0132065509. (Cited on page 3.)

StatLib [2010]. *Cereal Data Set*. `http://lib.stat.cmu.edu/datasets/1993.expo/`. (Cited on page 75.)

Steward, Ryan [2010]. *Differences Between Silverlight Out of Browser Experience and AIR*. `http://blog.digitalbackcountry.com/2009/03/differences-between-silverlight-out-of-browser-experience-and-air/`. (Cited on page 19.)

Sutherland, Ivan, Robert Sproull, and Robert Schumacker [1974]. *A Characterization of Ten Hidden-Surface Algorithms*. *ACM Computing Surveys*, 6(1), pages 1–55. doi:http://doi.acm.org/10.1145/356625.356626. (Cited on page 109.)

Swivel [2010]. *Swivel*. `http://www.swivel.com/`. (Cited on page 26.)

Telea, Alexandru and Jarke J. van Wijk [2001]. *Visualization of Generalized Voronoi Diagrams*. In *Proc. 2001 Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym '01)*, pages 165–174. Eurographics Association. ISBN 3211836748. `http://www.eg.org/EG/DL/WS/VisSym/VisSym01/165-174.pdf`. (Cited on page 13.)

Tufte, Edward [1983]. *The Visual Display of Quantitative Information*. Graphics Press. ISBN 0961392142. (Cited on page 3.)

Viewpath [2010]. *Viewpath*. `http://www.viewpath.com/Default.aspx`. (Cited on page 30.)

Visible Certainty [2010a]. *Going Public Beta Today*. http://blog.visiblecertainty.com/post/43134209/going-public-beta-today/. (Cited on page 27.)

Visible Certainty [2010b]. *Verifiable.com*. http://www.verifiable.com/welcome/. (Cited on page 27.)

Viégas, Fernanda Bertini [2010]. *What Happens When Just About Anyone Has Access to Sophisticated Visualization Tools?* http://fernandaviegas.com/democratizing_viz.html. (Cited on page 6.)

Viégas, Fernanda Bertini, 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. doi:10.1109/TVCG.2007.70577. http://www.research.ibm.com/visual/papers/viegasinfovis07.pdf. (Cited on page 25.)

W3C [2010a]. *Document Object Model*. http://www.w3.org/DOM/. (Cited on page 15.)

W3C [2010b]. *Flash in HTML*. http://www.w3schools.com/flash/flash_inhtml.asp. (Cited on page 35.)

W3C [2010c]. *Introduction to SVG*. http://www.w3.org/TR/2003/REC-SVG11-20030114/intro.html. (Cited on page 19.)

W3C [2010d]. *Paths*. http://www.w3.org/TR/SVG11/paths.html. (Cited on page 107.)

W3C [2010e]. *SVG 1.0 Specification*. http://www.w3.org/TR/SVG10/. (Cited on page 19.)

Ware, Collin [2004]. *Information Visualization: Perception for Design*. Second Edition. Morgan Kaufmann. ISBN 1558608192. (Cited on pages 3 and 4.)

Wattenberg, Martin [1999]. *Visualizing the Stock Market*. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems*, pages 188–189. ACM. ISBN 1581131585. doi:10.1145/632716.632834. http://www.research.ibm.com/visual/papers/marketmap-wattenberg.pdf. (Cited on page 79.)

Wattenberg, Martin [2005]. *Baby Names, Visualization, and Social Data Analysis*. In *Proc. 2005 IEEE Symposium on Information Visualization (InfoVis 2005)*, pages 1–7. IEEE Computer Society. ISBN 078039464X. doi:10.1109/INFVIS.2005.1532122. http://www.research.ibm.com/visual/papers/final-baby-margin-nocomments.pdf. (Cited on pages 6 and 25.)

Wikimedia [2010]. *Wikimedia Commons*. http://commons.wikimedia.org/wiki/Main_Page. (Cited on pages 4, 12, 104 and 106.)