Visual Analysis of Higher Education in Wales

Ferdinand Vesely

May 2011

Abstract

Project Dissertation submitted to the Swansea University
in Partial Fulfilment for the Degree of Bachelor of Science

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

Declaration

This work has not previously been accepted in substance for any degree and is not being currently submitted for any degree.

May 18, 2011

Signed:

Statement 1

This dissertation is being submitted in partial fulfilment of the requirements for the degree of a BSc in Computer Science.

May 18, 2011

Signed:

Statement 2

This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are specifically acknowledged by clear cross referencing to author, work, and pages using the bibliography/references. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure of this dissertation and the degree examination as a whole.

May 18, 2011

Signed:

Statement 3

I hereby give consent for my dissertation to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

May 18, 2011

Signed:

Contents

1 Introduction

Responsible planning of future development within any area of human activity encompasses collecting and analysing data representing current state in the area of interest. Naturally, the collected data does not have to be only limited to current state. It can add another dimension by including past states. Depending on the particular area of interest, the data which gets collected can have various properties: dimensionality, data types, structure, etc. However, it is very likely that the data collected will be sizeable at least in one dimension. Effective analysis of a large dataset is problematic. For this reason, various tools have to be employed to assist the analysis of such datasets. One of the obvious choices is a visual tool which will help present the data in such a way, that it is possible to find a suitable interpretation for it. This interpretation founds the basis for understanding of the data. Admittedly, visual tools can provide misleading output. This may be because they contain errors, they are being used incorrectly, or there is intentional manipulation to produce misleading results. Thus, it is always wishful to support the interpretation based on visual representations with other methods. Nevertheless, tools providing such visual representations are a part of decision making.

The Planning and Strategic Projects Unit (PSPU) of Swansea University takes part in a wider project of developing a regional strategy for higher education in South West Wales. As a part of this task, data has been collected about higher education institutions in the region. The data should provide understanding of how the region is provided with higher education. The PSPU has approached the Computer Science Department to help visualising the data to aid the task they are working on.

The project presented in this dissertation is an attempt to explore the possibilities of geographically oriented visualisation of the above mentioned data on higher education. The aim is to create visualisations which would link the figures expressing provision of education with the geographical location to which they are linked. As a part of the project, a software tool which allows the user to load in tabular data and visualise them in an interactive map scene has been developed. This dissertation summarises the work done on the project and includes documentation for the software tool.

The document is organised as follows. Section 2 contains the preliminaries for the project. In particular, it provides a short introduction into the topic, a literature review and an overview of previous software systems. A discussion on the specification of the project and the prototyping process that was involved in defining the specification can be found in section 3. Section 4 details the software design of the visualisation tool. Section 5 provides technical documentation of the implementation of the software. Screenshots of the software are included in the section as well. Finally, the dissertation is concluded in section 7.

2 Background

This section reviews preliminaries for this project. Here the reader can find a short introduction to information visualisation and geospatial data, an overview of related papers, previous systems and the input data.

2.1 Introduction to information visualisation

We can find several definitions of the term *visualisation*. A straightforward definition is offered by Ward, Grinstein and Keim: visualisation is the act of communicating something using graphical representations [20]. It is also possible to take a more subject-oriented approach in defining visualisation. According to R. Spence, visualisation is the act of forming a mental model or image of something [17]. The first definition implies that there is some information which needs to be effectively communicated. This assumes that the data or information has already been understood at least to some extent. We think that the second definition is more appropriate here as it is more general. Visualisation is a personal cognitive activity. It is about approaching something of interest and forming a visual image to understand it, to gain knowledge which is normally inaccessible due to the limitations of human perception. An graphical representation of some data or information is a tool helping to form the visual image. As such it is also a medium able to communicate some meaning to the perceiver.

Computer-aided visualisation, which is the main topic of this dissertation, provides an extension to this cognitive activity. The role of computers in visualisation is determined mainly by three factors:

- computer memory allows storage of large sets of data as well as a relatively fast access to them,
- increasing speeds of computation enables (inter-)active exploration of data as opposed to just observing a static representation of a single aspect, and
- computer graphics achieving high resolutions allows the visual representations to match the abilities of human perception.

These factors predetermine computer visualisation to be helpful when exploring large sets of data.

History provides some examples of successful use of visualisations to gain knowledge which was not originally obvious, or to communicate a certain message to the viewer. Both [20] and [17] list several such examples. We will look at some of those examples to illustrate the importance of good visualisations.

In the middle of the 19th century there was a severe cholera outbreak in the Soho district of London. A physician, Dr. John Snow was given the task to investigate the cause of the outbreak. To accomplish this, Dr. Snow investigated the area, interviewed the people living there and created a map, where he put a black dot for each death at the corresponding location. The result was that he was able to identify a concentration of deaths around a water pump in the area. Based on this, the pump was sealed and the cholera outbreak ended. This happened in times when the dominant theory used to explain cholera outbreaks was based on “bad air”. The map of Dr. Snow helped to convince the authorities that badly built water system was the real cause. This was a major event in the shift towards acceptance of the theory that microorganisms are the cause of diseases like cholera. Here visualisation was successfully used to discover and support a particular theory based on collected data. Figure 1 shows Dr. Snow’s map.

Another significant historic example of a visualisation is a map created by Charles Joseph Minard, a civil engineer active in the 19th century. The map effectively represents information the advance and retreat of Napoleon’s army during the French invasion of Russia in 1812. The image is a time-series visualisation depicting the number of men in the army throughout the

Figure 1: *The map drawn by Dr. John Snow as a part of investigation of the cholera outbreak in London's Soho in 19th century. The stacked represent deaths at the corresponding location. Crosses represent public water pumps. The aggregation of deaths around the water pump on Broad Street is clearly visible. This map helped to stop the cholera outbreak by convincing the authorities to seal the affected water pump. Image source:* `http://nl.wikibooks.org/wiki/Bestand:Snow-cholera-map.jpg`*. The image is in public domain.*

campaign. It is a very good example of a visualisation where numeric data expressing counts is linked with time and geographic location. Thickness of the line represents soldier counts. The time coordinates are given at the bottom along with the temperature the soldier had to endure. Moreover, the map uses colour coding to represent the two stages of the campaign: brown for advance, black for retreat. The map is remarkable for the striking amount of information it communicates to the observer. An image of the visualisation is in figure 2

In regard to this project, a maybe even more interesting visualisation by Ch. Minard is his map of France with pie charts representing the cattle count sent for consumption to Paris created in 1858. It is to be seen in figure 3.

These historic examples should offer a justification of use of visualisation in exploring and presenting data.

2.2 Geographic information systems and geospatial data

Geospatial data is a general term referring to data describing objects, or phenomena which have a specific location [20]. These spatial phenomena can be categorised as point, line, area, or surface phenomena. Their location is defined by a position within a coordinate system (geo-

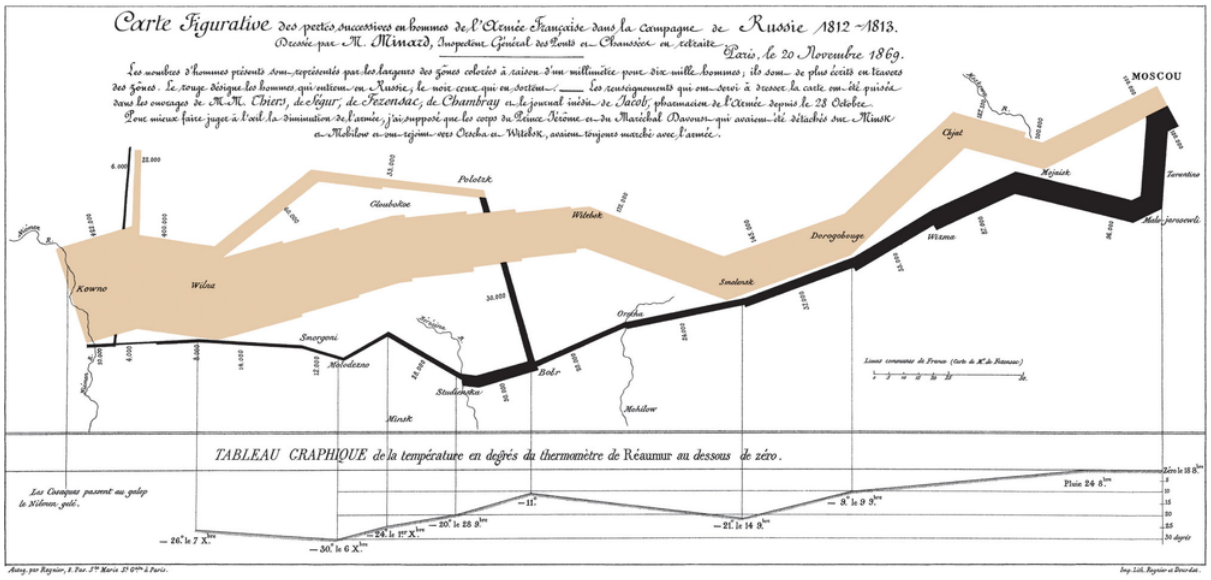

Figure 2: *A visual representation of Napoleon's invasion of Russia in 1812. The map shows the path, which Napoleon's army took on the march to Moscow as well as the retreat. The thickness of the line expresses the current soldier count in the army. The counts are also given as numbers along the path. The line forms a time-series graph annotated with dates and temperatures at the bottom. Additionally the light brown colour denotes advance, while black denotes retreat. The map was created by Charles Joseph Minard. Image source:* `http://en.wikipedia.org/wiki/File:Minard.png`*. The image is in public domain.*

reference). The location and non-spatial attributes define a geospatial data element. In this context, we understand a *map* as a visual representation of these phenomena. An important aspect of visual representations of geographic data is the topic of map projections. A map projection defines the mapping between globe coordinates, i.e. longitude (λ) and latitude (φ) and Cartesian coordinates (x, y).

The term *geographic information system* (GIS) is used to refer to computer-based systems which are concerned with the acquisition, storage, manipulation, analysis and visualisation of geographical data [3]. This data is usually represented as *theme layers* where each layer contains descriptions of different types of objects, phenomena or attributes. If the elements in the layers use a common georeferencing system, they can be stacked onto each other and processed (analysed, visualised) simultaneously. Various file formats can be used for storing GIS layer data. Mostly these formats are either raster or vector data oriented . Raster formats include general purpose image formats, like Microsoft Bitmap, JPEG 2000, or TIFF. There are also special GIS-oriented raster formats. Included here are also extensions to above-mentioned general purpose formats. These extensions add metadata containing, for example, information on coordinate systems or map projections. An example of such an extended format is GeoTIFF, which adds GIS-related metadata to the widely used TIFF format ([15]). In the context of GIS, raster data formats are mostly used to store map scans or satellite imagery.

Vector data formats used for geospatial data store information about points, lines, polygons and other shapes. These shapes are represented by a set of vector coordinates. Examples of vector formats used for geographical data are Scalable Vector Graphics (SVG), MapInfo Interchange format, Vector Product Format (VPF), or ESRI Shapefile.

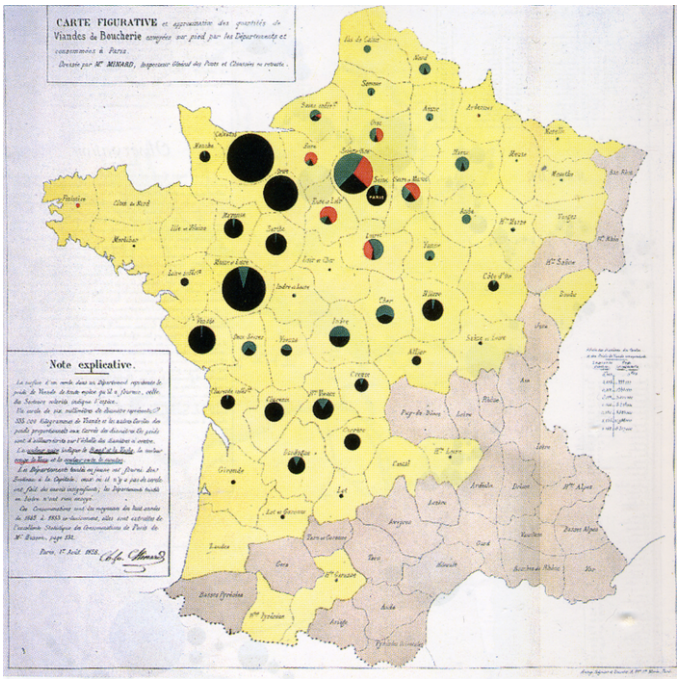

Figure 3: *Another example of Ch. Minard's art of representing information visually. The pie charts represent cattle sent to Paris for consumption. Minard created this map in 1858. Image source:* `http://en.wikipedia.org/wiki/File:Minard-carte-viande-1858.png`. *The image is in public domain.*

2.3 Related work

The papers listed here describe various aspects of multivariate data visualisation. Although they are not directly related to the approach which was taken in this project, they are nevertheless included. They have been reviewed as preparation for the development of this project and are included here to provide a more complete picture of the project work.

XmdvTool: Integrating Multiple Methods for Visualising Multivariate Data A valuable overview of some of the older multivariate data visualisation types is provided by Ward's XmdvTool[1] paper [21]. The paper sets a few criteria which can be used to evaluate visualisation techniques. These include data size constraints, occlusion, perceptibility, interaction, and use of colour. Using these criteria some visualisation types are described and evaluated. Description of the way these techniques are implemented in XmdvTool is also included. The paper describes the following techniques:

- *Scatterplots* – more specifically *scatterplot matrices*.

  In this visualisation the view is divided into $N \times N$ (where N is the number of dimensions in the dataset) smaller "sub-plots" aligned in a grid. Each dimension is drawn against all other dimensions in the dataset.

- *Glyphs* – XmdvTool uses *star glyphs* and these are described in more detail.

[1] The tool itself is reviewed below.

In a star glyph plot the dimensions of the dataset are mapped to lines originating in the same point. These lines are spaced at equal angles and their length is proportional to the value of the corresponding dimension. The end-points are then connected so the result is a polygon. Each point in dataspace is plotted as a separate polygonal glyph. In the view, the glyphs can either be aligned in a regular grid or their positions can be mapped to two dimensions of the dataspace.

- *Parallel coordinates.*

  In a parallel coordinates plot each dimension is mapped to an axis parallel to other axes. Each point in the dataspace is represented as a polyline connecting the axes.

- *Dimensional stacking.*

  Dimensional Stacking is a hierarchical visualisation technique. A pair of dimensions forms a regular grid where each position contains a grid formed by a another pair of dimensions lower in the hierarchy. The position of the point within the image is determined by recursively finding the position in each sub-plot.

The article also introduces the concept of *N-dimensional brushing* and *linking*. Brushing enables the user to visually select a subset of the dataset directly in the visualisation. Linking means that when multiple visualisations are available for display, the selections will be reflected in each visualisation.

Improving the Visual Analysis of High-dimensional Datasets Using Quality Measures When working with a large multi-dimensional dataset, the difficulty arises: how to choose the most suitable parameters for the dataset and a selected visualisation type. The paper by Albuquerque et al. [1] addresses this difficulty. They propose quality measures for three visualisation methods: Radviz, pixel-oriented displays (jigsaw maps), and table lenses for graphical compressed tables. The common goal for the measures presented by the paper is to find clusters or outliers in the visualisation. We can mention the quality measures for Radviz as an example. Radviz displays the dimensions along a circle. The position of the multi-dimensional point in the circle is dependent on the ratio of the attributes represented by the dimensions. The dimensions can be ordered arbitrarily. The quality of a particular Radviz visualisation depends on the ordering of the dimensions. It is possible to rate a visualisation according to how well it enables to identify outliers and clusters within the data. By using Gaussian and Laplacian filters it is possible to pre-process and analyse an image to identify clusters within it. The paper proposes a greedy algorithm using this technique which starts with 3 dimensions on the circle and determines the best configuration of the 3 dimensions. Then it repeats the same step for each added dimension. The algorithm terminates after all the dimensions are displayed. At the end of the process they should be configured in an optimal way, which reveals interesting patterns in the data.

Visual Analysis of Document Triage Data One of the papers that touch upon a subject more related to this project is the technical report by Geng et al. [10]. Given a multivariate dataset from a user study on document triage data, the authors compare several visualisation tools and techniques. The aim is to find suitable presentations of the data, which will enable the HCI researchers to understand the results and form better hypotheses. The visualisations presented in the paper are the following:

- a 2D stack graph using ManyEyes,

- a 3D stack graph using Microsoft Excel 2007,
- a tree map using ManyEyes,
- a matrix chart using ManyEyes,
- parallel coordinates using Xmdv,
- a multiple view visualisation in Mondrian, and
- a 3D bar chart using Excel 2007.

Overall, the paper provides a good and useful overview of the various tools and techniques for representing multi-dimensional data. Comments of the HCI researchers on the visualisations are also included in the paper.

2.4 Previous systems

There is a number of applications which can be used for visualising multi-dimensional datasets. Most of these applications are general in the sense that the user loads the data in a particular way and then chooses and configures a suitable type of visualisation. These systems allow the user to experiment with finding the proper visualisation for their data. The problem with this kind of systems is that the user already has to have some knowledge and experience with visualisation. Also some skill with manipulation and preparation of the data is necessary.

2.4.1 Liquid Diagrams[2]

Liquid Diagrams [2] are a set of visualisation tools written for the spreadsheet application of the Google Documents suite. They enable the user to directly visualise the data contained within a spreadsheet. This is possible through the Google Visalization API and the so-called gadgets, which can be inserted in the spreadsheet using the menu. This is similar to the way the insertion of charts of various types works in standard spreadsheet applications. The visualisation types which should be provided by the gadgets are: area, bar, line and pie charts, heat map, parallel coordinates, and tree map. The authors also plan to implement star plots and voronoi tree maps. The tool is promising as it delivers advanced visualisation to already existing spreadsheets. Unfortunately, at the time of writing this document the Liquid Diagrams gadgets were not available to the public. Thus, we did not have a chance to test them directly.

2.4.2 ManyEyes

ManyEyes [19] is a publicly accessible website providing a visualisation tool for a wide range of Internet users. It is being developed as a project at IBM Research. The aim of ManyEyes is to provide a community visualisation portal. The authors compare the website to YouTube. Users from many backgrounds and with different needs can upload datasets and create interactive visualisations from them. The datasets are uploaded in a simple text format using a table layout and tabs as column separators. Unformatted text is accepted as input for text visualisations. The visualisations created from the uploaded datasets are publicly accessible and other users can comment on them and modify them. This is the main focus of the social interaction on the site.

[2]The description of *Liquid Diagrams* is based on their authors' paper [2]. The gadgets themselves were not available to us for reviewing.

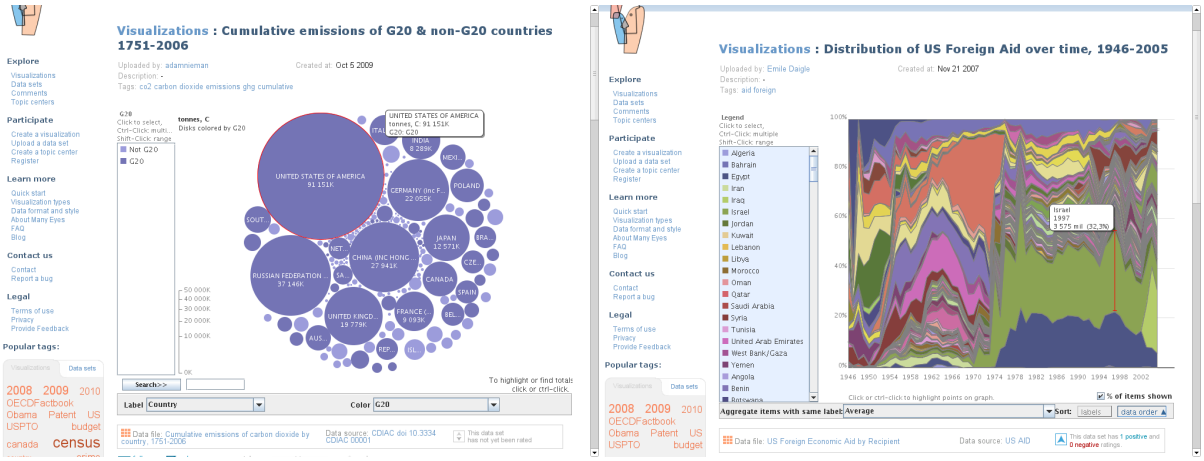

Figure 4: *Two screenshots of the* ManyEyes *web application. The picture on the left shows a* bubble chart *visualisation of cumulative CO_2 emissions (URL: `http://www-958.ibm.com/software/data/cognos/manyeyes/visualizations/cumulative-emissions-of-g20-non-g2-3`). The picture on the right is a* stack graph *visualisation of the distribution of US Foreign Aid over the years 1946-2005 (URL: `http://www-958.ibm.com/software/data/cognos/manyeyes/visualizations/distribution-of-us-foreign-aid-ove-3`).*

The visualisations may change over time so the problem of referring to a particular state of the visualisation arises. ManyEyes solves this by providing bookmarks in the form of special URLs which point to a specific state of the visualisation. It is then possible to refer to this particular state in the comments or on other pages on the web. The system also provides *annotations* which point to a particular aspect of the visualisation. ManyEyes is a very promising project, which will allow journalists, bloggers as well as other users to create good-looking interactive visualisations to accompany their work. A possible drawback is that every dataset uploaded to the site as well as every visualisation created there is public. This means that ManyEyes is not an option when privacy (or public non-availability) of the data is a concern.

The visualisation types supported by ManyEyes include: scatterplots, matrix charts, bubble charts, stack graphs, tree maps, network diagrams, and maps. Text analysis visualisations are provided by word trees, phrase nets, tag clouds, and word clouds. The website is built using standard-compliant HTML and JavaScript. The visualisations are created using Java Applets and Adobe Flash objects.

ManyEyes supports export of PNG images and allows linking to the visualisations (and their states).

See Figure 4 for screenshots.

The application is available at [11].

2.4.3 XmdvTool

XmdvTool [21] is a public domain visualisation program. It was originally created as a result merging of several small visualisation programs written by M. Ward. The software is mainly developed by academics and students with the goal of exploring multivariate data visualisation. It provides several types of visualisations for representing multivariate data. Included are parallel coordinates, scatterplot matrices, star glyphs, dimension stacking, and pixel-oriented

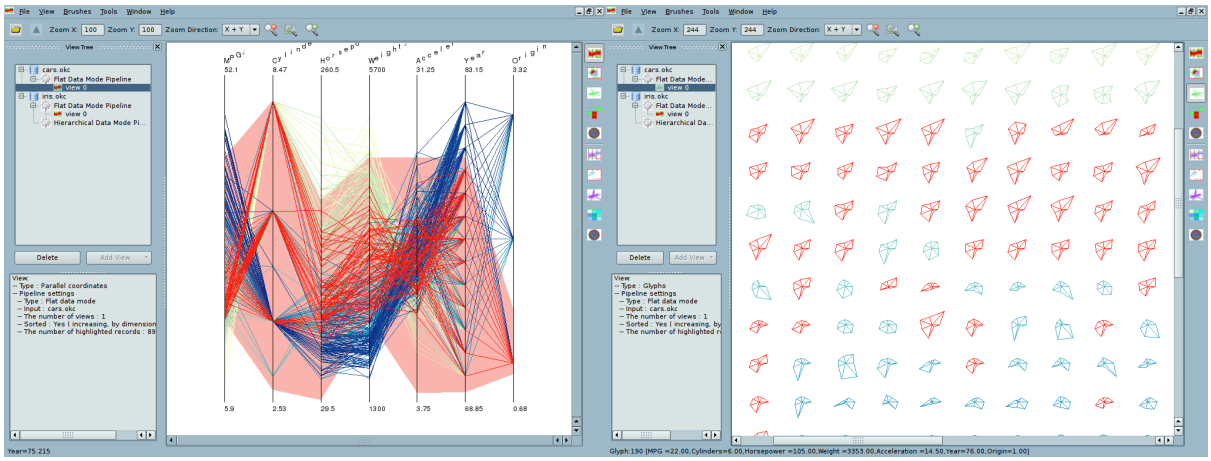

Figure 5: *Two screenshots of* XmdvTool 8.0. *The left side of the image shows a* parallel coordinates *plot in XmdvTool. The right side shows a* star glyph *plot. Both plots are representations of the "cars dataset" distributed with XmdvTool. It is possible to see the* N-dimensional brush *in the pictures. The red polylines on the parallel coordinates plot represent a selection achieved by manipulating the pink polygon in the background. Some of the corresponding glyphs can be seen in the right picture. They are highlighted in red as well.*

displays (although this is somewhat cumbersome to use). Each of the visualisation types also has a hierarchical variant. The primarily used input file format is a text file. The top of the file is occupied by meta data such as the number of dimensions and data items, the names of the dimensions ("column names" in a table terminology), and the value ranges for each dimension. The rest of the file is taken up by the data items. Dimensions are separated by blanks. The files using this format implicitly use an `.okc` extension. XmdvTool also supports CSV file import. It will automatically convert the data into the native okc format. For hierarchical plots previous versions required the users to supply a *cluster file*. The current version converts the data automatically into a binary cluster file. The users still can create their own cluster files using external tools. The program supports user interaction. After plotting the data the user can zoom in or out as well as pan the view. However, the tools could be more intuitive. A very interesting feature of XmdvTool is *N-dimensional brushing*. This enables the user to use a hyperbox to make a selection within the visualisation. The selection made in one visualisation will be reflected in the other ones. For hierarchical displays the software provides a structure-based brush [7].

Originally, XmdvTool was programmed in C using Athena Widgets and later Tcl/Tk. Currently parts of the code are written in C++ and use Qt and OpenGl[3]. XmdvTool supports export of Bitmap, JPEG and PNG images.

See Figure 5 for screenshots.

The XmdvTool can be downloaded at [22].

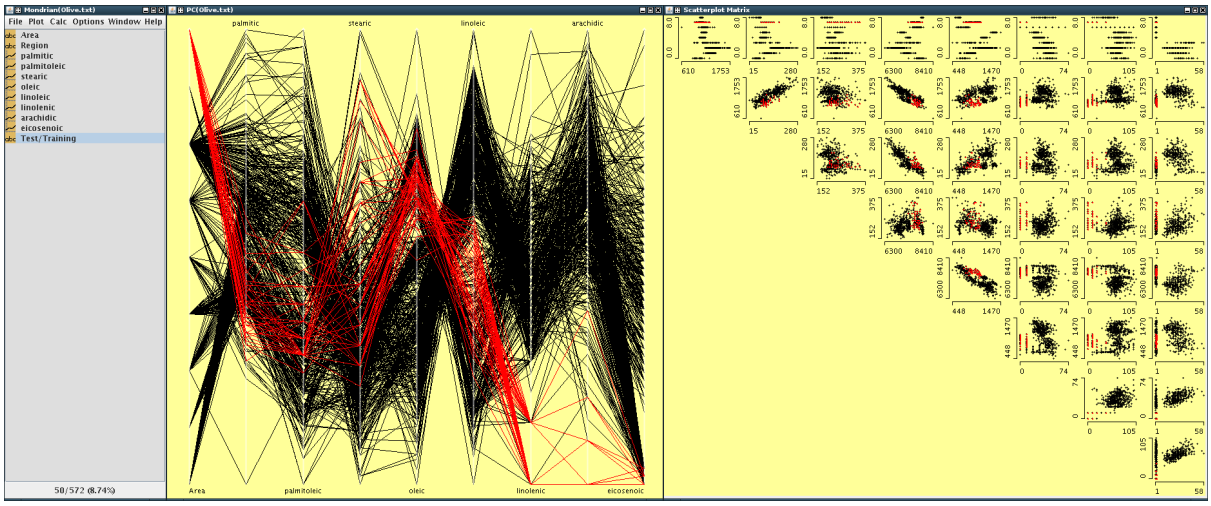

Figure 6: *The screenshot shows the main window of* Mondrian *as well as 2 visualisation windows. The dataset loaded in the application is the "Olive Oils" dataset (available from Mondrian's website), which contains data on fatty acids content in Italian olive oils. The middle window is a parallel coordinates plot and the right window is a scatterplot matrix of the same dataset. As can be seen, the selection of one area made in the parallel coordinates plot is reflected in the scatterplot matrix.*

2.4.4 Mondrian

Mondrian is another desktop visualisation tool. It is relatively similar to XmdvTool. The visualisation types it provides include scatterplot matrices, maps, histograms, bar charts, parallel coordinates, and parallel boxplots (as well as a combination of both parallel coordinates and boxplots). It also allows brushing – interactive selection of areas in the plot which are reflected in other plots of the same data. Over XmdvTool it allows the user to open multiple windows with different visualisations at once. This makes the brushing technique even more valuable than in XmdvTool. The user is able to immediately see how their selection in dataspace relates to representations in different types of visualisations. This facilitates good orientation within the data space. Mondrian allows to load the data from text files using standard comma or tab delimited value format. It also provides interaction with R[4] and has basic support for database access. The program does not allow export of images. Mondrian is being developed in Java and thus it is possible to run the binary on many platforms.

For screenshots see Figure 6.

Mondrian is available through its website at [12].

[3] An aside: The source code for XmdvTool 8.0 downloaded from the website needed fixing some errors to compile correctly (using GCC 4.4).

[4] `http://www.r-project.org/`

3 Project specification and prototyping

3.1 Preliminary requirements

At the start of the project it was unclear what exact features should be implemented. We have chosen to experiment with various approaches to visualising the data. As a guidance, we have set the following general requirements for the software in the initial document:

1. The tool should provide means to load in datasets in a suitable data format.
2. Multiple visualisation options should be available.
3. Interactive visual exploration of the data should be supported. This includes tools and techniques like brushing and linking, zooming, and panning.
4. It should provide export at least to a bitmap image format.
5. A help system should be provided.
6. The program should handle errors gracefully and recover from an error if possible.

3.2 Input data

3.2.1 Higher education data

The data, which was available for this project comes in the form of Excel spreadsheets. They are derived from data received from the Higher Education Statistics Agency (HESA). We have two types of figures available.

Student counts One spreadsheet contains the higher education provision in Wales for the academic year 2008/2009 using student counts. This file contains multiple sheets which give figures on student counts studying a particular subject area at higher education institutions in Wales. The subject areas are categorised according to the Joint Academic Coding System (JACS) devised by the HESA. The JACS subject area codes are listed in Table 1. The student counts for each subject area are further split according to student domicile into “UK”, “Other EU” and “Non EU” students. Additionally, for each institution the proportion of UK students studying at that institution is given. Aggregated totals for South West Wales, Wales and whole UK are also provided. The data is further divided into sheets according to the type of degree into:

- Undergraduate – first degree
- Undergraduate – other
- Higher Degree – taught
- Higher Degree – research
- Other Postgraduate

Each has a full time and a part time variant. This gives 10 sheets for the higher education provision file. Institutions from whole Wales are included, not just South West Wales. Figure 7 shows an example of the spreadsheet.

JACS Code	Subject Area
1	Medicine and dentistry
2	Subjects allied to medicine
3	Biological sciences
4	Veterinary science
5	Agriculture and related subjects
6	Physical sciences
7	Mathematical sciences
8	Computer science
9	Engineering and technology
A	Architecture, building and planning
B	Social studies
C	Law
D	Business and administrative studies
E	Mass communications and documentation
F	Languages
G	Historical and philosophical studies
H	Creative arts and design
I	Education
J	Combined

Table 1: *JACS Subject Area Codes. These subject areas are further subdivided into principal subjects.*

Full-time equivalent The other data set expresses education provision through full-time equivalent (FTE). FTE expresses what ratio of a full-time credit count for an academic year students are taking in a particular principal subject. The full-time credit count is 120 credits. This is equal to an FTE of 1. For example, if a student is taking 90 credits in subject A and 30 credits in subject B, this adds an FTE of 0.75 ($90/120 = 0.75$) to the total of subject A and 0.25 ($30/120 = 0.25$) to the total of subject B. The FTE data which we have currently available is split into multiple Excel spreadsheet files according to the following categories:

- Undergraduate – first degree – full-time
- Undergraduate – first degree – part-time
- Undergraduate – other – full-time
- Undergraduate – other – part-time
- Postgraduate – taught
- Postgraduate – research
- Total

The tables show the what full-time equivalent is taken in a principal subject at a particular institution. The subjects are listed as JACS codes for principal subjects. This means that the FTE tables give more detailed figures than the student counts tables which only give figures by subject areas. Again, totals for Wales and whole UK are also included. An example of the spreadsheet can be seen in Figure 8.

	A	B	C	D	E	F	G
1	**Regionalisation work - Provision**						
2	0809 Part time Other Postgraduate (HE FPE)						
3							
4							
6	**Institution**	**(1) Medicine & dentistry**			**(2) Subjects allied to medicine**		
7		**UK**	**Other EU**	**Non EU**	**UK**	**Other EU**	**Non EU**
8	Aberystwyth University	0	0	0	0	0	0
9	Bangor University	0	0	0	0	0	0
10	Cardiff University	590	80	425	415	75	20
11	University of Wales Institute, Cardiff	0	0	0	0	0	0
12	University of Glamorgan	0	0	0	80	5	0
13	Glyndŵr University	0	0	0	65	0	0
14	The University of Wales, Lampeter	0	0	0	0	0	0
15	The University of Wales, Newport	0	0	0	240	0	0
16	Royal Welsh College of Music and Drama	0	0	0	0	0	0
17	Swansea Metropolitan University	0	0	0	0	0	0
18	Swansea University	0	0	0	60	0	0
19	Trinity University College	0	0	0	0	0	0
20	The University of Wales (central functions)	0	0	0	0	0	0
21	University of Wales College of Medicine	0	0	0	0	0	0
22	**South West Wales**	0	0	0	60	0	0
23	**All Wales**	590	80	425	860	80	20
24	**All UK**	2740	175	625	16090	535	485

Figure 7: *An example of the student counts provision data spreadsheet. Displayed in OpenOffice.org.*

	A	B	C	D
1	SAS System			
2	Total FTE			
3	**subject**	**_0176_The_University_of_Wales__L**	**_0091_Swansea_Metropolitan_Unive**	**_0180_Swansea_University**
4	A1	.	.	114.96
5	A2	.	.	.
6	A3	.	.	119.46
7	A4	.	.	.
8	A9	.	.	.
9	B1	.	5.481	34.26
10	B2	.	.	19.04
11	B3	.	.	.
12	B4	.	0.186	.
13	B5	.	.	.
14	B6	.	.	8.96
15	B7	.	3.552	1269.05
16	B8	.	.	26.53
17	B9	.	84.837	273.06
18	C1	.	31.591	129.02

Figure 8: *An example of the Total Full-Time Equivalent provision data spreadsheet. Displayed in OpenOffice.org.*

3.2.2 Geospatial data

The map data available to us for this project uses the ESRI Shapefile format. The official description for this format can be found in the *ESRI Shapefile Technical Description* [5]. It is published by the Environmental Systems Research Institute (ESRI) and was created for their ArcGIS sys-

tem. The basic data storage for this format consists of three files. The shape file (extension `shp`) contains information about the vector shapes. The index file (extension `shx`) stores indexes for the shapes specified in the shape file. This serves for fast shape look-up in the shape file. The third file type (extension `dbf`) is a simple feature attribute database. The format of this file was the native format of the once popular dBASE database management system developed by Ashton-Tate. This file contains feature attributes (or attribute keys for linking with other tables) of the shapes. The file contains one record per feature, with the same record order as the shape order. Optionally additional files may be included. For example, the "projection file" (extension `prj`) contains a description of the coordinate system used (projection specification). This knowledge allows transformation between different georeference systems.

The shapes supported by the ESRI Shapefile format are points, lines and polygons. Each shape has a planar variant (with only x and y coordinates) or a variant extended to the third dimension (an additional z coordinate). Additional shape variants are achieved by adding a *measure* attribute to the three dimensional version of a shape.

For the project we have different kinds of shape files available. They have a varying degree of detail, raging from a simple polygonal contour of Wales to detailed division of Wales into communities. The most suitable for the purposes of this project appears to be the map "5 UAs", which contains a subdivision of South West Wales into the 5 unitary authorities that constitute it. An example rendering of this map can be found in Figures 11.

3.3 Prototype implementations

The implementation and prototyping work on the project started by developing routines for reading comma-separated value files and display the data contained in them in a table widget. The file reader module from this small program is also included in the final application. The program was also used as the basis for the first prototype of a map oriented visualisation tool.

The first prototype was implemented using Google Chart Tools and integrated with the CSV file reader program. A screenshot of the program can be seen in Figure 9. This prototype allows the user to select a data range from the higher education provision table. After the user confirms the selection by clicking on the "Show Map" button, a URL is constructed using the figures in the selected region. This URL is used to send a request to the Google Maps server. If the request succeeds, the resulting map picture is displayed to the user in the application window. Being designed for easy and simple visualisations to be inserted into web pages, the Google Charts have very limited possibilities. All they allow the user to do is to choose regions to be displayed, select colours for them and add simple labels as shown in Figure 9. Implementing a map visualisation similar to the map chart is quite easy and thus there is no point in embedding the Google Chart Tools into the application.

The map visualisation created using the Google Map Chart was unsatisfactory and lacked any opportunities of customisation. The next prototype was an attempt to create a versatile 3D map object using OpenGL. First approach was to create a rectangular polygon in 3D space and use texture mapping to paint a map of Wales image on it. The Figure 10 shows an example output of this prototype. The prototype allows user interaction with the object. Clicking the mouse button and dragging the mouse, the user is able to rotate the object around all three axes as well as moving it in horizontal and vertical direction. The plus and minus keys allow to zoom in and out. Implementation of the prototype revealed two basic difficulties with this approach. One difficulty is the quality of the texture. If the quality of the texture image is

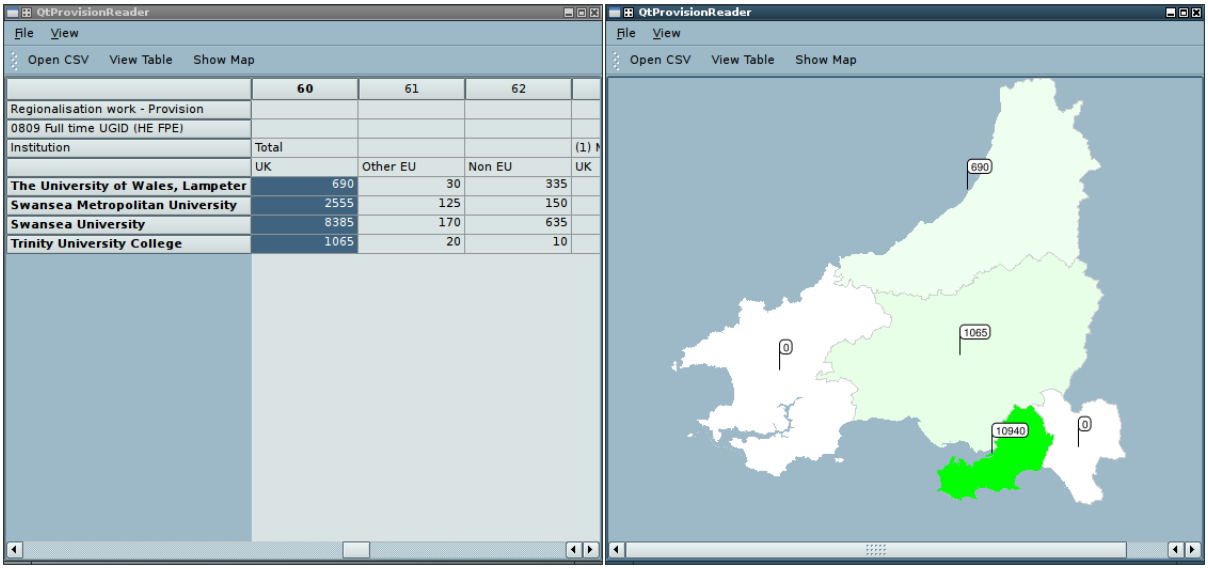

Figure 9: *A prototype using Google Chart Tools to generate a map image. The user selects a column of data to be visualised (left part of the image). After the user requests the map using the toolbar button, Google Chart Tools servers are queried and the result is displayed (right part of the image).*

low, this becomes too obvious when zooming in on the object. However, if the quality of the texture image is increased, the interactivity is slightly reduced because of the higher rendering times. Finding a good compromise between texture quality and performance is made difficult by the fact that for OpenGL, both the width and the height of the texture has to be a power of two. This means that if an image with the size 2048 (2^{11}) in one dimension is not satisfactory, the nearest possible size is 4096 (2^{12}). The other difficulty is that there is no georeferencing information given for the map texture images we can use. It would be necessary to "create" such information for the texture image. This would make it possible to place objects on the map at positions which correspond to their real-world location attribute.

To overcome both of the difficulties, an alternative approach was chosen for map rendering in 3D space. Rendering a map using a geographical vector specification solves both problems. The quality is not affected when zooming in on the map and using metadata in the vector map data file it is possible to place objects on the map with proper georeferencing. The vector map files available to us use the ESRI Shapefile format, which is described in section 3.2.2. The prototype uses the OGR library for reading shapefiles and OpenGL for rendering the map object. As a basic simple optimisation, it is possible to adjust the amount vertices to be skipped from the shape when rendering. Figure 11 contains a screenshot of the prototype. This prototype allows the same types of interactions as the texture-based one.

The last mentioned prototype provided the basis for the final application. The map rendering implementation from the prototype was reworked and improved and included in the final application.

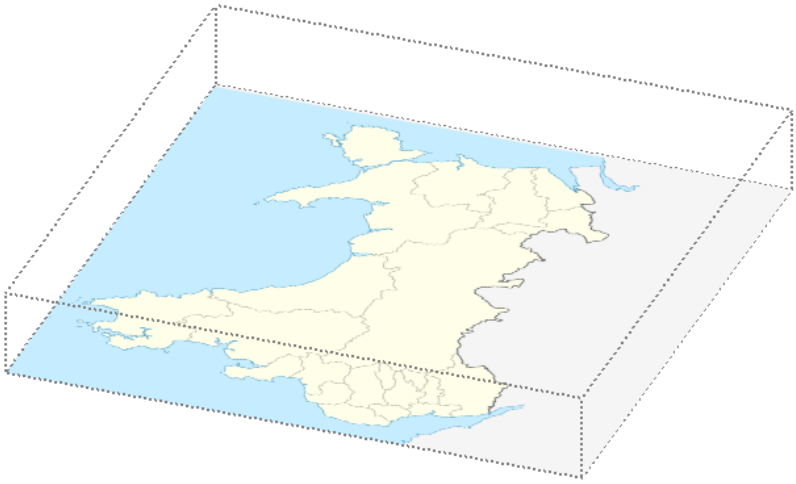

Figure 10: *A map of Wales placed into 3D space using OpenGL and texture mapping.*

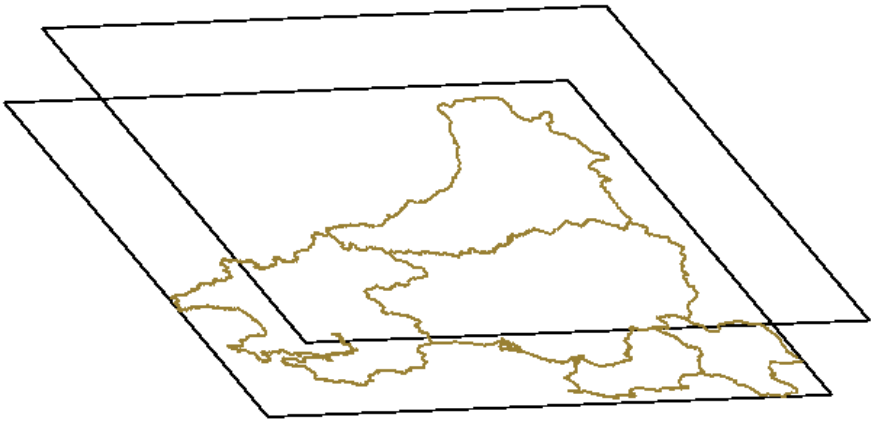

Figure 11: *Vector based map object rendered using OpenGL.*

4 Design

This section details the design of the software system. It is appropriate here to remind the reader that the software did not follow a standard software development model. It was developed in an evolutionary way. This implies that the parts of the software have been designed "on-the-fly" and gradually added to the overall design of the system.

4.1 Overview of Classes

The HiEdVis software is composed of several classes. In this subsection, we introduce the classes in the design and provide a short description of each.

MainWindow This is the main class of the application. The responsibility of the class is to create the main window of the application and manage it. This includes creation of the menus, tool-bar, data, control and visualisation views. Additionally, the class is responsible for the file I/O. Reading of the data from the user specified file is done in collaboration with CsvReader. Also the request to save an image of the visualisation scene is handled by this class.

GlSceneView The main responsibility of this class is to allow viewing of the 3D visualisation scene. This maintains a list of the 3D objects in the scene and requests them to draw themselves. It handles input events and allows interactive manipulations of the scene such: rotation, translation and scaling (zoom in and out).

MapVisManager This class is responsible for managing the map visualisations. Upon request, it processes data from the data table in the main window and creates visualisation objects. It collaborates with GlSceneView to place the objects into the scene at the correct positions. It allows enabling and disabling of objects on the map. For this it provides a user interface, where the user can select which objects are to be shown or hidden.

VisControl Customization and configuration of the applications behaviour and appearance is the main responsibility of this class. Especially it provides the user with means of controlling the appearance of the visualisation scene.

MapObject The GlObjects (described bellow) are designed to be independent of the geographical character of the visualisation they are used in. Because of this, the MapObject class encapsulates a GlObject along with its location.

CsvReader This class "knows" the CSV file format and upon request from MainWindow reads and parses the specified file. It loads the data into the memory and provides it to the application.

CsvValue This class represents a single value in the from the CSV data file. It maintains the value itself along with its type.

PersistentState This is a pure abstract class (interface). It is implemented by classes which store their state in a file.

4.1.1 3D Objects

Described bellow are the classes of objects which can be placed in the scene.

GlObject This abstract class is the ancestor of all classes of objects which can be added to the scene. The descendants of GlObject can draw themselves into the scene. They allow changing their dimensions (width, height, depth) and provide these upon request. Additionally, they have a label which they can also draw when requested.

GlMapPainter The main responsibility of this class is to draw a map in 3D space. It allows the setting of the map data file. Additionally, it takes care of placing other GlObjects at the specified latitude and longitude.

GlChartObject The abstract class GlChartObject is the ancestor for chart objects like bar charts and pie charts. A GlChartObject needs access to values, colours and labels associated with the values. It uses the information to draw a specific type of a 3D chart. Chart objects can have contours.

GlBarChart This is the class of bar chart objects. It receives a set of values along with additional information such as colours and labels for individual bars. Its responsibility is to correctly visualise the values as bars with appropriate height and colour and annotate them with the labels given.

GlPieChart Similar to GlBarChart, this is the class of pie chart objects. It receives sets of values and colours. It is then responsible for visualising the values as in a pie chart object where the slices are drawn with the appropriate angle and using the associated colour.

4.2 Diagrams

To further illustrate the relationships between the classes outlined in the previous subsection, we provide diagrams here. The design contains only one real hierarchy of classes. Most of the classes in the design shall inherit from and implement the PersistentState interface. This is due to the fact that they should allow saving their state in a settings file. The hierarchy is depicted in figure 12.

5 Implementation

This section describes how the software system is implemented. In particular it gives description of solutions to some interesting problems associated with implementing software of this kind.

The complete design as it was presented in section 4 was implemented with the exception of the PersistentState functionality for saving settings of the objects. The application is implemented using OpenGL for accelerated polygon-based 3D rendering, ORG for vector map data reading and Qt for user interface creation and other services. Also, the standard C++ library is used extensively.

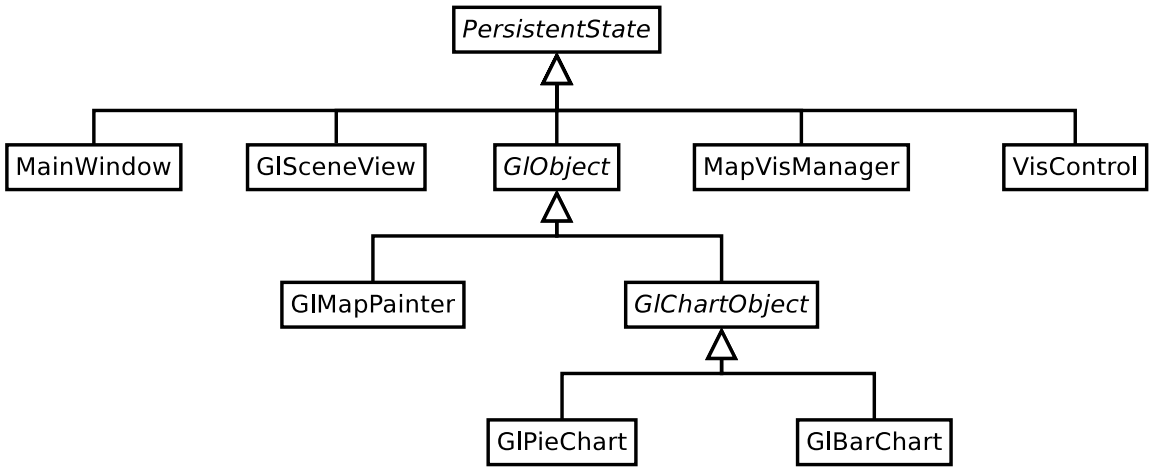

Figure 12: *The main hierarchy of the application. Abstract classes are indicated in the diagram by italic font. The abstract class* PersistentState *is the root of the hierarchy. Classes which inherit this class shall implement routines for loading and storing their state in a settings file. The abstract class* GlObject *forms the root of a "sub-hierarchy" containing 3D objects which can be placed into the 3D scene. The* GlChartObject *sub-hierarchy contains 3D chart objects, which access a list of values and render a chart object using the specified colours and labels.*

5.1 Tools and libraries

5.1.1 Programming Language: C++

C++ has been originally developed at AT&T's Bell labs by Bjarne Stroustrup as an object-oriented extension of C. The language which resulted is for many years the industry standard for general software development. It combines the advantages of compiled code which has a minimal overhead (a feature inherited from C) with the abstraction provided by the object-oriented approach. Many critics of C++ point out that the language does not provide as many run-time safety features as other more recent languages like Java. Also, as the output of the compiler is machine code (embedded in an OS-specific binary format) it is not possible to use the same binary across different platforms. However, if the software system is well designed, well implemented and the right tools are used, it is possible to develop fast, safe, and portable applications with C++. In the case of C++ the cost of portability lies on the shoulders of the developer and not the user as is the case with Java for example. Modern programming frameworks like Qt (which is introduced below) further reduce the cost and ease the implementation of large systems which need to be portable. C++ is also mostly the language of choice when programming graphics software.

5.1.2 Qt: Application framework and widget toolkit

There are certain factors which limit the choice of libraries for this project. The library has to be freely usable (at least for academic purposes), it should be accompanied with well written documentation and it should allow development for multiple platforms. The documentation factor is important to avoid or minimise risks related to unexpected (undocumented) behaviour of a part of the library. The multi-platform character of the library should enable to develop the

software for example using the Linux operating system and deploy it on Windows or Mac OS X. For most software products it is very likely that their potential users will want to use them on operating systems, which were not the primary development platform. It is an advantage if the software has been developed with this possibility in mind.

Even considering the aforementioned three constraints, there are several options for a GUI library – widget toolkit for C++. Options include GTK+, wxWidgets, Qt, Juce, FOX Toolkit, or FLTK. Probably the two most widely used toolkits from this list are GTK+ and Qt. The final choice for this project is Qt. One of the reasons is that it has a cleaner C++ API (probably due to being primarily a C++ library, whereas the C++ bindings of GTK+ are built on top of the original C library).

The development of Qt began in the early 90s in Norway. The impetus for its creation was the lack of an appropriate object-oriented GUI system for C++. By 1995 the library has been named Qt and its creators founded the company Trolltech to develop and market Qt. Today Qt is owned by Nokia, which has acquired Trolltech. The library evolved into an extensive framework for developing applications targeting various platforms. Its main advantages can be summarised as:

- a powerful signal-slot mechanism for event-driven GUI programming
- a set of high-quality widgets to be used for user interfaces

  (The look of Qt applications is themeable. On Windows and Mac OS X the widgets are rendered using the system look by default. Qt also makes it easy for the programmer to extend and modify existing widgets or to create new ones.)
- a set of general purpose classes and functions as well as a set of template classes.

  (These provide modern replacements and extensions of most of the C++ standard library. The programmer can find, for example, Unicode aware string manipulation classes or platform independent binary file format classes here.)
- a clean API which is relatively easy to learn and use
- it contains its own meta-object system which extends the C++ language
- it is very well documented

Apart from the documentation (which is available online) and the support forums, valuable sources of information about programming with Qt are [4] and [18].

5.1.3 OpenGL

OpenGL is a software library providing an interface to graphics hardware. Using the commands provided by the library, the programmer is able to take advantage of the graphics hardware. The library has implementations on multiple operating system platforms (Linux, Window, Mac OS X). On each platform, a wide array of OpenGL graphic card drivers is available. This means that source code written using the OpenGL C API is fairly portable, i.e. it can be compiled on various platforms and make use of different graphic cards. The standard reference book, informally called the “Red Book” is the *OpenGL Programming Guide* [16].

5.1.4 OGR: Geospatial vector format library

OGR is an open-source library for accessing vector map data formats. It supports various formats and data sources, including ESRI Shapefile, MapInfo, Scalable Vector Graphics (SVG) with georeferencing, spatial data from databases including Oracle, PostrgreSQL, MySQL, and other formats. The library allows accessing layers in the files, features in the layers – their geometry and attributes, and also the *spatial reference* – the projection data. It is available for download as a part of the GDAL library, which itself is a raster map handler library. They are available at [9].

5.2 Compilation and running

To compile the program, it is necessary to have a working C++ build environment. We recommend the C++ compiler included as a part of the GNU Compiler Collection (GCC) available at [8]. For development, we used GCC version 4.5.2 installed on a Debian GNU/Linux system. The software also needs the Qt library along with the header files ([14]), the GDAL library and headers ([9]) and an installation of OpenGL (this depends on the platform). Compiling starts by running the `qmake` program, which generates a Makefile for the program. The program can be then compiled by simply using `make`. This should produce a binary executable (`hiedvis` on Unix-like platforms). The program accepts one optional command-line argument. This argument is the path to an alternative map file. If no argument is specified, the default map file will be used.

The entry point of the program, `main` is located in `main.cpp`. In `main` an instance of the main application class, MainWindow is created. If a custom map data file was specified on the command line, the name of the file is passed to MainWindow. Subsequently, the main event loop of the application is started which effectively hands control over to MainWindow. Once the main window is shown, it is possible to use the application.

5.3 User interface

As mentioned before, the user interface is created using the widgets provided by the Qt library. The interface is dominated by the main window. The main window contains the menu bar with sub-menus, a tool bar for easy access to some functions, an 3D object list, the 3D scene and a tabbed widget containing the data table view and a visualisation settings section. Figure 13 shows a screen-shot of the main window of the application.

The table view is somewhat problematic. The data tables which were available to us, were very wide. Due to this it is very difficult to get an overview of the data that is selected. To ameliorate the situation slightly, next to the table, to the right, the interface provides a slider for changing the font size of the table. This is far from an ideal solution. However, font size reduced even beyond the limit of readability allows for a partial overview of the selected areas. For future work, the application would benefit from an additional area within the main window, which would provide some kind of graphical overview of the distribution of the data. This overview would be linked to the table and the visualisation scene.

The user interface is relatively standard compared to many contemporary desktop applications. This should make it easier for users to get accustomed to the application. However, usually the user expects to find the settings for the application in a separate dialogue accessible through the main menu. This is not the case with the presented system. Instead the settings are

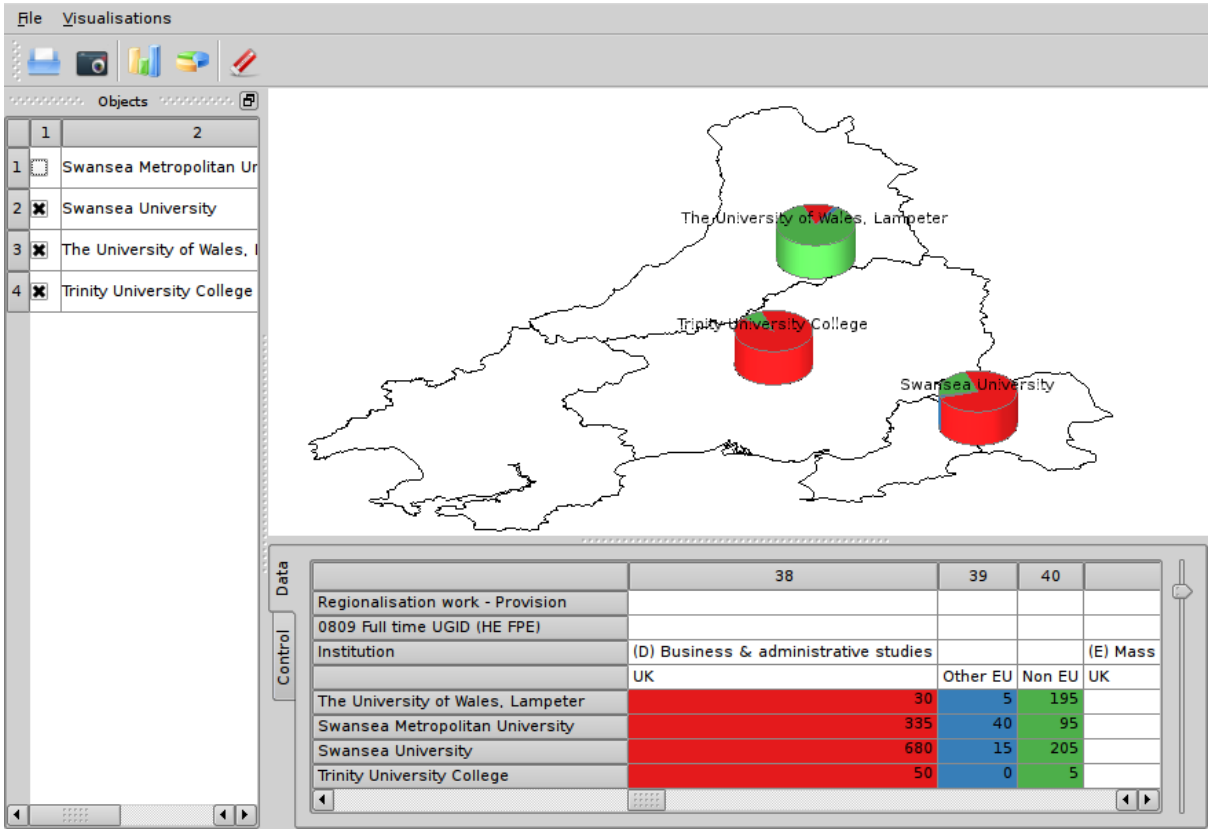

Figure 13: *Main window of the application. Menus are accessible through the menu bar at the top. Right bellow is the tool-bar containing buttons for easy access to some functions. The buttons are labelled with easy-to-understand icons. On the left is the object list, which allows deactivations of objects in the scene. The main part of the window is divided between the 3D scene view and part the data & control part. The latter is divided into tabs containing the data table and the visualisation settings.*

available under the 3D scene. This solution was chosen, because changing the settings has an immediate effect what the user sees on the screen. The intermediate step of pressing a button to apply settings was removed. This, in conjunction with the direct availability of the controls makes the user experience more direct. The user is able to immediately see how the changes affect the look and behaviour. On the other hand, if the application would become more complex, there would likely be a need for an additional separate settings dialogue to accommodate all available options. The visualisation control can be seen in figure 14.

It is possible to manipulate the 3D scene by clicking a mouse button and dragging. The following combinations are possible:

Keyboard modifier	Mouse button	Drag direction	Result
none	left or right	vertical	rotate around the X axis
none	left	horizontal	rotate around the Y axis
none	right	horizontal	rotate around the Z axis
Shift	right	horizontal	translate (move) in the horizontal direction
Shift	left	vertical	translate (move) in the vertical direction
Ctrl	left	vertical	moving up zooms in; moving down zooms out

Figure 14: *The visualisation settings ("Control"). This collection of options allows changing the appearance of the visualisations and of the application. In particular it allows to change the way the values from the table are aggregated and the appearance of the 3D scene and the objects in the scene. Additionally it is possible to set the font used for the data table. As the table is very wide, it can be helpful to change the font to a narrow face, if the user has such available.*

5.4 Chart Visualisations

The creation of geographically located 3D charts can be divided into 3 steps:

1. Collection or aggregation of user-selected data.
2. Creation of chart objects.
3. Scene update: positioning and rendering.

We will look at each of these steps in more detail.

5.4.1 Data aggregation

The application provides two types of chart visualisation: 3D bar charts and 3D pie charts. Data to be visualised is selected in the table view. The selection can have a form of one contiguous region or can be composed of multiple regions (achieved by holding down the control key). The Map Visualisation Manager is responsible for creating the visualisations upon request from the Main Window class. To achieve this, the manager accesses the data table and requests a list of the selections made by the user. It then processes the selections and aggregates the values in the table. The aggregated values are then used to create the charts, which in turn are placed

on their respective locations in the map view. The following assumptions are made about the layout of the data in the table:

- the table rows are associated to a particular location and the columns contain data for that location,
- the first column contains the name of the institution, or whatever "main key" should be used for the data, and
- there is a column containing a name of the location for the corresponding row.

As just mentioned, for the aggregation to happen correctly, the table has to contain a specially labelled column with location names for each of the rows. The location column is expected to be labelled "[[[LOCATION]]]". As the program was specially written for a particular dataset, currently only 3 locations are supported: Swansea, Lampeter and Carmarthen. The coordinates for the locations are written directly into the source code. However, it should be fairly easy to extend the program to support reading of coordinates from a user-supplied file, if such a need arises. To provide for some flexibility, a dictionary is used to associate different location names to a single location coordinate. Thus, the exact location is determined indirectly by taking the location name specified in the table, converting it to upper case, looking up the corresponding location index, and then finally retrieving the coordinates from the coordinate array. This is a relatively simple solution to the problem of determining to which location the table rows correspond.

When aggregating values, an associative array is used. The array is indexed by a string. The source of the key depends on the aggregation mode. It is possible to switch between two modes of aggregation: "group by institution" or "group by location". In the former case, the institution name (i.e. the string in the first column) is used, in the latter the location name is used. The aggregation happens for each selection by iterating over the rows, determining the location and then adding the individual column values in that row. For each row an array with the size equal to the total column count in the selections. For each column, a zero is first inserted and subsequently the values are added to the appropriate array cells. This means that grouping by location gives a sum of the values for that location. In fact, if there would be multiple rows with exactly the same institution name, they would be aggregated the same way.

5.4.2 Chart object creation

After the values from the table are collected, they are iterated over and for each $(key, value)$ pair, the selected type of chart object is created. The chart object receives the array of table values and the key value as a label. The newly created object is then wrapped into an enhanced smart pointer, `MapObject`, which associates geographic coordinates with an object and also its distance from the map plane. `MapObject` also automatically frees the char object's memory as a part of its own destruction. This smart pointer is inspired by `auto_ptr` available from the standard library. The "map object" which is created like this, gets added into `MapVisManager`'s list of current objects. Additionally, each map object is added to an associative array with locations as keys. This is a helper array which aids when finally placing the objects into the scene. After the objects are created an update of the visualisation scene is triggered.

5.4.3 Geo-positioning and rendering

The scene update is executed independently from the object creation. A scene update is triggered when objects are created or when they are enabled/disabled, that is, when the state of the corresponding check-box in the object list is toggled. A scene update is the result of collaboration of `MapVisManager`, 3D scene view (`GlSceneView`) and the OpenGL map painter (`GlMapPainter`). First `GlSceneView` is requested to clear all objects from the map scene. Then the associative array of location-objects is processed and objects from `MapVisManager`'s (possibly modified) object list are added back to the map scene. For each location, there is a vertical distance counter, which is increased every time an object is placed into the scene. The next object is placed at this height from the map plane. This allows positioning objects, which are at the same location, above each other, effectively creating stacks of objects. The responsibility for handling placement of multiple objects in the same location is held by `MapVisManager`, rather than `GlSceneView` or `GlMapPainter`. This way the latter classes do not have to deal with the equality of two location; and this is all handled within `MapVisManager`.

The placing of objects into the scene is carried out by adding the objects to `GlSceneView`'s list of object pointers. When the scene view receives an update event request, it first applies 3D transformations. Then `GlMapPainter` is requested to render the map. It also takes care for rendering of the objects in the correct locations. For this, it first computes the projected coordinates of the object and then requests the object to render itself. Naturally, the projection is computed using the specifications of the currently loaded map file. The coordinate transformation itself is computed using a utility function `GeoUtils::TransformCoordinates` (GeoUtils being a namespace), which in turn uses routines provided by the OGR library. Once the (x, y) pair of coordinates in map-plane is retrieved, `GlMapPainter` translates the OpenGL coordinate system to position the objects. Finally, it requests the objects to render themselves. Descendants of `GlObject` provide a member function `GlDraw` for this purpose. Additionally, also the labels for the objects are rendered by calling the corresponding member functions of those objects.

Figure 15 provides an example of a scene with bar charts and pie charts.

Figure 15: *Examples of 3D bar charts and pie charts rendered in the application.*

5.5 OpenGL text rendering

An interesting problem, encountered during the implementation of the software, was connected with trying to find a good way of rendering labels for the 3D objects.Surprisingly, rendering text to to annotate OpenGL objects is relatively problematic. OpenGL itself does not provide font rendering routines which would be universal to all platforms. The platform specific parts (GLX for X Window, WGL for Windows) provide some functionality to access the platform's fonts. However, this functionality is limited and would require extra work to make it portable. There are external libraries which provide font rendering with OpenGL, for example FTGL [6]. A short, slightly outdated, survey of font rendering options can be found in [13]. There are three main possibilities of rendering fonts in OpenGL:

1. Texture mapped text. The glyphs of the font are rendered into a bitmap buffer and the used as texture on objects.
2. Outline or polygon fonts. The font glyphs are transformed into polygons which can be textured, manipulated and rendered as any other polygon in OpenGL.
3. Bitmap or "overlaid" text. With this approach the text (font glyphs) are rendered directly over the framebuffer of the viewport. In its simplest form, this approach does not allow any OpenGl transformations to be applied to the rendered text.

Qt, through its `QGLWidget` class, provides a function to render overlaid text on an OpenGL viewport. There are two overloaded versions of the `renderText` function. One allows specifying the position of the text in window coordinates. The other takes 3D world coordinates, to which also all the transformations are applied. This means that the rendered text always "follows" the point as it is being moved in the scene. This version of the function also provides visibility detection, so when the parts of the text which should be behind an object are not rendered.

We have experimented with the text rendering capabilities of Qt and also with FTGL. After a few attempts with textured and outline text using FTGL, this approach was ruled out, as the transformations applied to the scene made the text unreadable from many view angles. The overlaid text solution was chosen as the best option for labelling. As in this area both Qt and FTGL offered more or less the same capabilities, FTGL was dropped to avoid adding an extra library dependence. Unfortunately, with Qt (the same applies to bitmap text in FTGL) it is only possible to specify the left starting position of the text. This is problematic when annotating objects in 3D space. When the scene gets rotated 180 degrees around the y-axis, the text starts at the corner of the object, written in the direction away from the object. The result looks like a confusing mix of objects and unrelated labels. We solved this problem by rendering centred labels. To do this, it was necessary to compensate the lack of a readily available function. Our solution takes as input the x, y, z coordinates of the centre of the rendered text, the text itself and the font to be used for rendering. The process of rendering the text is the following

1. Transform the coordinates using the current OpenGL model and projection matrices onto the current viewport and retrieve the projected coordinates. OpenGL provides a function `glProject` for this purpose.
2. Using the font metrics data determine the width and the height of the text. Qt provides functionality for this through its simple-to-use font management classes.
3. Determine the centre of the bounding box of the text. Transform the calculated coordinates of the central point back into the 3D world coordinates. The `glUnproject` function is used for this.

4. Use the 3D world-coordinate version of `renderText` to overlay the text over the scene at the relevant position.

It is possible to eliminate step 3, i.e. reverse the reverse projection of the coordinates and to use the 2D coordinate version of `renderText`. However, this way, the rendered text would not go through the visibility detection. Rendering the text with visibility detection, so that it hides behind objects when it is supposed to, aids orientation in the scene. For this reason we decided to add the extra step.

Using this approach, we are able to create good looking labels, which are always visually associated with the corresponding object in the scene.

5.6 Summary and evaluation

The software, which is submitted as a part of this dissertation, implements the following features:

1. Reading data from a CSV file. The CSV file has to be prepared for the used with the program by adding a location column. This column has to be labelled "[[[LOCATION]]]" and contain a name for the location. Currently accepted values are "Swansea", "Lampeter" and "Carmarthen".
2. The user can select regions from the data, which they want visualised.
3. Two kinds of visualisations are available: 3D pie charts and 3D bar charts. Both are rendered at the corresponding location in the map scene.
4. It is possible to interact with the scene. The user can use the mouse to move, rotate and zoom the map scene in and out.
5. Aggregation of the data can be done on a per-institute or per-location basis. The user can toggle this interactively and see the immediate result.
6. Various customisation options are available. The user can change the scene colours, fonts used for labels or the table and toggle scene lights.
7. The program allows saving of an image with a screenshot of the visualisation scene.

The program does not provide an integrated help system.

6 Conclusion

The dissertation presented a project on visualising higher education data. A short overview of the relevant topics in visualisation was given to add context to the subject. Also a survey of literature and previous software systems was presented as a part of the work on this project. The rest of the dissertation was dedicated to discussing the starting points of the project, the design and the implementation of the software tool. The software tool which was developed as a part of this project allows users to visualise numerical data related to geographic locations using 3D pie charts and bar charts visualisations.

7 Acknowledgement

I would like to thank my supervisor, Dr. Robert Laramee. Firstly, for providing me with an opportunity to work on this project. Secondly, with the help and support he has given me. I have found his comments and advices very helpful during the work. He also provided me with tips on literature and engaged with me in discussions on various topics related to this project.

I would also like to thank my second marker, Dr. Oliver Kullmann, for offering a different perspective on the project and giving suggestion.

Finally I'd like to thank my wife, Olga, for being generally supportive and trying to help always when I needed it.

Thanks also goes to people involved in projects developing the free tool which I used throughout this project. Open-source software used during writing and development includes LaTeX, Qt, GCC, Vim, OpenOffice.org, Inkscape, Gimp and Dia.

References

[1] Georgia Albuquerque, Martin Eisemann, Dirk J. Lehmann, Holger Theisel, and Marcus Magnor. Improving the Visual Analysis of High-dimensional Datasets Using Quality Measures. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (IEEE VAST)*, 10 2010.

[2] Keith Andrews and Martin Lessacher. Liquid Diagrams: Information Visualisation Gadgets. *Information Visualisation, International Conference on*, 0:104–109, 2010.

[3] Tor Bernhardsen. *Geographic Information Systems: An Introduction*. John Wiley and Sons, 3rd edition, 2002.

[4] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2008.

[5] Environmental Systems Research Institute, Inc. *ESRI Shapefile Technical Description. An ESRI White Paper*, July 1998.

[6] FTGL. `http://sourceforge.net/apps/mediawiki/ftgl/index.php?title=Main_Page`. Link valid as of May 17th, 2011.

[7] Ying-Huey Fua, M.O. Ward, and E.A. Rundensteiner. Navigating hierarchies with structure-based brushes. In *Information Visualization, 1999. (Info Vis '99) Proceedings. 1999 IEEE Symposium on*, pages 58 –64, 146, 1999.

[8] GCC, the GNU Compiler Collection. `http://gcc.gnu.org/`. Link valid as of May 17th, 2011.

[9] GDAL – Geospatial Data Abstraction Library. `http://www.gdal.org`. Link valid as of May 17th, 2011.

[10] Zhao Geng, Robert S. Laramee, Fernando Loizides, and George Buchanan. Visual Analysis of Document Triage Data. Technical report, Visual and Interactive Computing Group, Department of Computer Science, University of Wales, Swansea, December 2009.

[11] Many Eyes. `http://www-958.ibm.com/software/data/cognos/manyeyes/`. Link valid as of May 17th, 2011.

[12] Mondrian. `http://rosuda.org/mondrian/`. Link valid as of May 17th, 2011.

[13] Survey Of OpenGL Font Technology. `http://www.opengl.org/resources/features/fontsurvey`. Accessed on May 17th, 2011.

[14] Qt – Cross-platform application and UI framework. `http://qt.nokia.com/`. Link valid as of May 17th, 2011.

[15] Niles Ritter and Mike Ruth. GeoTIFF Format Specification. `http://www.remotesensing.org/geotiff/spec/geotiffhome.html`, December 2010. Accessed on May 17th, 2011.

[16] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison-Wesley, 5th edition, 2006.

[17] Robert Spence. *Information Visualization: Design for Interaction (2nd Edition)*. Pearson Education Limited, Harlow, Essex, England, 2007.

[18] Mark Summerfield. *Advanced Qt Programming: Creating Great Software with C++ and Qt 4*. Prentice Hall, 2010.

[19] F. B. Viegas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon. ManyEyes: a Site for Visualization at Internet Scale. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1121 –1128, nov. 2007.

[20] Matthew Ward, Georges Grinstein, and Daniel Keim. *Interactive Data Visualization: Foundations, Techniques and Applications*. A K Peters, Ltd., Natick, Massachusetts, 2010.

[21] Ward, Matthew O. XmdvTool: integrating multiple methods for visualizing multivariate data. In *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99)*, pages 326 –333, oct. 1994.

[22] XmdvTool Home Page. `http://davis.wpi.edu/xmdv/`. Link valid as of May 17th, 2011.