Hierarchy Browsers

Integrating Four Graph-Based Hierarchy Browsers
into the Hierarchical Visualisation System (HVS)

Alexander Nussbaumer

Hierarchy Browsers

Integrating Four Graph-Based Hierarchy Browsers
into the Hierarchical Visualisation System (HVS)

Master's Thesis

at

Graz University of Technology

submitted by

Alexander Nussbaumer

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

23th August 2005

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

TUG Graz University of Technology
Erzherzog-Johann-University

Hierarchiebrowser

Integration von vier graphbasierten Hierarchiebrowsern
in das Hierarchical Visualisation System (HVS)

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Alexander Nussbaumer

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

23. August 2005

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

Technische Universität Graz
Erzherzog-Johann-Universität

Abstract

Information visualisation seeks to transform abstract information structures into a visually understandable form. Hierarchical structures are particularly prevalent and widely used. This thesis presents four hierarchy browsers developed using the Java framework provided by the Hierarchical Visualisation System (HVS).

The four hierarchy browsers are all graph-based and use node-link visual representations. The Walker layout browser implements a classic tree drawing algorithm and is used as common basis of the other three visualisations. The hyperbolic browser is based on hyperbolic geometry, which achieves a focus plus context effect and highly efficient usage of two-dimensional space. The Magic Eye browser achieves its focus plus context effect as a result of spherical projection. Finally, the InfoLens browser makes use of a two-way fish-eye distortion technique.

Kurzfassung

Informationsvisualisierung befasst sich mit der Umwandlung abstrakter Informationsstrukturen in eine visuell verstehbare Form. Hierarchische Strukturen sind besonders gebräuchlich und weit verbreitet. Die vorliegende Diplomarbeit stellt vier Hierarchiebrowser vor, die unter Benutzung des vom Hierarchical Visualisation System (HVS) bereitgestellten Java Framework entwickelt wurden.

Die vier Hierarchiebrowser sind alle graphbasiert und verwenden ein Knoten-Kanten-Modell für die visuelle Darstellung. Der Walker Layout Browser implementiert einen klassischen Baumdarstellungsalgorithmus und wird als gemeinsame Basis der drei anderen Visualisierungen verwendet. Der hyperbolische Browser basiert auf hyperbolischer Geometrie, mit der ein focus plus context Effekt bei hocheffizienter Verwendung des zweidimensionalen Raums erzielt wird. Der Magic Eye Browser erreicht seinen focus plus context Effekt durch sphärische Projektion. Der InfoLens Browser schließlich verwendet eine Zweiwege-Fischaugenverzerrung.

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

Ich versichere hiermit, diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Acknowledgements

I especially would like to thank my advisor, Keith Andrews, for advancing scientific thinking while developing and writing, for support, and for much patience.

My colleagues at the IICM merit appreciation, those of the HVS group for suggestions and collaboration, and those of former days for prime technical explanations.

Furthermore I wish to thank my friends for efficient and productive language revisions, for assistance in terms of graphical applications, and for inspiring me during eligible, eliciting, and mirable interlocutions.

Last but not least I thank my parents, relatives, friends, and all people who have supported and encouraged me to finish this thesis.

Alexander Nussbaumer
Graz, Austria, August 2005

Credits

The following figures are used subject to the ACM Copyright Notice[1]:

- Figure 2.1 extracted from Proc. of UIST '93.
- Figure 3.6 extracted from Proc. of CHI 2000.
- Figure 3.9 extracted from Proc. of CHI '95.
- Figure 3.12 extracted from Proc. of CHI '91.
- Figure 3.15 extracted from Proc. of NPIV '99.
- Figure 6.1 extracted from Proc. of KDD 2002.
- Figure 6.2 extracted from Proc. of KDD 2002.
- Figure 8.3 extracted from Proc. of UIST 2001.

The following figures are used subject to the Copyright Notice of the University of Maryland[2]:

- Figure 2.6 and Figure 3.8.

The following resources are created by Keith Andrews:

- This thesis was written using Keith Andrews' wonderful LaTeX skeleton thesis (Andrews, 2004).
- Figures 2.3, 2.9, 3.8, 3.14, 3.16, 3.17, and 3.18 are used with kind permission.

[1] ACM Copyright Notice:
Copyright © by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM, Inc., fax +1 (212) 869-0481, or permissions@acm.org.

[2] Copyright Notice of the University of Maryland:
All works herein are Copyright © University of Maryland 1984-1994, all rights reserved. We allow fair use of our information provided any and all copyright marks, trade marks, and author attribution are retained.

Contents

List of Figures

Chapter 1

Introduction

The topic of information visualisation is introduced in Chapter 2 in terms of its origins and characteristics. Information is classified into several different types depending on its structure. Each type is explained with the use of examples.

Chapter 3 focuses on hierarchical information and explains the term *hierarchy* and how it is used in this thesis. An enumeration of properties which are usable for classifying hierarchies is followed by several visualisation methods implementing these properties.

Chapter 4 discusses the *Hierarchical Visualisation System* (HVS), which is an extensible toolkit for hierarchical visualisations. The features of HVS are described focusing on the usage and the synchronisation of the visualisations. An explanation of the software design advances the understanding of HVS.

The integration of a visualisation into HVS is described in Chapter 5 using the example of the Walker layout browser. Furthermore, this chapter describes the Walker layout algorithm and the implementation of a general graph-based browser, which the following browsers build.

Chapter 6 starts with an explanation of hyperbolic geometry followed by a method for laying out a tree in the hyperbolic plane. The features of the hyperbolic browser resulting from hyperbolic geometry characterise the functionality of this browser. Implementation details complete the discussion in this chapter.

Chapter 7 describes the Magic Eye browser and the underlying technique of three-dimensional projection from a tree laid out on a hemisphere onto a 2d disc. The features and functionality of this browser are explained. Some extensions to the original Magic Eye View and implementation details complete this chapter.

Chapter 8 introduces the InfoLens, which is a new visualisation method based on several properties and ideas of the Magic Eye View and the hyperbolic browser. The visualisation technique and the resulting features are described in detail.

A comparison of the four hierarchy browsers summarises the essential features and contrasts the strengths with weaknesses of each browser in Chapter 9. A table gives an overview of these properties for direct contrast.

A user guide in Appendix A explains the practical usage of the four presented hierarchy browsers. The menus, navigation facilities, and mouse functions are explained for each browser. Rendering settings and dialogues are listed.

Chapter 2

Information Visualisation

Research in information visualisation was started by scientific communities to handle their increasing amount of scientific data (Gershon et al., 1998). The enormous accretion of data resulted from the use of modern scientific instruments and supercomputer simulations. Since the explosive growth of the internet and the widespread use of computers, a rapid increase of digital data can be observed. Thus a need for visualisation techniques is emerging for non-scientific sectors.

Often, scientific data represents a specific structure with a given 2d or 3d geometry. Therefore, the visual representation of these data is a mapping into the given structure. Large information spaces on the internet or on desktop PCs often have no obvious geometric structure. They are abstract and can be visualised in many ways. The goal is to find an appropriate visual representation. Information visualisation seeks to visualise abstract information spaces. Since information visualisation is a new area in computer science it is comparatively easy to invent new techniques.

Both modern computers and the human mind are powerful information processing systems. Visualisation is the transformation of abstract data and information into a form that can be recognised and understood by humans. In this sense, information visualisation can be seen as an interface to abstract information spaces. So exploring large volumes of data can be done effectively by humans.

The abilities of humans to recognise visual information are highly developed (Shneiderman, 1996). Patterns, colours, shapes and textures can rapidly be detected. Movements and changes in size are observable without any difficulty. On the other hand, the perception of text-based content is much more effort than the perception of visual information. Information visualisation systems should take these human characteristics into account.

2.1 General Principles

Searching for information is different in textual and visual information systems. Textual information is often given as an alphabetical list in which the user can scroll up and down. Visual systems are more complex and offer many diverse strategies to seek for items of interest. Shneiderman (1996) introduces the *Visual Information Seeking Mantra*, which is a basic principle for visual designs of information visualisation systems:

> “Overview first, zoom and filter, details on demand.”

Shneiderman (1996) classifies seven tasks which users try to perform on information visualisation systems. They are at a high level of abstraction, further tasks are refinements of these tasks:

- *Overview*: an overview of the entire information

- *Zoom*: zoom in on a point of interest
- *Filter*: filter out uninteresting information
- *Details-on-demand*: get details of an information when needed
- *Relate*: view relationships among pieces of information
- *History*: provide undo possibilities
- *Extract*: extraction of a domain of interest

Based on these tasks some characteristics of visual information system at a higher level can be enumerated (Herman et al., 2000; Andrews, 2002). They are not necessarily needed all in the same system, but do assist in information seeking. They do not replace the support for the tasks described above, but enhance visual systems:

- *Visual Interaction*. In comparison to the amount of information the computer screen is extremely small. A simple visual representation does not fulfil the task very well to bring much information to the mind. An interactive design of the interface that makes use of the human perceptual behaviour is an essential requirement of information visualisation systems. Animated transitions help the user to understand the change of visual representation. This reduces loss of orientation.
- *Focus plus Context*. When a user zooms on a point of interest, contextual information is normally lost. The loss of context often causes a loss of orientation. There are some techniques that provide both a focus on a special item and the view of the context. The term *focus plus context* is used to describe such a technique. Examples are the hyperbolic layout (see Chapter 6) and the fish-eye view (see Chapter 8). Focus plus context is not intended to replace zooming, but rather to complement it.
- *Space-filling displays*. To show a huge amount of information, a computer display would need to be as large as a dining room table. Screen space should be used very efficiently.
- *Direct Manipulation*. When navigating through an information space, a need for manipulation of the visualised data often arises. It would be circumspect to manipulate the data somewhere else. Systems should provide the capability to manipulate the data which users see at the moment.

Following the discussion in Shneiderman (1996) and Andrews (2002), information classification can be done by considering the structure of the data. The subsequent types can be enumerated:

- *Linear data:* textual documents, alphabetical lists, etc.
- *2d data:* planar or map data (such as geographic maps)
- *3d data:* real-world objects (such as buildings)
- *Hierarchical data:* trees with links between parents and children
- *Networked data:* general graphs with an arbitrary number of links
- *Multi-dimensional data:* items (such as documents) with attributes, whereby each attribute means one dimension.
- *Content-based vector space:* content extracted from text documents are represented as a vector space.

Hierarchies are described in detail in Chapter 3, since this thesis is based on hierarchically structured data. 2d and 3d data visualisation are not further treated here, since they do not represent abstract information structures. The other types are described in the next four sections.

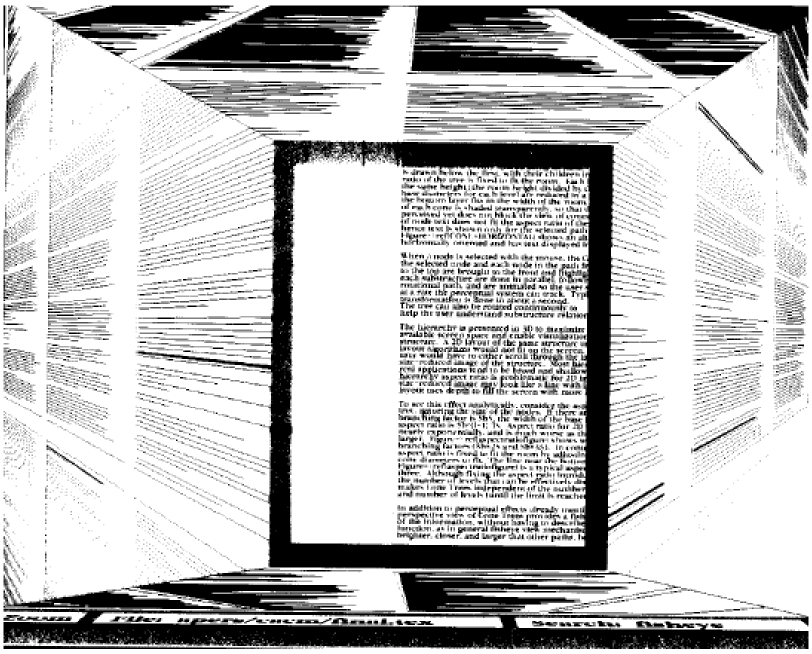


Figure 2.1: Document Lens. An entire document is shown in the Document Lens. The central panel magnifies the text part in the focus, the surrounding parts give an overview of the entire document. [Figure extracted from Proc. of UIST '93. Copyright by the Association of Computing Machinery, Inc.]

2.2 Linear Information

Linear information is organised in a sequential manner. Examples are text documents, program source code, or alphabetical lists of names. With ordinary visualisation techniques (such as scrolling), problems may arise if the amount of information is huge.

Mackinlay et al. (1991) proposed the Perspective Wall to achieve a better overview of the whole information. The information is mapped onto a 3d wall with three panels. Then the wall is projected into a 2d visual space. The two panels on the sides are smaller for perspective. Thus the information in the central panel is larger than on the sides. In this way, a natural focus plus context effect is achieved.

The Document Lens (Robertson and Mackinlay, 1993) concept extends the Perspective Wall to a five panel display (see Figure 2.1). This makes better use of the available space. The text document is laid out onto a large rectangular region. A special rectangular lens is used, which can be moved over the text area. The lens magnifies a rectangular focus area which contains the text of interest. Uniform magnification on the focus is done to keep the text readable. The other four sides contain the rest of the document.

2.3 Networked Information

For networked information a node-link model is used for the visualisation, where each piece of information is a node and the relations between them are links. Networked information has an inherent graph structure. For example, web pages are networked by hyperlinks or in a city map the crossings are linked by streets. Hierarchies are special cases of general graphs. However, a general graph can be transformed to a spanning tree plus hyperlinks.

Following the discussion in Herman et al. (2000) and Tatzmann (2004), in information visualisation

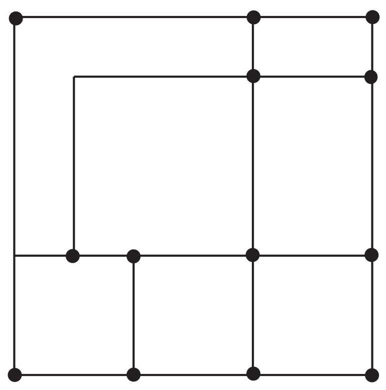


Figure 2.2: An example of an orthogonal graph drawing.

the layout of graphs is important. How a graph is drawn influences the perception of the contained information by the user. Good layouts help to better understand the hierarchical structure. Therefore aesthetic rules are often imposed on the layout, which are automatically applied by the graph drawing algorithm. However, it is often not possible to satisfy all rules in one graph. Sometimes the improvement of one rule causes a worsening of another. The most important rules are:

- *Angle.* The angle between the edges of one node should be maximised.
- *Length of edges.* Edges should all have the same length and the average of all edge lengths should be as small as possible.
- *Edge lines.* Edges should be drawn as straight lines.
- *Crossings.* Edge crossings should be avoided if possible, since they make it difficult for humans to grasp the structure.
- *Graph area.* The area for the whole graph should be minimised. This ensures space-filling at a certain zoom factor.

Many graph drawing algorithms were developed which satisfy the aesthetic rules more or less. In *orthogonal graph drawings*, nodes are placed on an orthogonal grid and edges are drawn along the grid lines (see Figure 2.2). Many aesthetic rules are accomplished, such as the maximisation of angles and the minimisation of edge crossings.

Layered-based algorithms, which implement an idea of Sugiyama (Eades and Sugiyama, 1991), are used for directed graphs. They organise the graph in several levels with the restriction that no edges between nodes at the same level are allowed. A modified algorithm of Sugiyama is implemented in the *Harmony Local Map* (Schipflinger, 1998). Besides the improvement of cycles in the graph, the layout can be focused around a specific node (see Figure 2.3). The focused node is always visualised in the centre of the display and no other nodes are in the same layer.

Another approach of graph drawing methods is to use *physical analogies*. These sorts of graph drawing algorithms place the nodes freely on the plane and use straight lines to connect the nodes. Since they use physical analogies like springs or magnetic fields, their result is intuitive for the user. Furthermore they are easy to implement. There are two types of physical analogies: force-directed placement and energy-based placement.

The first *force-directed placement* method was published by Eades (1984). The graph is modelled as a physical body, whereby the nodes are iron rings and the edges are springs. There are two different

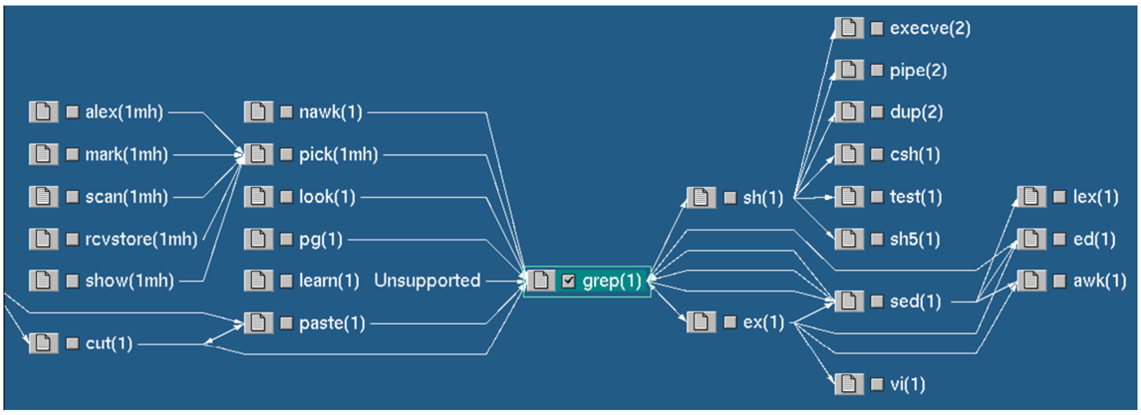


Figure 2.3: Harmony Local Map. An example of layered based graph drawing in the Harmony Local Map. [Image used with kind permission of Keith Andrews, Graz University of Technology.]

kinds of forces which have effect on the node placement, one which connects the nodes by springs, the other prohibiting collision of nodes. The algorithm works iteratively until the total stress in the system is below a certain threshold. This method is also called *spring embedder*.

The *prefuse* (Heer et al., 2005) toolkit for interactive information visualisation has implemented the spring embedder algorithm from Eades. Figure 2.4 shows a graph with relationships between people. Another example for a spring embedder implementation is *HyperSpace* (formerly Narcissus) (Hendley et al., 1995). It visualises web pages in a self-organising structure.

Energy-based placement algorithms are similar to the spring embedder. The potential energy of each spring is modelled. An objective function sums up all potential energies. Then the energy of the system is minimised by directly minimising this sum. This is done with a numerical function. In this way the position of each node is calculated.

A related technique is the *simulated annealing* method. This algorithm approximately minimises the energy of the system step by step. A temperature parameter is introduced which indicates the current energy of the system. In this way new arrangements of the graph can be compared with the previous one. The temperature is slowly reduced. This algorithm is powerful, but can be very slow. *SemNet* (Fairchild et al., 1988) uses the simulated annealing algorithm for visualising a complete knowledge base. A semantic graph is laid out in 3d space. The nodes are positioned by using the simulated annealing technique, their relations are visualised by coloured edges.

A further approach is the *hierarchical visualisation* of general graphs. The graph is reduced to a tree plus hyperlinks. The tree layout can be done by one of the layout techniques described in Chapter 5. The residual edges are added afterwards.

2.4 Multidimensional Information

In documents and files metadata attributes are often used in addition to the content. Metadata attributes span a multidimensional space, whereby each attribute represents one dimension. For example, the attributes author, title, modification date, type, and size span a 5d space. Documents are placed within this space, according to the attribute values of them. Then the multidimensional space is mapped on a 2d or 3d visual representation. There is a variety of visual representations, which can be viewed and explored by the user.

In a *parallel coordinates* visualisation the axes of the multidimensional space are drawn vertically,

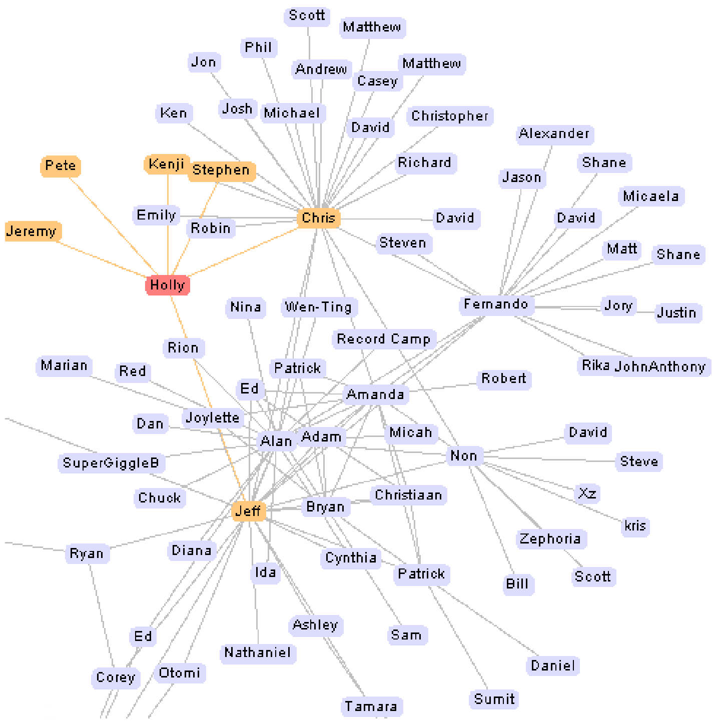


Figure 2.4: prefuse. A graph laid out with the original spring embedder algorithm of Eades.

whereby each vertical line represents an attribute. The documents are drawn as polylines from left to right. The crossings with the vertical attribute lines are determined by the attribute values of the document. Documents with similar attribute values have similar polylines, which can be detected by the user as visual pattern. One implementation of the parallel coordinates system is done by Goel (1999). Figure 2.5 shows a dataset with six dimensions and eleven documents in a parallel coordinates display.

Some information visualisation techniques use *scattergrams* to represent multidimensional data. In scattergrams points of a graph are plotted but not connected. An example for this method is FilmFinder, published by Ahlberg and Shneiderman (1994). Dynamic queries are used to filter out information by adjusting sliders and other graphical widgets. The result sets from the database are plotted on the 2d graphical display (see Figure 2.6). The parameters of the axes are the year of production and a measure of popularity. Furthermore the documents are plotted in different colours according to their genre. When seeing the graphical result, the query can be refined.

The *Table Lens* (Rao and Card, 1994) implements a technique for visualising large relational datasets. This method makes use of focus plus context when exploring a large table. Users can view details of arbitrary cells while the overview of the whole table is remained (see Figure 2.7). The distortions in the two dimensions are independent from each other. The support of multiple focus areas enables users to compare distal areas of the table. The graphical layout can be changed by sorting the data by a certain attribute. The Table Lens is a space-filling technique, which empowers users to explore much larger tables than conventional spreadsheet approaches.

The *Attribute Explorer* (Tweedie et al., 1994) uses vertical lines for the attributes, like the parallel coordinates technique. For each attribute a *histogram* of the population spread of documents is assigned to the corresponding line. The user can filter out documents by interacting with the attributes. Selecting a range of values for a certain attribute causes appearing or disappearing of documents in the other attribute histograms.

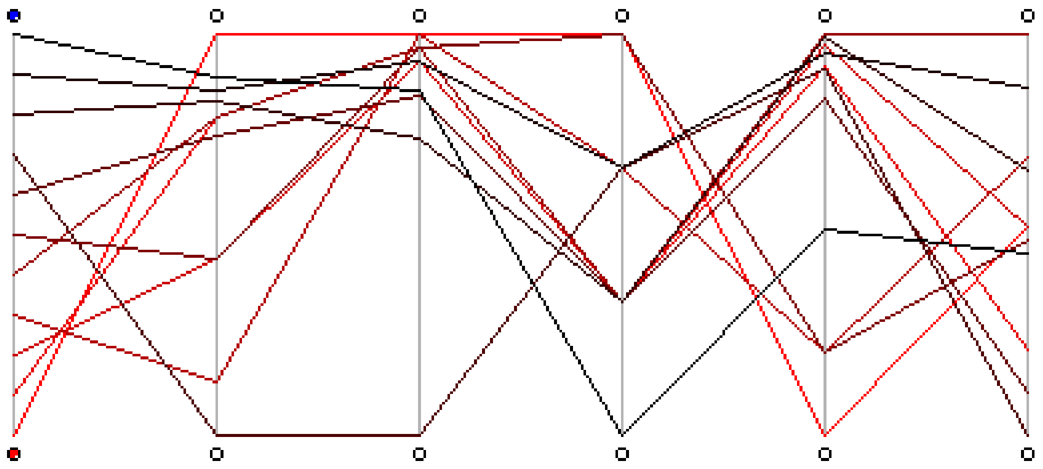


Figure 2.5: Parallel Coordinates. Metadata attributes are plotted along each vertical line. A document is represented by a polyline connecting each of its attribute values. A dataset with six attributes and eleven documents is shown.

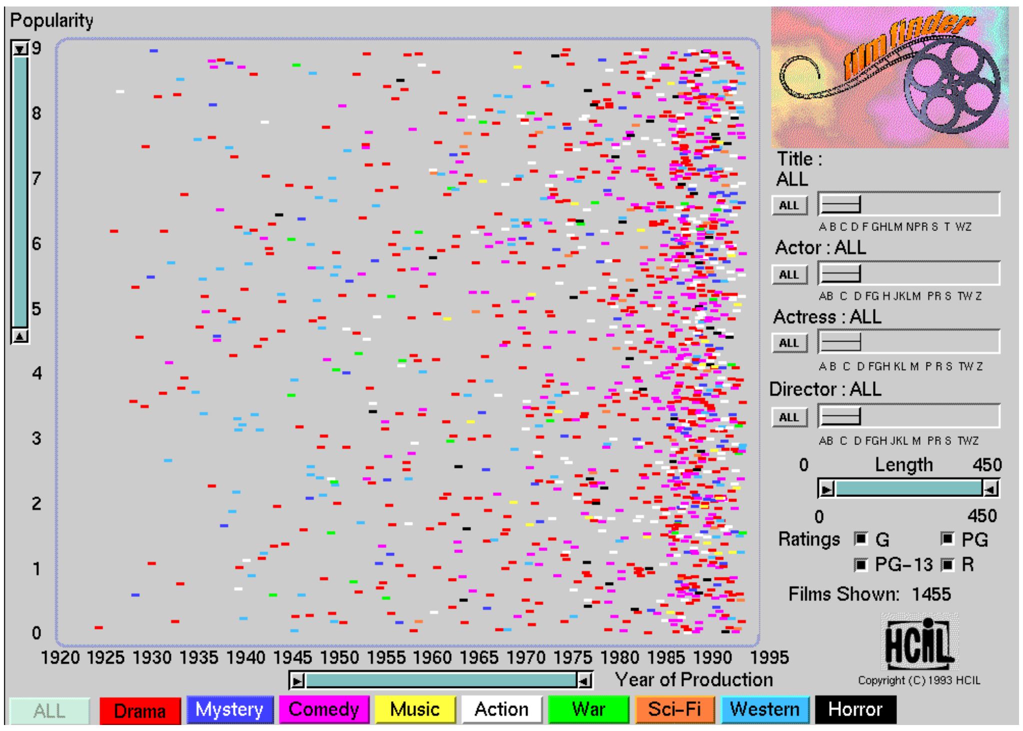


Figure 2.6: FilmFinder. A database of film from 1920 to 1995 is plotted on a scattergram. They are arranged according to their popularity value and colour coded according to their genre. [Copyright University of Maryland, all rights reserved.]

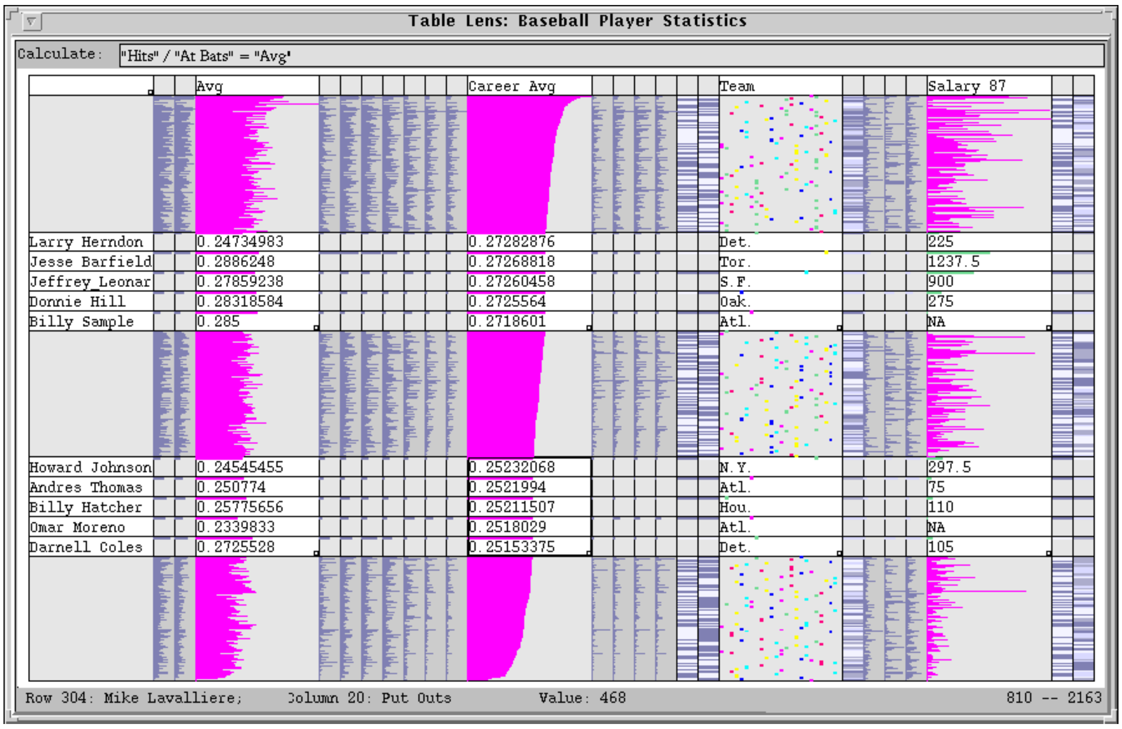


Table Lens: Baseball Player Statistics

Calculate: "Hits" / "At Bats" = "Avg"

	Avg	Career Avg	Team	Salary 87
Larry Herndon	0.24734983	0.27282876	Det.	225
Jesse Barfield	0.2886248	0.27268818	Tor.	1237.5
Jeffrey Leonar	0.27859238	0.27260458	S.F.	900
Donnie Hill	0.28318584	0.2725564	Oak.	275
Billy Sample	0.285	0.2718601	Atl.	NA
Howard Johnson	0.24545455	0.25232068	N.Y.	297.5
Andres Thomas	0.250774	0.2521994	Atl.	75
Billy Hatcher	0.25775656	0.25211507	Hou.	110
Omar Moreno	0.2339833	0.2518029	Atl.	NA
Darnell Coles	0.2725528	0.25153375	Det.	105

Row 304: Mike Lavalliere; Column 20: Put Outs Value: 468 810 -- 2163

Figure 2.7: Table Lens. Two separate areas in the table are magnified showing the focused cells. [Figure extracted from Proc. of Proc. CHI '94. Copyright by the Association of Computing Machinery, Inc.]

2.5 Content-Based Vector Spaces

Large collections of text documents can be characterised according to their content. The *vector space model* (Salton et al., 1975) is often used to characterise documents. All words contained in the whole collection of the documents span an *n*-dimensional vector space, where *n* is the total number of all unique words. Even if common words such as "is" and "the" are excluded, the number of words used is very large. Thus the vector space is very high-dimensional. Each document is a point in the *n*-dimensional space. The similarity of two documents can be calculated by the inner product of the two vectors. There are also other similarity metrics, such as euclidean distance. As for multidimensional information (see Section 2.4), a mapping from the high-dimensional space to the display is needed and several methods have been developed.

The similarity data of the documents can be interpreted as a weighted graph. The similarity value of two documents is used as the weight of the edge. No edge exists if the similarity value is zero. To visualise a content-based vector space, *graph drawing* techniques can be used, as described in Section 2.3. *VxInsight* (Davidson et al., 1998) is an implementation of this method. A combination of force-directed and energy-based placement is used for the layout. Figure 2.8 shows a bibliographic set of more than thousand articles. The shapes in the landscape indicate a higher density of similar documents.

An approach based on *neural networks* is implemented in *WEBSOM* (Lagus et al., 2004). The self-organising map (SOM) algorithm is used to organise large collections of text documents onto a 2d map displays. A document landscape is formed by arranging similar documents closely to each other. Areas of the landscape can be labelled with automatically identified descriptive words. Colour coding is used as a measure for the density of documents.

InfoSky (Kienreich et al., 2003) empowers users to explore large collections of documents. In addi-

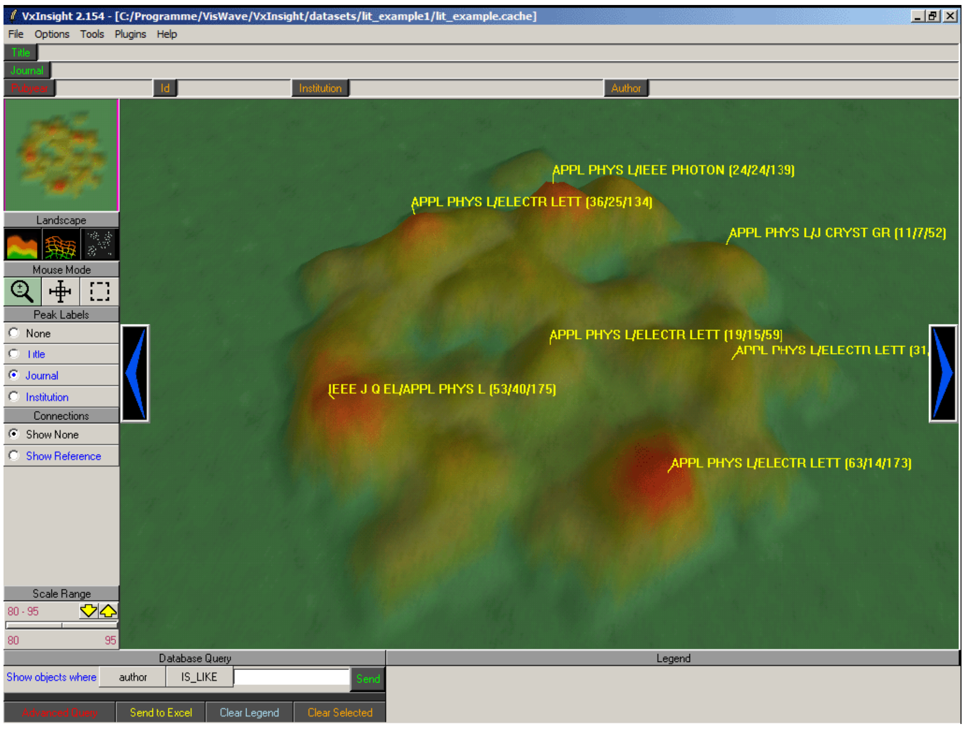

VxInsight 2.154 - [C:/Programme/VisWave/VxInsight/datasets/lit_example1/lit_example.cache]
File Options Tools Plugins Help
Title
Journal
Id
Institution
Author
Landscape
Mouse Mode
Peak Labels
None
Title
Journal
Institution
Connections
Show None
Show Reference
Scale Range
80 - 95
80
95
APPL PHYS L/IEEE PHOTON [24/24/139]
APPL PHYS L/ELECTR LETT [36/25/134]
APPL PHYS L/J CRYST GR [11/7/52]
APPL PHYS L/ELECTR LETT [19/15/59]
APPL PHYS L/ELECTR LETT [31
IEEE J Q EL/APPL PHYS L [53/40/175]
APPL PHYS L/ELECTR LETT [63/14/173]
Database Query
Legend
Show objects where
author
IS_LIKE
Send
Send to Excel
Clear Legend
Clear Selected

Figure 2.8: VxInsight. A bibliographic set of more than a thousand articles is visualised. The mountains indicate a high density of similar documents.

tion to similarity values of documents, the hierarchical structure of documents is utilised by this technique. Documents of similar content are placed close to each other and form a recognisable cluster on the display (see Figure 2.9). Bounding polygons are drawn for documents at the same level, the size of this shape depends on the number of the contained documents.

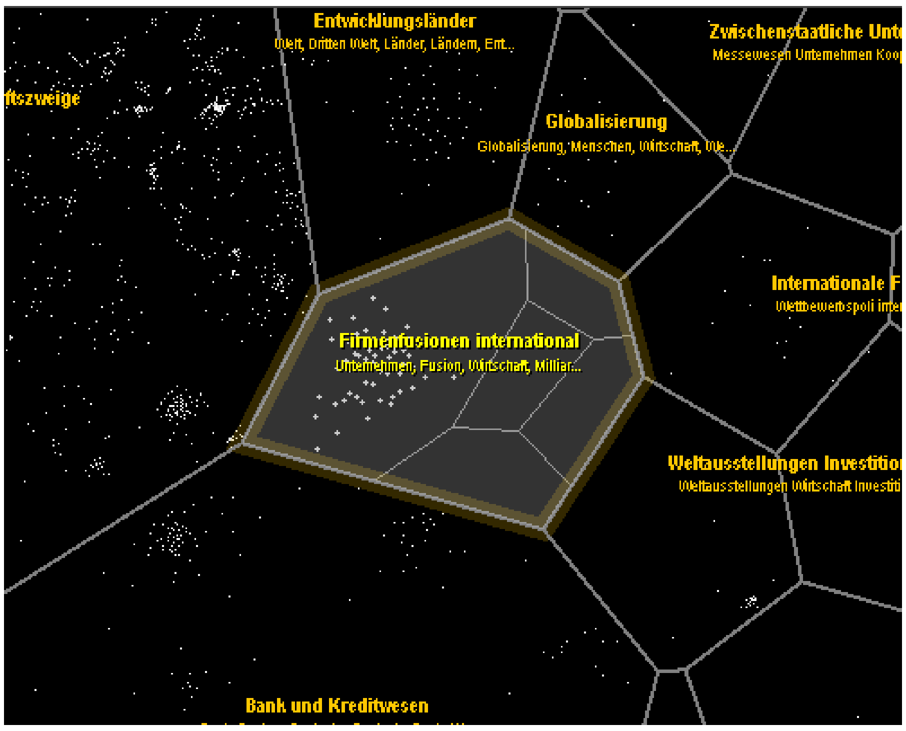

Entwicklungsländer
Welt, Dritten Welt, Länder, Ländern, Ent...
Zwischenstaatliche Unt
Messewesen Unternehmen Koo
ftszweige
Globalisierung
Globalisierung, Menschen, Wirtschaft, We...
Internationale F
Wettbewerbspoli inte
Firmenfusionen international
Unternehmen, Fusion, Wirtschaft, Milliar...
Weltausstellungen Investitio
Weltausstellungen Wirtschaft Investi
Bank und Kreditwesen

Figure 2.9: Infosky. Archived articles of a newspaper publisher are visualised as white stars. They are clustered according to their content similarity. [Image used with kind permission of Keith Andrews, Graz University of Technology.]

Chapter 3

Visualising Hierarchies

Hierarchical structuring is an approved and traditional method of organising information. A large quantity of the world's information is hierarchically structured, for example library catalogues, file systems, or corporate organisations. These structures are easily understandable, as long as they are small. Since the amount of information has heavily increased, the usage of hierarchies causes difficulties. Many new visualisation techniques have arisen to support managing hierarchical information. This chapter gives an overview of the most important methods.

A hierarchy, usually called *tree*, is a special case of a general graph. In trees there is one special node called the *root node*. All other nodes have a parent node and a unique path to the root node. Nodes with the same parent are called siblings. Each node including the root node can have child nodes. The number of branches from the root node to a specific node is called *depth* or *level* of a node. Usually, a tree has an order from top to down, in which the root node is at the top (see Figure 3.1(a)).

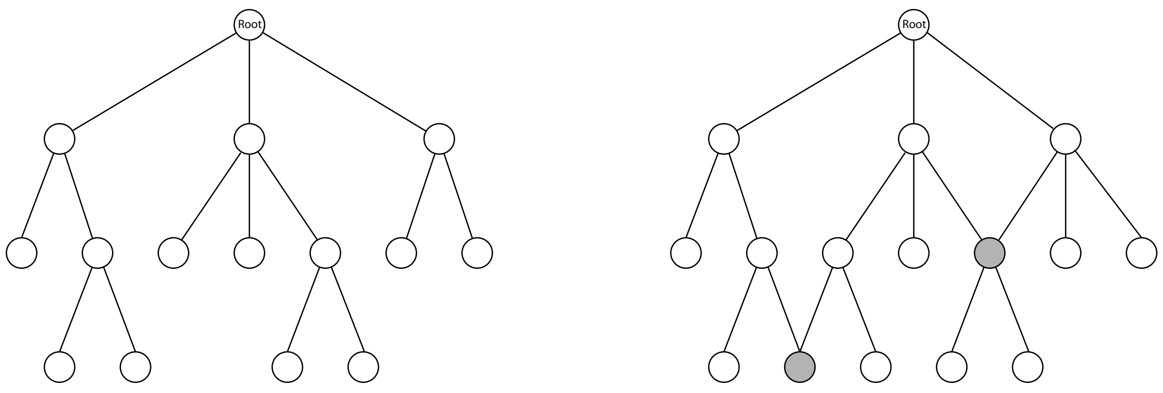
In contrast to this strict definition, hierarchies are often extended to *directed acyclic graphs* (DAG). In a DAG the nodes (except the root) may have more than one parent. Hence, there can also be more than one path from each node to the root. For example, such data structures are found in file systems, where files can be linked to several locations. In the following discussion the terms *hierarchy* and *tree* are used synonymously to mean directed acyclic graphs (see Figure 3.1(b)).

Nodes which actually have children are called *inner nodes*, nodes which do not have children are called *leaf nodes*. In a file system directories which have children are inner nodes and documents are leaves. There is a distinction between structural and content information. The structural or organisational information is built by the parent-child relationship of the tree, the content information is contained in the leaves.

3.1 Classification Issues

A taxonomy of tree visualisation methods can barely be done, since many techniques feature more than one possible classification term. Therefore the most important terms are listed and explained separately in this section. Then the visualisation methods are characterised individually in the following sections. The classification follows the discussion of the resources cited in this chapter, chiefly Johnson and Shneiderman (1991), characteristics of the visualisation methods include:

- *Listing.* Listings can provide detailed content information, but hardly any structural information. The paths are often provided as content, which forces the user to parse the path information and to build a mental model of the structure. An example of a listing is the result of the UNIX command *ls -lR*.



(a) A strict tree. **(b)** A directed acyclic graph.

Figure 3.1: The illustrations show a strict tree according to the classic tree definition and a directed acyclic graph which extends the classic tree by the property that a node can have more than one parent.

- *Outline.* Outline methods can show both structural and content information. To visualise the structure, indentation is applied to a listing. For deep or wide trees, the outlined visualisation can be very long. The necessary amount of vertical space is proportional to the number of nodes. The left panel of the Microsoft Windows Explorer is an example for an outline view (see Figure 3.2).

- *Diagram.* A tree diagram, also called a tree drawing, is a graph based view, which represents the tree as a node-link model. Nodes are drawn on a 2d area, the structural relationships are illustrated by connecting lines. This is an excellent visualisation method for small trees. However, for large trees the diagrams are very extended. Examples are the classic tree drawings described in Section 3.3.

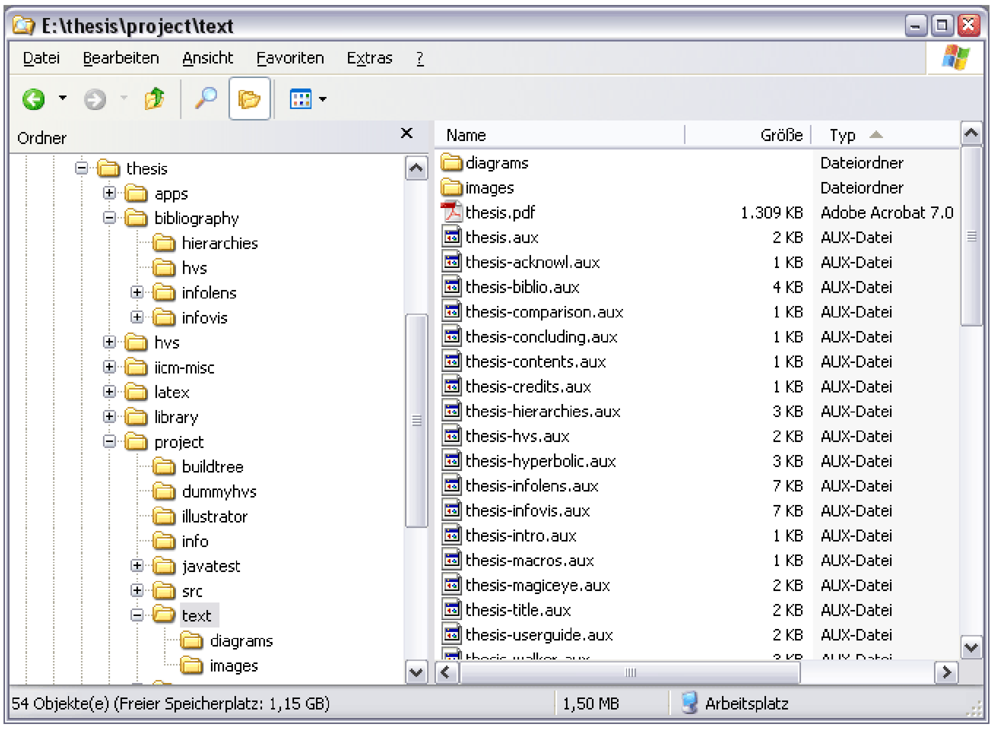
- *Radial.* Radial tree views are determined by the fact that the root node is placed in the centre of a circle and the descendants are placed around the centre on concentric circles. The higher the level of a node, the more distant the nodes are from the centre. A semi-circle can also be used instead of the whole circle. The illustration can either be done as a node-link model (see Figure 3.4) or by drawing areas (see Figure 3.14).

- *Inclusive.* In inclusive visualisations the each parent graphically contains all its descendants. Obviously, the nodes are extended in two or three dimensions. Examples are the Treemap (see Figure 3.8), in which the nodes are rectangles, or the Information Pyramids method (see Figure 3.17), which uses 3d pedestals.

- *Space-filling.* Space-filling methods make use of (almost) all of the 2d space. They satisfy as far as possible the often mentioned need for effective use of the display area. Examples are the Treemap method (see Figure 3.8) and the Information Slices technique (see Figure 3.14).

- *Distortion oriented.* Distortion oriented techniques address the often discussed problem, that large trees need more space than available on the screen. They offer a solution by illustrating the whole tree on the drawing area and magnifying the point of interest, which leads to a fish-eye view. This method is one of several focus plus context techniques (see Chapter 2). Examples are the Hyperbolic Browser (see Figure 3.9) and the Magic View (see Figure 3.15).

- *3D models.* Hierarchies can be transformed to three-dimensional models representing the structural and content information. They can be explored by rotating, scaling and zooming in 3d space.



Name	Größe	Typ
diagrams		Dateiordner
images		Dateiordner
thesis.pdf	1.309 KB	Adobe Acrobat 7.0
thesis.aux	2 KB	AUX-Datei
thesis-acknowl.aux	1 KB	AUX-Datei
thesis-biblio.aux	4 KB	AUX-Datei
thesis-comparison.aux	1 KB	AUX-Datei
thesis-concluding.aux	1 KB	AUX-Datei
thesis-contents.aux	1 KB	AUX-Datei
thesis-credits.aux	1 KB	AUX-Datei
thesis-hierarchies.aux	3 KB	AUX-Datei
thesis-hvs.aux	2 KB	AUX-Datei
thesis-hyperbolic.aux	3 KB	AUX-Datei
thesis-infolens.aux	7 KB	AUX-Datei
thesis-infovis.aux	7 KB	AUX-Datei
thesis-intro.aux	1 KB	AUX-Datei
thesis-macros.aux	1 KB	AUX-Datei
thesis-magiceye.aux	2 KB	AUX-Datei
thesis-title.aux	2 KB	AUX-Datei
thesis-userguide.aux	2 KB	AUX-Datei

Figure 3.2: Tree Browser. A focused directory with some documents are shown in the right panel, its contextual hierarchy structure as an outline in the left panel.

Examples are the Cone Tree (see Figure 3.12) and the Information Pyramids method (see Figure 3.17).

- *3D landscape.* Visualising trees as three-dimensional landscape metaphors is used to provide free navigation through 3d space. Examples are the File System Navigator (see Figure 3.16) and the Harmony Information Landscape (see Section 3.16).

3.2 Tree Browsers

Tree browsers are very popular and are used by most people working on computers. The Microsoft Windows Explorer (Figure 3.2) or the KDE Konqueror (on Linux) are traditional examples for this tree visualisation technique. Tree browsers use an outline method to visualise the hierarchical information structure.

Tree browsers provide an intuitive and easily understandable interface to hierarchical data. Usually, they consist of two panels, the left one for the structural data, which are the inner nodes, and the right panel, where content data such as document files are located. Focused documents at an arbitrary level are shown beside their context in the other panel (see Figure 3.2). Detailed information is provided by tooltips and context menus.

To navigate through the hierarchy, users expand and collapse inner nodes. Since the expanded nodes need vertical space, the visualised tree can become very large. Thus scrollbars are needed as additional graphical element. However, large hierarchies cause a huge amount of scrolling for the user. Much interaction with the browser has to be done to navigate to the items of interest.

3.3 Classic Tree Drawings

Classic tree drawing techniques deal with aesthetically pleasing drawings of trees. Wetherell and Shannon (1979) presented a tree drawing algorithm for ordered binary trees which use as little space as possible. The algorithm works in linear time and satisfies the following aesthetic rules:

- *Aesthetic 1:* Nodes of a tree at the same level should lie along a straight line, and the straight lines defining the levels should be parallel.
- *Aesthetic 2:* In a binary tree, each left son should be positioned left of its father and each right son right of its father.
- *Aesthetic 3:* A parent should be centred over its children.

Reingold and Tilford (1981) addressed some deficiencies in this algorithm. Isomorphic subtrees are not drawn identically, but depending on their position in the tree. Therefore they introduced a new aesthetic rule and presented a modified algorithm which satisfies this new rule:

- *Aesthetic 4:* A tree and its mirror image should produce drawings that are reflections of one another; moreover, a subtree should be drawn the same way regardless of where it occurs in the tree.

The tree is recursively drawn in a bottom-up pass. Leaves are placed at an arbitrary x-coordinate, the y-coordinate is determined by the the level of each node. After drawing a subtree, it is moved towards the adjacent subtree as close as possible. The subtrees are drawn independently, parent nodes are placed centrally above their children. The edges are inserted at the end. The algorithm performs these steps in linear time for binary trees. This very popular algorithm is called the *Reingold-Tilford algorithm*.

Since only binary trees are dealt with by the Reingold-Tilford algorithm, Walker II (1990) extended the algorithm to trees of arbitrary degree. The nodes are traversed from left to right. The corresponding subtrees are placed and shifted as done by the binary algorithm. In an analogous second step the nodes are traversed from right to left taking average positions of the subtrees. This leads to a well balanced tree layout (see Figure 3.3). The algorithm is described more detailed in Chapter 5.

As published in Buchheim et al. (2002), the Walker algorithm needs quadratic time, even though Walker claims linear time for it. An improved algorithm is presented by Buchheim et al. (2002) which creates the same layout in linear time.

3.4 Radial Tree Layout

In a *radial tree layout* the nodes of a tree are put on concentric circles depending on the level of the node. The root node, which is at level zero, is drawn at the centre. A recursive algorithm (Herman et al., 1999) places the children of a subtree into circular wedges (see Figure 3.4). Each parent node has its own wedge, which prohibits overlapping with adjacent subtrees. The spanning angle of a wedge is proportional, for example to the total number of leaves. The algorithm is very simple and intuitive, but it is not optimal in using the available space. Furthermore it is suitable for small and compact trees, but malfunctions for large trees, because nodes at higher levels can hardly be seen.

An implementation based on radial tree layout was done by Sheth and Cai (2003) (see Figure 3.4), which is designed to display large hierarchies. Therefore a new focus plus context technique was developed. The focus node is always placed in the centre of the layout circle, its children are rendered on the appropriate concentric circles. If the focus node is not the root node, its parent and siblings are also drawn on the concentric circles, but smaller wedges are assigned to them. When a new focus node is selected, a smooth animation does the transition to a new layout.

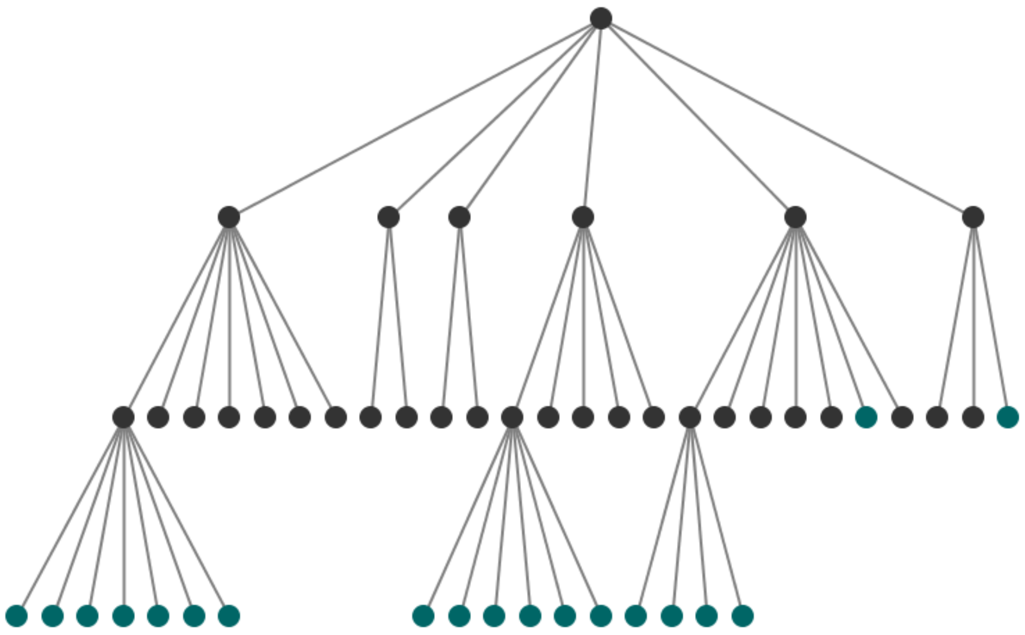
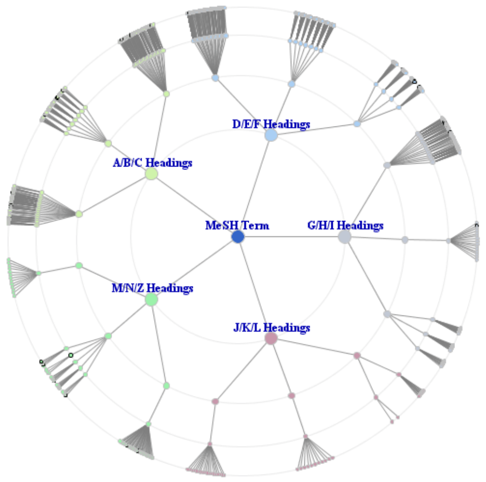
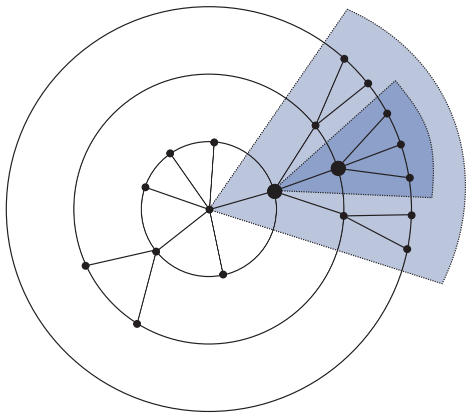


Figure 3.3: Walker Layout. A balanced tree drawing laid out by the Walker algorithm. The nodes at the highest level (at the bottom of the drawing) have the same distance to each other on the x-coordinate.

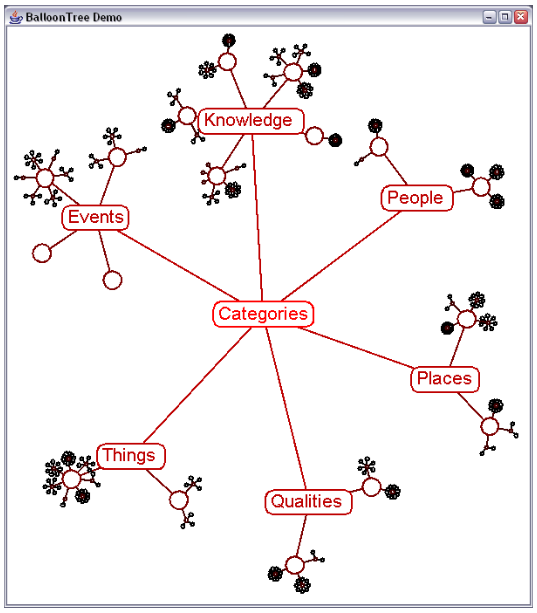


(a) A radial tree layout drawing.

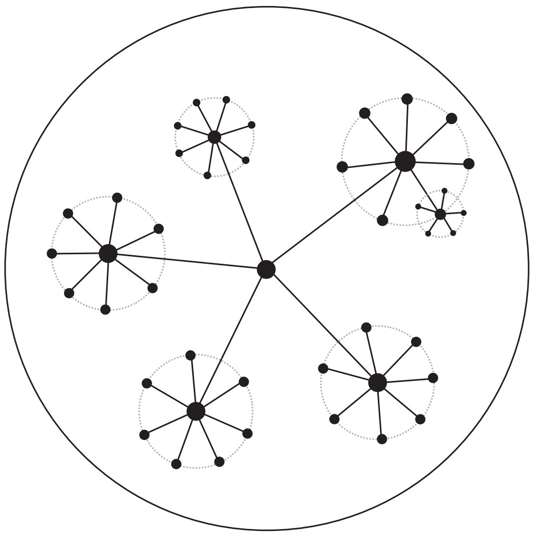


(b) A schematic illustration of a radial tree layout. Wedges are outlined for two larger drawn nodes in the first and second level.

Figure 3.4: The radial tree layout illustrated by a drawing and a schematic illustration.



(a) A balloon view implemented by prefuse (2004).



(b) A schematic illustration of the smaller circles in the balloon view.

Figure 3.5: A balloon view, also called a circular tree layout. The subtrees are laid out in recursively smaller circles.

3.5 Balloon View

A similar visualisation technique is the *circular tree layout*, which is also called the *Balloon View*. An algorithm for the layout was published by Melancon and Herman (1998). Circles with smaller radii than the layout circle are assigned to nodes at higher levels. Each node is surrounded by its children, which are placed on the circumferences (see Figure 3.5). This process is repeated recursively. Every node is a local root node of its subtree. In contrast to radial tree layouts the circles are not concentric and are used instead of wedges to assign an area to every subtree. The performance time of the algorithm is linear.

In order to modify the radius for the circle of a node, a scaling factor is assigned to it. The effect of this modification will influence the rest of the layout automatically. In this manner, subtrees can be zoomed and explored by users. The circular layout gives a good overview of the tree, but is not suitable for large trees.

3.6 Bubble Tree

A similar method to Balloon View is the *Bubble Tree*, introduced in Boardman (2000), which uses structure-based clustering of hierarchical information for the tree visualisation and navigation. Each subtree at any level is represented as a bubble, which contains the local root node and all its descendants. Initially the root bubble is opaque, so only this node is visible. To explore the hierarchy three detail-increasing interactions are available, which apply a higher level of detail to the bubble. An analogue set of three detail-decreasing interactions is supported, which provides a zoom-out functionality. The Bubble Tree method is an inclusive information visualisation technique.

The fist detail-increasing interaction is the revealing of the content of a bubble. The mouse is used to burst the bubble in order to show its children. Child bubbles which are inner nodes can also be burst

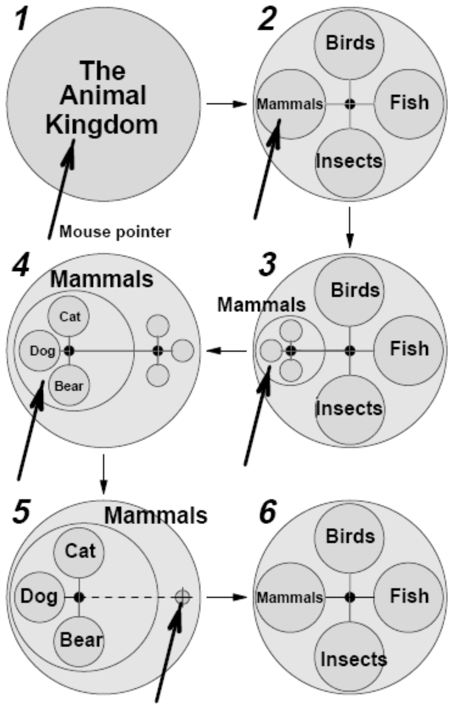


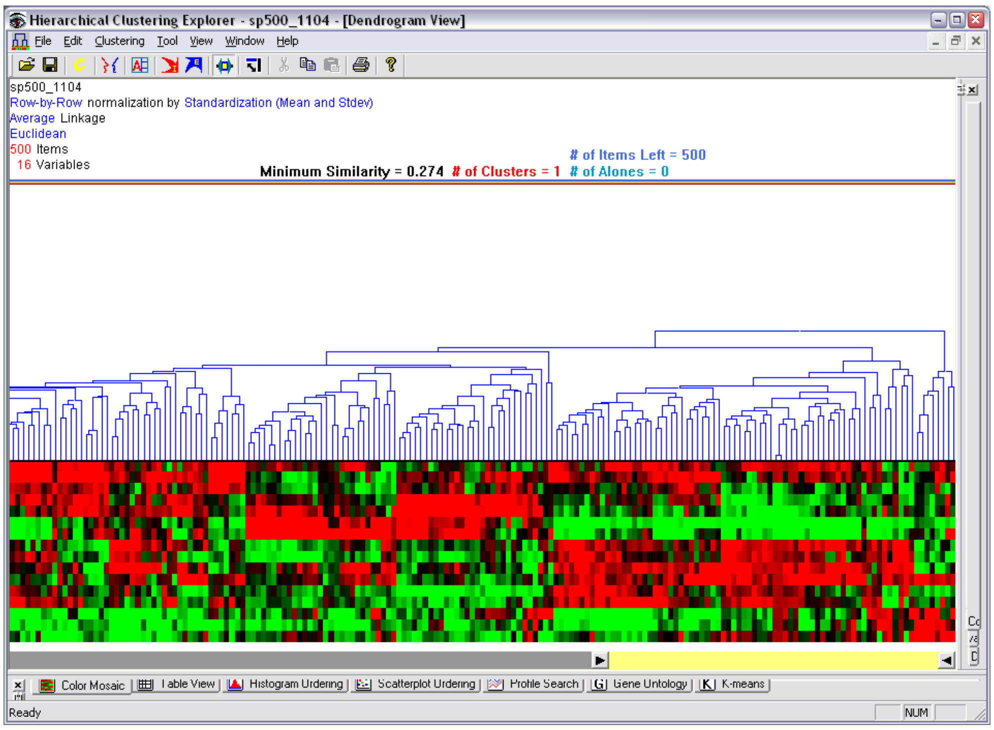
Figure 3.6: Bubble Tree. Each subtree at any level is represented as a bubble, which contains the local root node and all its descendants. To explore the hierarchy the content of a bubble is revealed and the other bubbles are downsized and moved towards the edge. [Figure extracted from Proc. of CHI 2000. Copyright by the Association of Computing Machinery, Inc.]

to disclose their immediate descendants. This leads to the second detail-increasing action. The selected bubbles are spacial expanded, while the other bubbles are downsized. A further mouse click applies the third detail-increasing action, which focuses the selected bubble. Focusing means in a bubble tree, that the selected bubble is further expanded and moved to the centre, while the other bubbles are abstracted to an opaque bubble (see Figure 3.6).

3.7 Dendrograms

Traditionally, *dendrograms* are used to visualise hierarchically clustered multidimensional information. For an explanation of multidimensional information (see Section 2.4). A cluster is a set of items with similar attributes with respect to a similarity function. In a first step, such a function seeks to detect similarities between items and builds clusters. In a second step, similar clusters are joined to form new clusters, thus creating a hierarchical structure from multidimensional data (see Figure 3.7).

The inner nodes represent clusters and groups of clusters. The leaves in the dendrogram tree are the items, which are placed on a straight line at the bottom of the drawing. Their distances on the x-axis to each other indicate their similarity distance. The dendrogram visualisation can also be rotated to have the leaves on the right on a line from top to bottom. This has the advantage that labels can be better attached to the nodes. Beside the described bottom-up approach, there is also a top-down algorithm to create a hierarchically structured cluster. Hierarchical clustering is only one method among a variety of other clustering techniques. Dendrograms are widely used to analyse and explore large sets of data. This visualisation technique empowers users to rapidly recognise patterns in multidimensional information. Natural groups can easily be detected. Application areas are for example statistics and data mining.


Hierarchical Clustering Explorer - sp500_1104 - [Dendrogram View]
File Edit Clustering Tool View Window Help
Color Mosaic
Table View
Histogram Ordering
Scatterplot Ordering
Profile Search
Gene Ontology
K-means
Ready
NUM

sp500_1104
Row-by-Row normalization by Standardization (Mean and Stdev)
Average Linkage
Euclidean
500 Items
16 Variables
Minimum Similarity = 0.274 # of Clusters = 1 # of Items Left = 500 # of Alones = 0

Figure 3.7: A dendrogram view in the Hierarchical Clustering Explorer (HCE). A set of five hundred items with sixteen attributes is visualised as a dendrogram. The colour map at the bottom represents the attributes for each item.

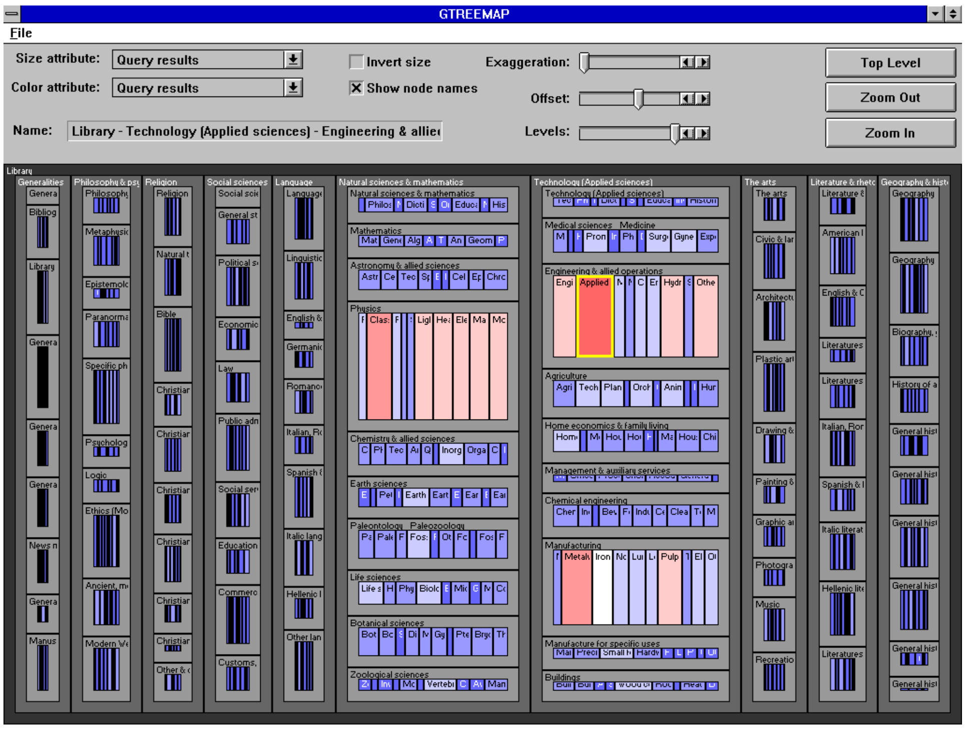
The *Hierarchical Clustering Explorer (HCE)* (Seo and Shneiderman, 2002) was implemented to find patterns in genomic microarray data (see Figure 3.7). This approach finds pairs of genes with the most similar expression profiles. It iteratively builds a hierarchy by pairing genes or existing clusters. The HCE creates a binary tree of similar genes and groups of genes.

The *TreeJuxtaposer* (Munzner et al., 2003) makes use of dendrograms to visually compare large trees. Unlike the above described method of hierarchical clustering, the TreeJuxtaposer visualises pre-existing trees as dendrograms. Trees are laid out from left to right, where the root is on the left and the labelled leaves are on the right. There are two panels for the two trees which should be compared. A focus plus context technique is used to magnify particular leaves and their labels. Colour coding is used to emphasise differences in the tree. The user can navigate to different nodes and structures by dragging the focus vertically along the leaf nodes. This system is capable of real-time interaction with a single tree of 775.000 nodes nodes.

An enhancement of the TreeJuxtaposer is the *TJC* (Beermann et al., 2005), which allows interactive browsing of trees of up to 15 million nodes. Having analysed the weaknesses of the TreeJuxtaposer, new drawing and culling algorithms were implemented and memory usage was refined.

3.8 Treemaps

The *Treemap* visualisation technique maps hierarchically structured information onto a rectangular 2d display in a space-filling manner. Unlike other methods, Treemaps use the complete available space. The entire structural and content information is drawn in a single panel. The drawing area is partitioned into rectangles representing tree nodes. Rectangle which are inner nodes are also partitioned in order to contain their child nodes. This is recursively done until the whole tree is drawn (see Figure 3.8).



Size attribute: Query results
Color attribute: Query results
Invert size
[✓ Show node names]
Exaggeration:
Offset:
Levels:
Name: Library - Technology (Applied sciences) - Engineering & allie

Figure 3.8: Treemap. A directory structure laid out by the slice and dice algorithm. [Copyright University of Maryland, all rights reserved.]

The sizes of rectangles represents the weight of the corresponding nodes. The weight is calculated from the attribute information, such as file size or creation date, if the items are files. This makes it easy for users to rapidly detect large or old files. There are different layout algorithms which determine how the rectangles are partitioned.

The original Treemap algorithm described in Johnson and Shneiderman (1991), is called *slice and dice*. Partitioning is done alternately horizontally and vertically. The root node, which lies on the whole drawing area, is divided into a number of vertical slices, where the number is determined by the number of its children. Then each child is divided in horizontal slices, also according to the number of its children. This is recursively done until all nodes are drawn. The deeper the tree, the smaller the rectangles representing the nodes (see Figure 3.8).

The slice and dice algorithm often produces long thin rectangles which are difficult to see. The *Squarified Treemaps* method (Bruls et al., 2000) overcomes this problem. An algorithm is introduced which forces the slices to be more square. The partitioning is not done either horizontally or vertically, but a combination is used. Thus, child nodes are placed as sub-rectangles into nested parent rectangles. This method loses any ordering of the tree, therefore refinements were later made to produce ordered squarified Treemaps.

3.9 Hyperbolic Browser

The *hyperbolic browser*, introduced in Lamping et al. (1995), uses a focus plus context technique to visualise large trees. The method is based on hyperbolic geometry, which provides infinite space within a drawing circle. The tree is laid out as a node-link model, and the root node is initially placed in the centre of the circle.

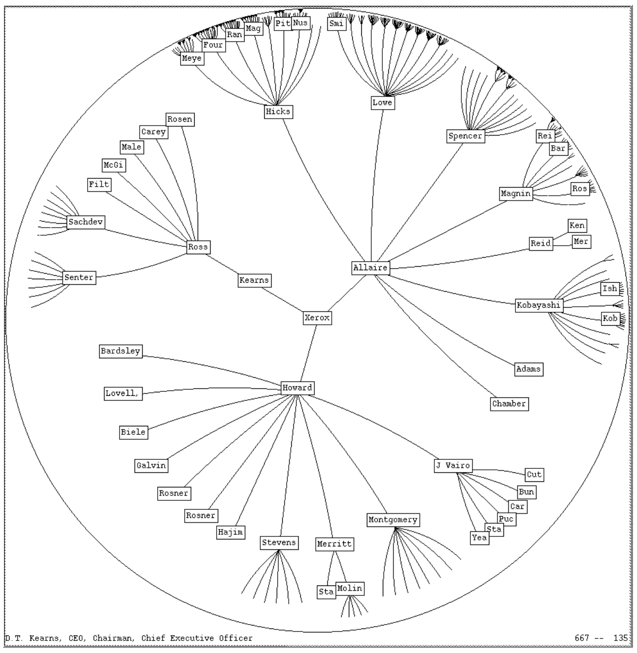


Figure 3.9: A hyperbolic layout. The nodes placed in the centre of the drawing circle are focused, towards the border the distances of the nodes become smaller. [Figure extracted from Proc. of CHI '95. Copyright by the Association of Computing Machinery, Inc.]

The hierarchy is laid out radially on an infinite hyperbolic plane. In hyperbolic space each node can allocate nearly the same amount of space for its children, independent of how deep they are in the tree. This layout is then mapped onto the unit disc with in euclidean space. Due to the mapping the unit circle is characterised by varying density. The central area has low density, therefore there is enough space for the tree. Towards to the border the density grows, which leads to the tree being very compact near the border. Nodes at higher levels are drawn more distant from the centre. Due to the growth of density of space, distant nodes and branches are smaller. Infinite space at the border can contain branches and nodes of any level. Thus the tree never actually reaches the edge of the disc (see Figure 3.9). A detailed description of the geometric background is given in Chapter 6.

The unit circle and the contained nodes and edges are scaled to screen size. The central area provides the focus, the area near the border shows the context of the tree. To browse the hierarchy, the tree is dragged around. The part which is in the centre of the drawing circle is focused. By this means the whole tree can be explored. Smooth animations are featured, which automatically drag the tree to bring the selected node into the focus. Further explanation of hyperbolic browsers can be found in Chapter 6.

3.10 3D Hyperbolic Browser

A *3d hyperbolic browser*, published by Munzner (1997) and called *H3*, is designed to draw large directed graphs in three-dimensional hyperbolic space. In a first step, the graph is transformed to a spanning tree. Links which are not part of the tree are selectively drawn by user request. The transformation algorithm is developed to detect underlying hierarchical structure of the graph, on which it builds the tree. For example, web sites are often designed hierarchically and are connected by hyperlinks afterwards.

The tree is then laid out in hyperbolic space on a hemisphere. The H3 algorithm lays out child nodes on the surface of a hemisphere in hyperbolic space. Then the nodes are projected to euclidean space, where each subtree has its own part of the global hemisphere. The child parts lie side by side with the

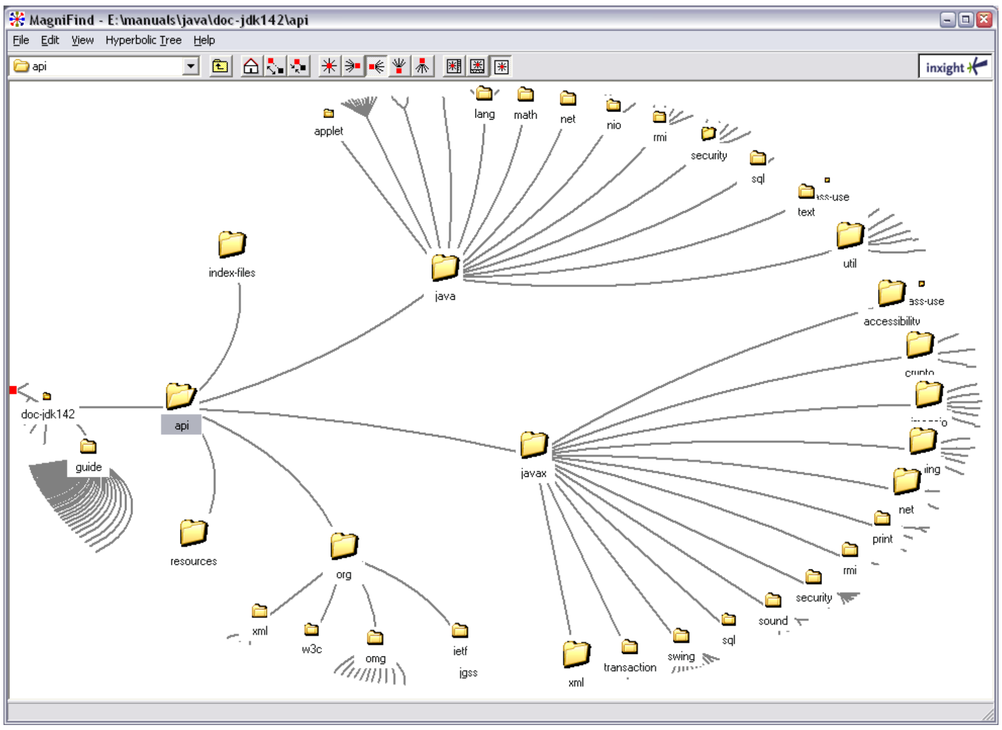


Figure 3.10: A hyperbolic view of a tree. The screenshot was taken from MagniFind, a free hyperbolic browser demo from InXight.

parent parts of the global hemisphere. In hyperbolic space, each parent node has the same space on a hemisphere, in euclidean space, the radii become smaller the deeper the nodes are in the tree.

The pole of the hemisphere is the focus where the nodes and links nearly have normal size. To browse the hierarchy, the interesting nodes are dragged to the pole. Double clicking on a node starts a smooth animation which automatically drags the clicked node to the focus (see Figure 3.11).

3.11 Cone Trees

The *Cone Tree*, introduced by Robertson et al. (1991), is similar to classic tree drawings, however, it is laid out in three-dimensional space. The tree is built top down with nodes drawn as cones. The root node is placed at the top in the centre, its children circularly below in the next layer. Each layer represents one level in the tree. This is recursively done for the whole tree. The body of each cone is shaded transparently to enable the user to see the whole tree (see Figure 3.12).

When a node is selected, the branches from the root to the selected node are smoothly rotated, so that every node on this path is in front and highlighted. The rotations of each substructure are done in parallel and are smoothly animated to maintain the user's orientation.

The Cone Tree method provides a perspective view on the three-dimensional tree. Lighting techniques and shadows are used to produce a realistic view, which reinforces a sense of spatiality. The selected node and its path are larger, brighter, and closer to the user. This provides a natural focus plus context effect, similar as described for the Perspective Wall (see Section 2.2). Screen Space is used very effectively because of the three-dimensional visualisation. Cone Trees utilise spatial depth to fill up the screen with information.

An alternate layout is the *Cam Tree*, which is horizontally oriented. This has the advantage, that

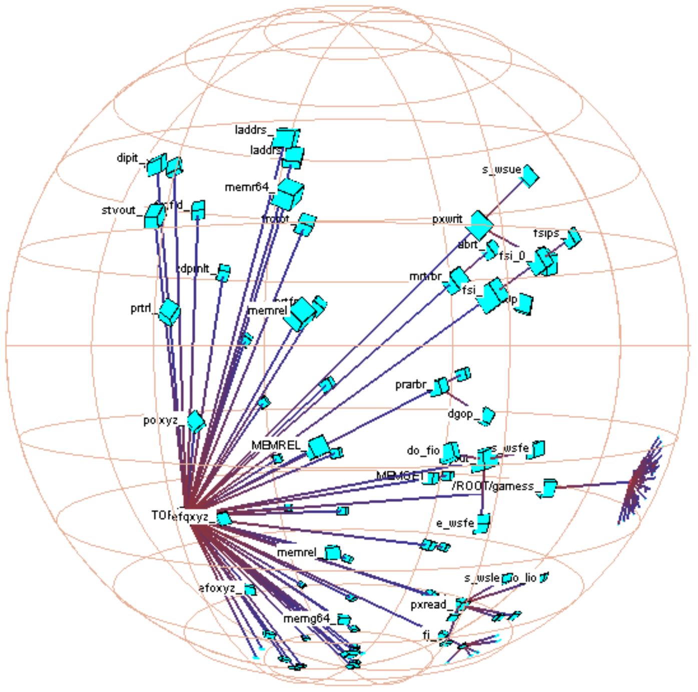


Figure 3.11: The 3d hyperbolic browser H3. A hierarchy is laid out in 3d hyperbolic space on a sphere.

labels fit better to the aspect ratio of the horizontal cones.

3.12 Cheops

Cheops, published in Beaudoin et al. (1996), is a compressed visualisation of a tree in order to deal with large and complex hierarchies. The nodes are represented as triangles and are laid out in a top-down manner, similar to the classic tree drawings. However, in contrast to them, the used display space is optimised by overlapping and overloading the drawn nodes. In this way the whole tree gains a triangular shape. Overloading of nodes results in many triangles which are ambiguous. This is resolved by selecting nodes (see Figure 3.13).

When a node is selected, the whole subtree is selected. The siblings and their subtrees of the selected node are disabled. The colours of the affected nodes are changed in order to emphasise the different states of the nodes. The colour coding includes selected nodes, children of selected nodes, non-selected nodes and their subtrees. In the new selection state, only nodes within the subtree of the selected node can be clicked by the mouse. In this way the user can navigate through the hierarchy, from top to bottom.

Colour coding is also used to distinguish between single and ambiguous nodes. Large hierarchies can be visualised by the Cheops method, however it is difficult to gain an overview of the structure of the tree because of the ambiguous node shapes.

3.13 Information Slices

Information Slices is a space-filling, radial, and inclusive approach (Andrews and Heidegger, 1998). One or more semi-circular discs are used to visualise large hierarchies. Each disc contains a user-configurable

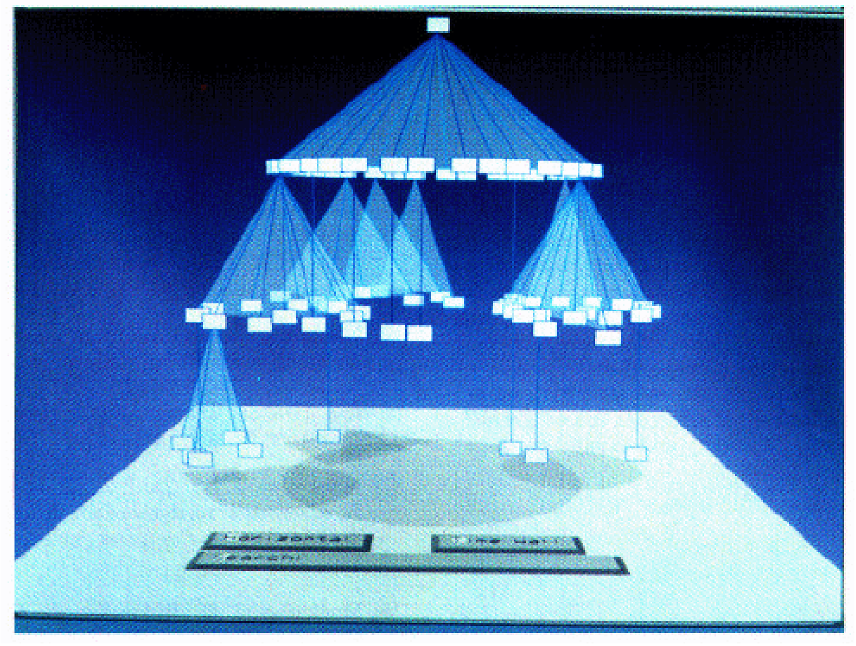
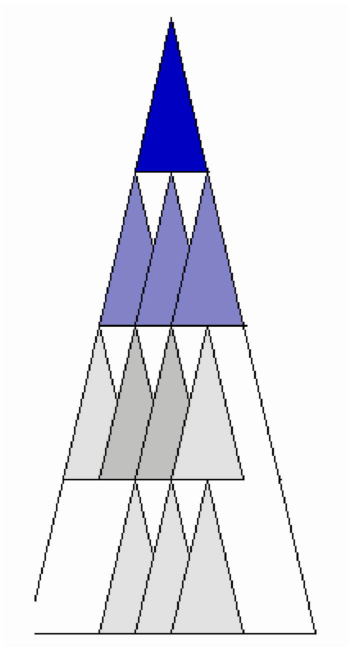
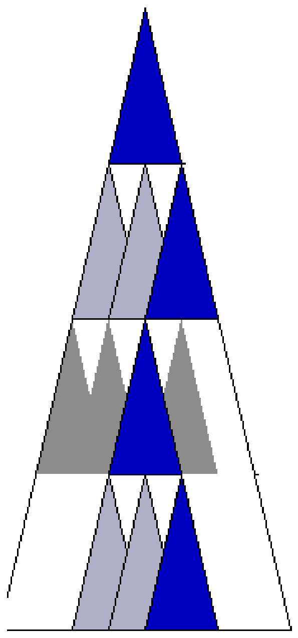


Figure 3.12: The Cone Tree. A hierarchy is visualised in 3d space using cones to represent inner nodes. When a node is selected, the respective branches are rotated, so that they are in front. [Figure extracted from Proc. of CHI '91. Copyright by the Association of Computing Machinery, Inc.]



(a) A full tree with a selected root node. The bright grey triangles are single nodes, the grey triangles are overloaded nodes.



(b) A tree with a selected node at the second level. The dark grey triangles are disabled, the blue ones show the selection path.

Figure 3.13: Two Cheops visualisations with different selection states.

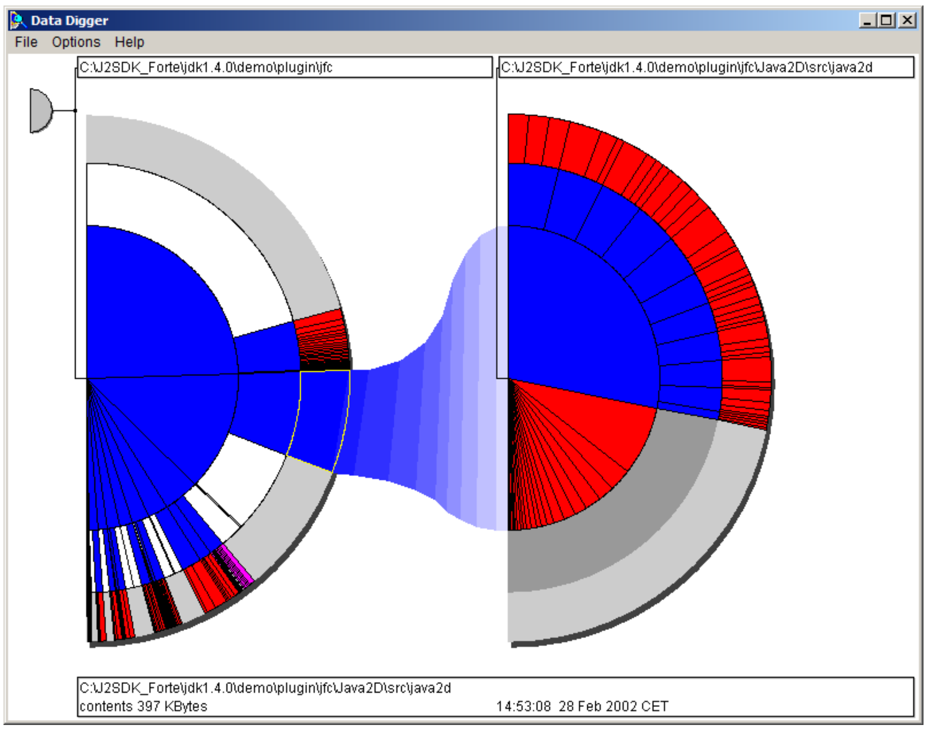

Data Digger
File Options Help
C:\J2SDK_Forte\jdk1.4.0\demo\plugin\jfc
C:\J2SDK_Forte\jdk1.4.0\demo\plugin\jfc\Java2D\src\java2d
C:\J2SDK_Forte\jdk1.4.0\demo\plugin\jfc\Java2D\src\java2d
contents 397 KBytes
14:53:08 28 Feb 2002 CET

Figure 3.14: Information Slices. A tree is visualised on two semi-circular discs. The right disc shows a subtree of a selected inner node in the left one. [Image used with kind permission of Keith Andrews, Graz University of Technology.]

number of levels of the tree. At each level, the nodes are fanned out in the available space on the disc. As described for Treemaps, the space for each node depends on its weight. Deep trees are cascaded on multiple discs, with subtrees visualised in further discs (see Figure 3.14).

Navigating through a tree is done by clicking on inner nodes. When a user expands an inner node, it is fanned out on the adjacent disc. Therefore, deep trees can easily and rapidly be explored. However, broad hierarchies can result in dense, thin slices.

3.14 Sunburst

Building on the method of Information Slices, an approach called *Sunburst* was published in Stasko and Zhang (2000). Sunburst uses a full disc for the layout and features different kinds of fan-out of subtrees. Size, angle, and colour of the slices correspond to the attributes of the represented item. Three interaction techniques which provide flexible browsing through the hierarchy are implemented.

When a node is selected, the *angular detail* method causes the entire hierarchy to shrink and to move to the boundary. The selected item expands outward from the overview visualisation. The *detail outside* method shrinks the hierarchy in the centre and the selected item expands to be a new complete ring around the overview. The detail inside method pushes the overview outward and expands the selected item in the centre of the ring. All three methods can be used alternately to focus a node. The transitions are smoothly animated.

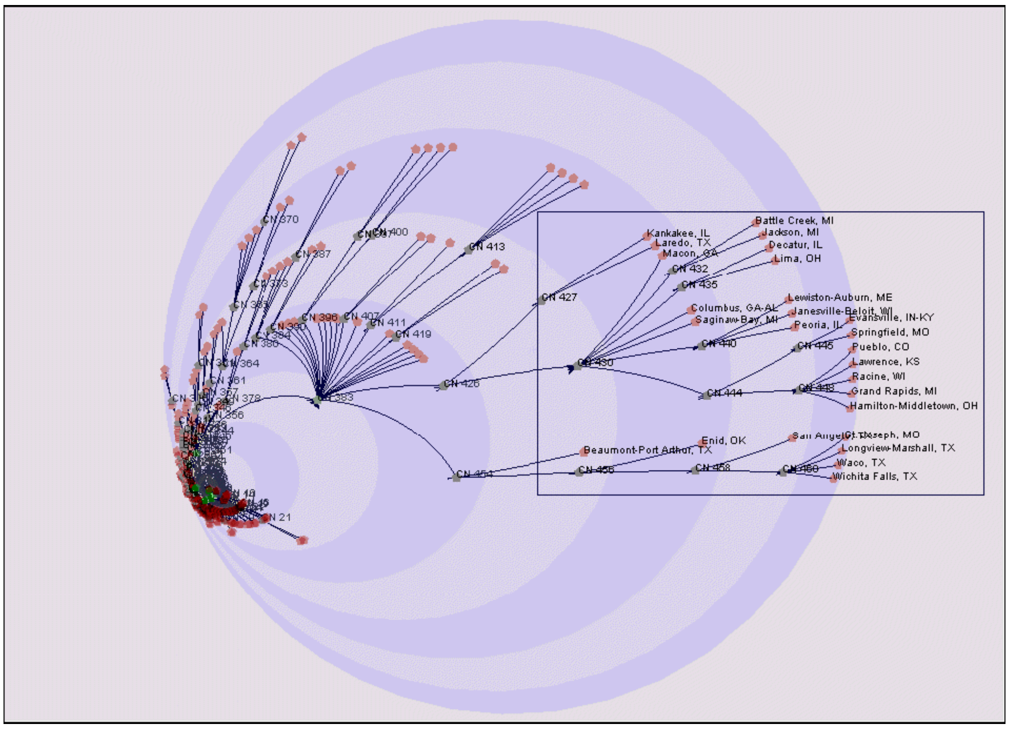


Figure 3.15: Magic Eye View. The focus is on the right side of the projection circle. [Figure extracted from Proc. of NPIV '99. Copyright by the Association of Computing Machinery, Inc.]

3.15 Magic Eye View

The Magic Eye View (Bürger, 1999; Kreuseler and Schumann, 1999) is a focus plus context hierarchy browser which uses a node-link representation of the tree. The hierarchy is laid out radially using the Walker layout algorithm (see Section 3.3). Then the radial layout is mapped onto the surface of hemisphere using the polar circle length of the nodes as spherical angle. The tree on the surface is projected onto the equatorial circle of the hemisphere. There is a special point called projection centre, which lies on the equatorial circle and which determines the focus area. The tree is moved on the hemisphere using the angles of the intersection lines connecting the projection centre with the node points on the hemisphere. If the projection centre does not lie in the point of origin, distortion is achieved. This kind of projection achieves a grater magnification in the focus area than a simple orthogonal projection. The technique is described in detail in Chapter 7.

To explore the hierarchy the user can move the projection centre by dragging the mouse. The focus area is always on the opposite side of the projection centre to the point of origin. If the user moves the projection centre to the border, the part of the tree at a deeper level is magnified, which lies graphically on the other side of the circle. Figure 3.15 shows a magnification of a particular part of the tree.

3.16 File System Navigator

The File System Navigator (FSN) (Tesler and Strasnick, 1992) is a three-dimensional directory browser, which was developed for the Silicon Graphics IRIX operating system. A landscape metaphor is used for the tree layout and the navigation through the tree. The layout is done by a conventional algorithm. Directories are displayed as pedestals. FSN makes a distinction between child nodes which are files and

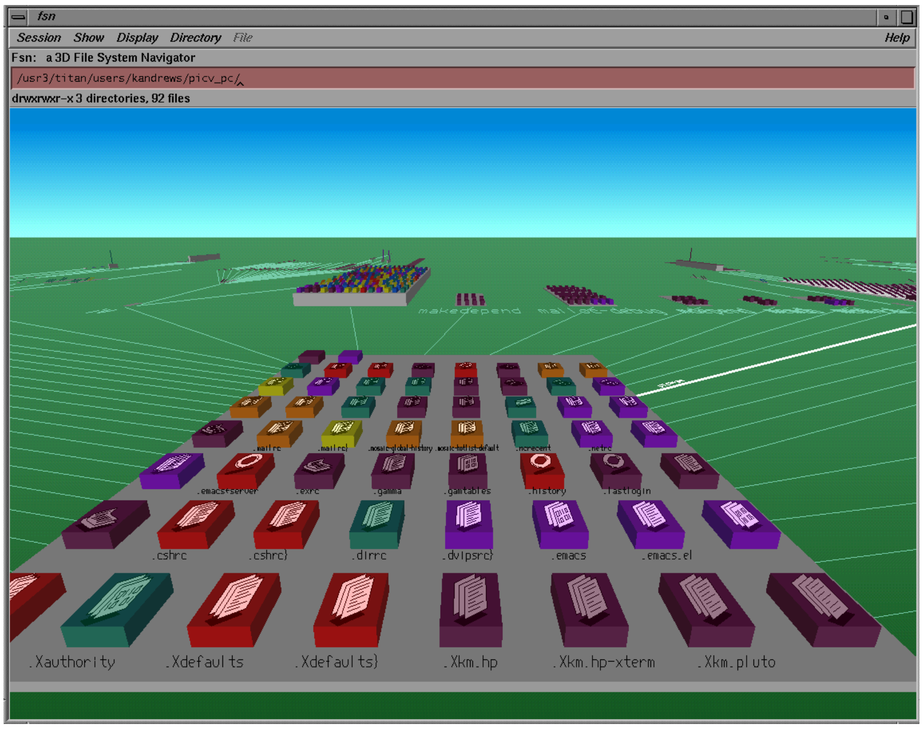


Figure 3.16: A landscape metaphor is employed by the File System Navigator (FSN). A directory with its contained files is shown, its subtrees recede into the background. [Image used with kind permission of Keith Andrews, Graz University of Technology.]

child nodes which are directories. The files are placed atop of the parent pedestals and are visualised as columns. The directories are laid out spatially, similar to classic tree drawings, and are connected by lines with their parent node (see Figure 3.16).

The height of a column represents the size of the corresponding file, the colour symbolises the creation date. The file type is expressed by an image on top of its column and the file name is drawn in front of its column. In this manner four attributes of a file are visually illustrated. The height of a pedestal depends on the number of the contained files, the directory names are drawn in front of each pedestal.

Exploring the hierarchy is done by virtually flying through the landscape. The user has the possibility to freely fly around using the mouse. Selected nodes are highlighted with a spotlight and an animated zoom is provided by the system. Documents are opened by the appropriate viewer with a double-click. Other 3d navigational features are supported, such as remembering viewpoints, navigation undos, and a 3d overview window. The user can easily gain an overview of the structural information of the hierarchy including some attributes of the files. A natural context plus focus effect is achieved by the 3d perspective view. Navigating to a point of interest enlarges these directories and files, distant directories recede to the background.

A similar visualisation technique is used by the Harmony Information Landscape (Andrews et al., 1996; Eyl, 1995), which is part of Harmony, the former hypermedia client of the Hyperwave web server. As well as displaying hierarchical structure in the landscape plane, hyperlink relationship is superimposed on the spatial layout. Texturing of the 3d elements is supported for an appleasing visualisation of document types and content.

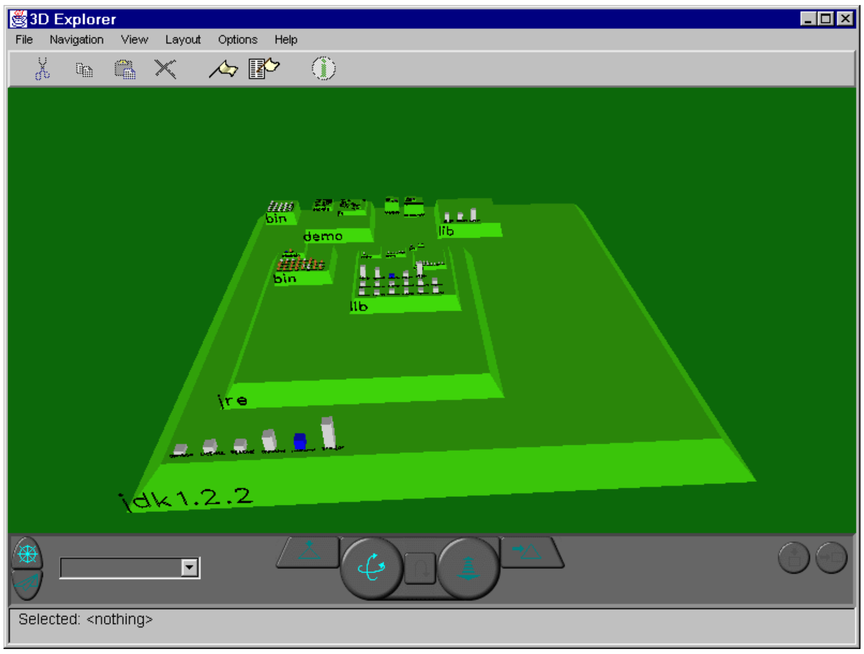


Figure 3.17: Information Pyramids. A hierarchy is visualised as a pyramid model in 3d space. The pedestals represent the inner nodes, the columns on them are leaf nodes. [Image used with kind permission of Keith Andrews, Graz University of Technology.]

3.17 Information Pyramids

Information Pyramids (Andrews, 2002) is a technique to model hierarchical information in a three-dimensional pyramid object. Nodes are visualised as pedestals, similar to the File System Navigator (see Section 3.16). However, in contrast to FSN, all nodes are placed atop a parent node. Each inner node contains all its children, leaves are final nodes. Thus a compact pyramid visualisation is created, where the plateaus represent the levels of the hierarchy (see Figure 3.17). Obviously, the size of the nodes shrink, when the pyramid grows. Leaves nodes are marked with icons, which represent additional information such as the node type. Exploring the hierarchy is done by navigating through 3d space.

The first implementation of the information pyramids approach was the *3D Explorer*, published in Wolte (1998) and shown in Figure 3.17. The visual representation of nodes regarding their size, colour coding and layout is similar to the FSN. A rich set of navigational aids are featured for exploring the file system hierarchy. The user can freely fly through the 3d scene (explore mode) or rotate and scale the pyramid (examine mode). Navigating closely to a particular node means zooming to a point of interest in the file system hierarchy.

Information pyramids provide a good overview of the hierarchy and are wellsuited to exploring both broad and deep hierarchies. However, users often have problems to freely navigate through the scene. A further development resulted in the *Java Pyramids Explorer* (Welz, 1999). The extensive 3d navigational facilities were replaced by three sliders to control the navigation. One slider is used for rotating around the x-axis, one for rotating around the z-axis, and one for zooming.

3.18 Botanical Visualisation

The *Botanical Visualisation* (Kleiberg et al., 2001) uses a three-dimensional tree metaphor to represent abstract tree information. The method is based on the observation that the perception of branches, leaves,

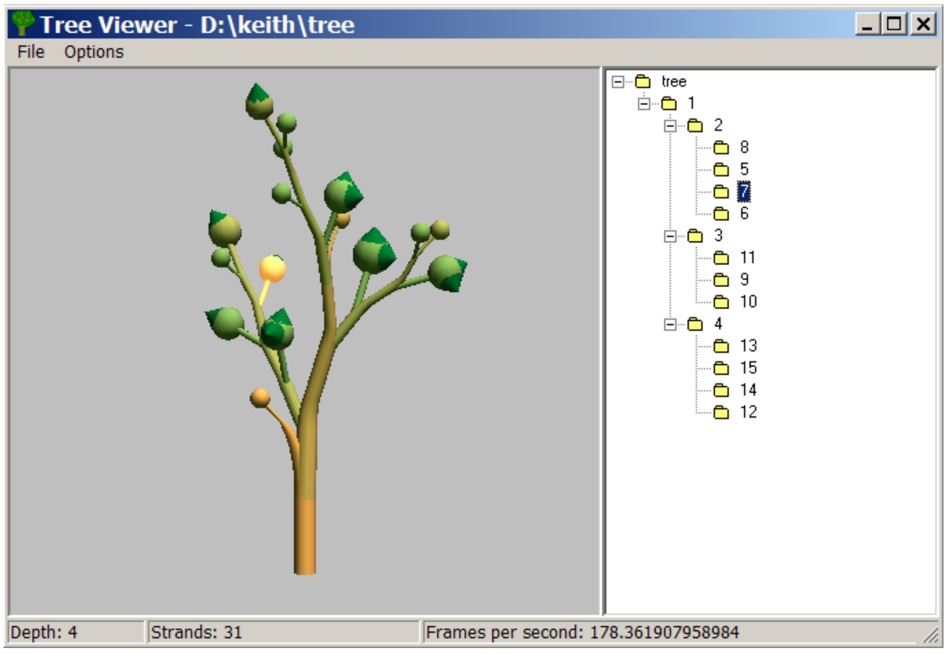

Tree Viewer - D:\keith\tree
File Options
tree
1
2
8
5
7
6
3
11
9
10
4
13
15
14
12
Depth: 4
Strands: 31
Frames per second: 178.361907958984

Figure 3.18: Botanical View. An abstract tree is modelled as a botanical tree. The fruits represent the leaves, the branches represent the hierarchical structure. [Image used with kind permission of Keith Andrews, Graz University of Technology.]

and their arrangement in a botanical tree is easy for humans, even if the number of elements is great and the structure complex. Inner nodes are represented as branches, their children as fruit hanging from them. To support perception and to improve aesthetics, continuing branches are emphasised, long branches are contracted, and sets of leaves are shown as fruit (see Figure 3.18).

Since modelling of botanical trees has been intensively studied by the computer graphics community, there are many methods available The strand model of Holton (1994) is used to generate the 3d model from the hierarchical information structure. This method is convenient, because the strands enable a simple mapping of the size of elements to the radii of branches.

Chapter 4

The Hierarchical Visualisation System (HVS)

The *Hierarchical Visualisation System* (HVS), introduced by Putz (2005), is an extensible framework for integrating and synchronising different types of tree visualisations. The framework provides key infrastructure such as hierarchical data import and management, which various visualisations can then build upon.

This chapter describes the main features of HVS, followed by the software design and its components. The term *framework* is used for all parts of HVS except the pluggable components, which are the visualisations and the data source modules.

4.1 Features of HVS

HVS is designed as a visualisations toolkit, which provides a multiple view environment. This is a benefit for both users and developers. Hierarchical information is visualised in various manners, so users see complementary aspects of data. For better orientation the views are synchronised. Developers of visualisations can focus on visualisation methods and do not have to attend to the underlying data structure and operations on it.

4.1.1 Hierarchical Data Model

Unlike other visualisation toolkits, HVS focuses on hierarchical data. The internal data model is designed to process hierarchies in the sense of directed acyclic graphs (see Chapter 3). The terms *hierarchy* and *tree* are used synonymously in this discussion to mean DAGs. Figure 4.1 shows an illustration of the HVS data model.

The data model consists of both structural and content information. The structural information is represented by inner nodes, which contain parent-child relationship information. Content information is chiefly contained in leaf nodes as sets of attributes. There are various attributes, which can be divided into three types:

- *textual:* node name, author, title, subject, and keywords.
- *numerical:* number of pages, document type (enumeration number).
- *chronological:* creation and modification date.

Importing tree data into the HVS data model is done by pluggable modules. HVS provides an interface for data source modules, which import data from any source. Currently there are two data

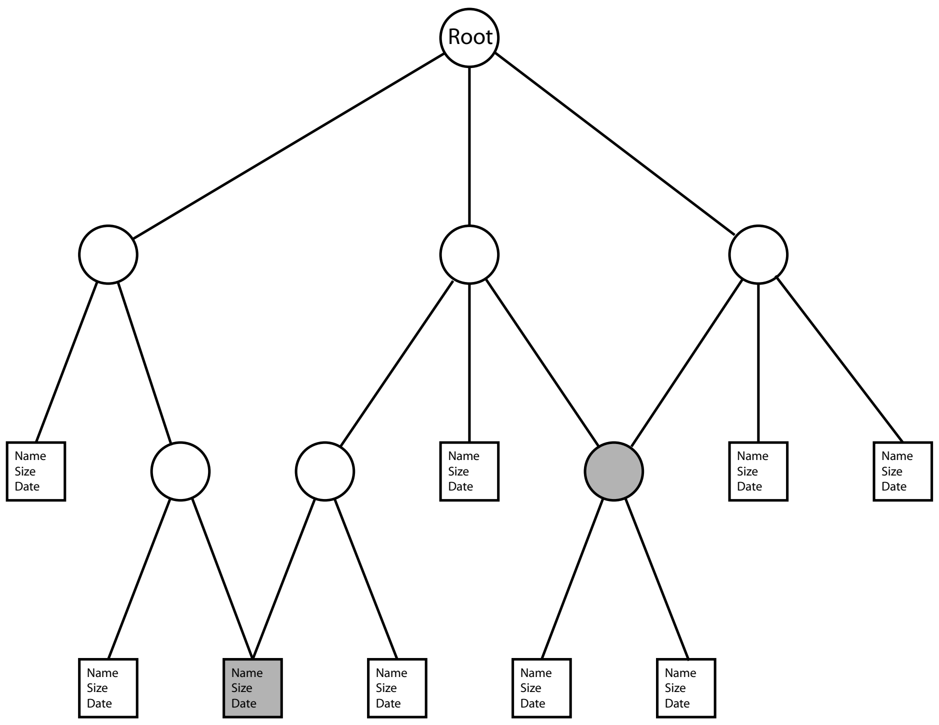


Figure 4.1: The HVS Data Model. The diagram shows an example hierarchy (actually a directed acyclic graph). The leaf nodes illustrated as rectangles contain attribute data. The grey nodes have two parents and therefore two paths to the root node.

source modules, one for reading data from the local file system, the other to read hierarchical data from XML files. However, pluggable modules can be developed and added, for example, a module which gathers hierarchical data from web content.

The hierarchical data can be modified by the user. The visualisations provide a user interface to insert, rename, or remove tree nodes. A flag controls whether only the internal data structure is modified, or whether changes are propagated to the source hierarchy.

4.1.2 Synchronised Views

HVS provides different types of views of the same data structure (see Figure 4.2). The views are opened and closed by user request. Through a plug-in interface, views can be integrated into HVS independent of the visualisation method. It depends on each view how information is visualised and how events from the framework are interpreted. The framework only defines the interactions at a logical level.

A distinction of three synchronisation types can be made with respect to the initiation of an event. The first synchronisation type concerns events which are initiated by the user in a visualisation panel as a consequence of navigational actions. To keep the user oriented, the views are synchronised by the framework. If the user performs an action in a visualisation, the framework and the other visualisations are notified. The following actions are included by this type:

- *selection:* changing of selection states of nodes.
- *expansion:* expanding and collapsing nodes.
- *navigation:* setting a focus node, scrolling, and maximising.

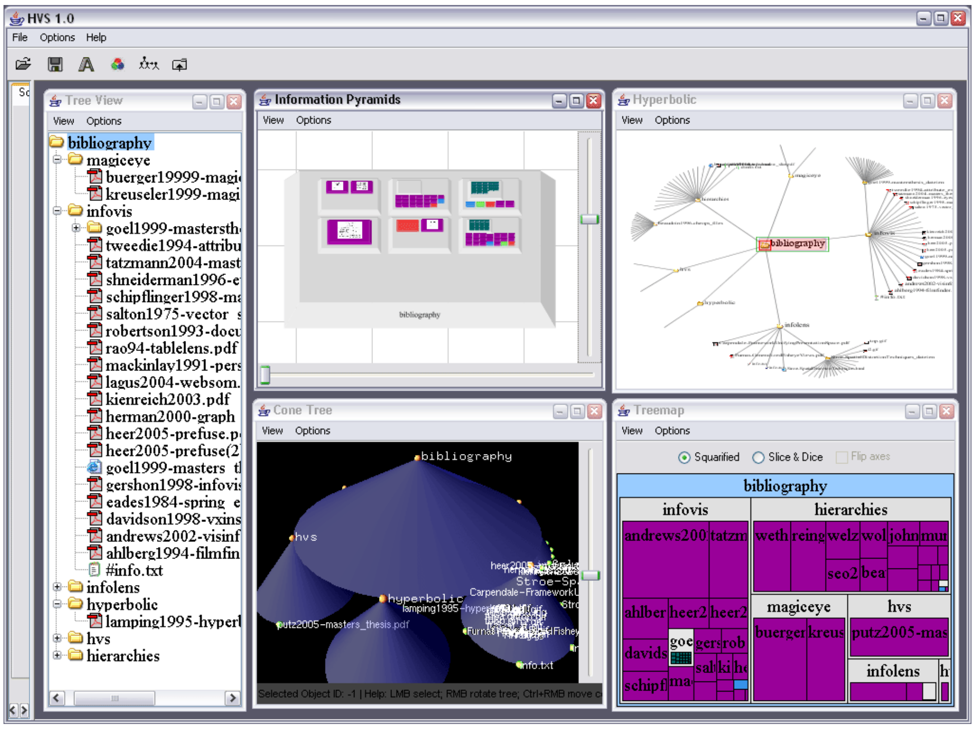


Figure 4.2: HVS Visualisations. Five different visualisations of the same hierarchical data are opened and synchronised by the HVS framework.

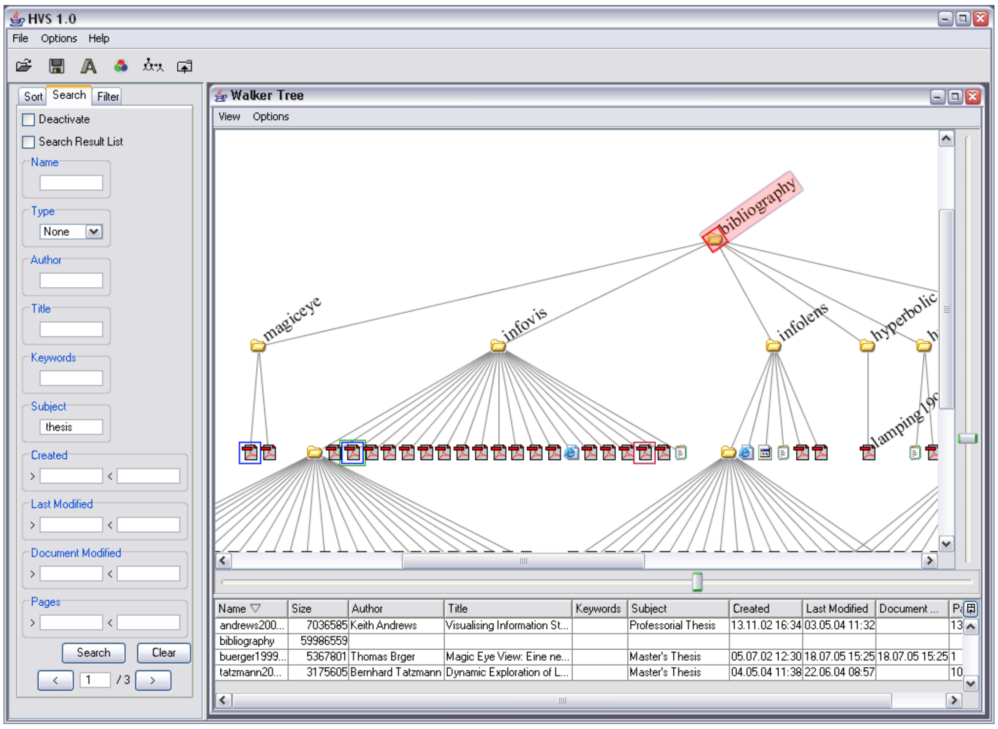
HVS provides selection and navigation state processing. If nodes are selected in a view, a message is sent to the framework containing the new selections. The framework updates its internal selection state and sends this to all visualisations. An analogous behaviour is applied to focused nodes, except that only one node can be focused, but an arbitrary number of nodes can be selected.

The second synchronisation type concerns actions of the user made in the HVS framework to search, filter, and modify the hierarchical data. Due to these actions, synchronisation events are sent to all visualisations by the HVS framework. The following events are assigned to this synchronisation type:

- *hierarchy data:* setting a new explorable tree, continuously reading of tree data by a data source thread, adding, removing, or renaming a node.
- *search result:* displaying the result of a search performed in the framework.
- *filtering:* filtering out data by user request.
- *render settings:* update render settings such as colours.

The third synchronisation type is related to individual views controlling the synchronisation behaviour of single views. The user can set three synchronisation modes for each view independently from the others:

- *Synchronised versus Independent:*
  To explore a hierarchy without changing other views, the synchronisation of a visualisation can be turned off.
- *Overview versus Detailed View:*
  In detailed view mode, only the selected nodes and their paths to the root are visualised.



Name	Size	Author	Title	Keywords	Subject	Created	Last Modified	Document ...	Pa
andrews200...	7036585	Keith Andrews	Visualising Information St...		Professorial Thesis	13.11.02 16:34	03.05.04 11:32		13
bibliography	59986559								
buerger1999...	5367801	Thomas Brger	Magic Eye View: Eine ne...		Master's Thesis	05.07.02 12:30	18.07.05 15:25	18.07.05 15:25	1
tatzmann20...	3175605	Bernhard Tatzmann	Dynamic Exploration of L...		Master's Thesis	04.05.04 11:38	22.06.04 08:57		10

Figure 4.3: The HVS search and property panels. A search is done for the subject word ‘thesis’, the result is highlighted by blue and red rectangles around the nodes. The property panel displays the currently selected nodes in a table, which are the root node and the three search result nodes.

- *Show Documents versus Hide Documents:*
  Documents can be hidden, so that only the structure is shown.

The framework provides a *properties panel* for each visualisation. This panel consists of a table, in which all currently selected nodes with their attributes are listed (see Figure 4.3). The synchronisation of the selection state and this panel is done by the synchronisation mechanism of the framework. Since multiple selections of nodes are possible, items and attributes can easily be compared in this panel. This behaviour is a detail-on-demand characteristic of HVS.

4.1.3 Searching

Textual search functionality is provided by the HVS framework. Since the data model is capable of processing items and attributes, search can be performed on various attribute fields. The HVS search panel with fields for all attributes allows the input of the search query by the user (see Figure 4.3).

The framework executes the query and sends the result to the visualisations. It depends on the views how they visualise the search result, usually they draw coloured bounding boxes around the found nodes. Among these nodes there is one node which is marked. This node is usually visualised differently, for example with a different colour. In this way, the user can browse through the result list by stepping from one node to the next using the next and previous buttons in the search panel. Additionally, an optional search result window is provided by the framework, where the result list is presented as a table. The items are listed there together with their attributes.

4.1.4 Filtering and Sorting

Filtering is an important task in information visualisation systems. HVS provides textual filtering, which is similar to textual searching. The user can input the filter criteria in the filter panel with attribute fields. A filter result list of found nodes is created internally analogously to the search result list.

According to this result list the data model is modified. All found nodes and the nodes on their paths to the root are contained in the new data model. Then the data model instructs the visualisations to update their views based on the new filtered hierarchy data.

HVS supports ordered trees, where the order is determined by the name, the size, or the type in either ascending or descending order. The user can configure the order type and direction in the sort panel beside the visualisations. The effect can be seen in the visualisations, which recreate their tree layout according to the current sort order.

4.1.5 Rendering Control

The framework provides a user interface for various rendering settings. A colour dialogue defines colours for various graphical elements in the views, such as nodes of different types, the background, highlighting of selected nodes, or nodes which are in the search result. A font dialogue is implemented to choose the label font, size, and type globally for all visualisations. Icons for different item types can be chosen in a list box.

4.2 Software Design of HVS

4.2.1 Software Architecture

HVS has an object-oriented and open software architecture, which supports extension by visualisations and data source modules. The implementation is in Java (Java, 2005). The software architecture is based on the model-view-controller (MVC) design pattern, which is often used by graphical user interfaces, such as the Java Swing components. A diagram of the software architecture can be seen in Figure 4.4. This design pattern is a good solution for systems which implement multiple views of the same data. A synchronisation of views and data and a coordination of the user input with views and data is included in this design concept. The modules shown in Figure 4.4 can be assigned to the MVC pattern as follows:

- *Model:* a model of the underlying data.
  The according component in HVS is the *data model*.
- *View:* the views presented to the user.
  The corresponding modules in HVS are the *visualisations*, the *search result window*, and the windows and dialogues of the *application*.
- *Controller:* the module which coordinates the user input, the views and the data model.
  The module in HVS with this functionality is also named *controller*.

4.2.2 Application Module

The application module starts HVS, which includes several tasks, such as creating instances of classes and initialising configurable user settings. Furthermore it manages the basic event handling, which does not comprise the synchronisation of the views with the data model. Examples are opening dialogue windows, starting data import, or choosing a colour.

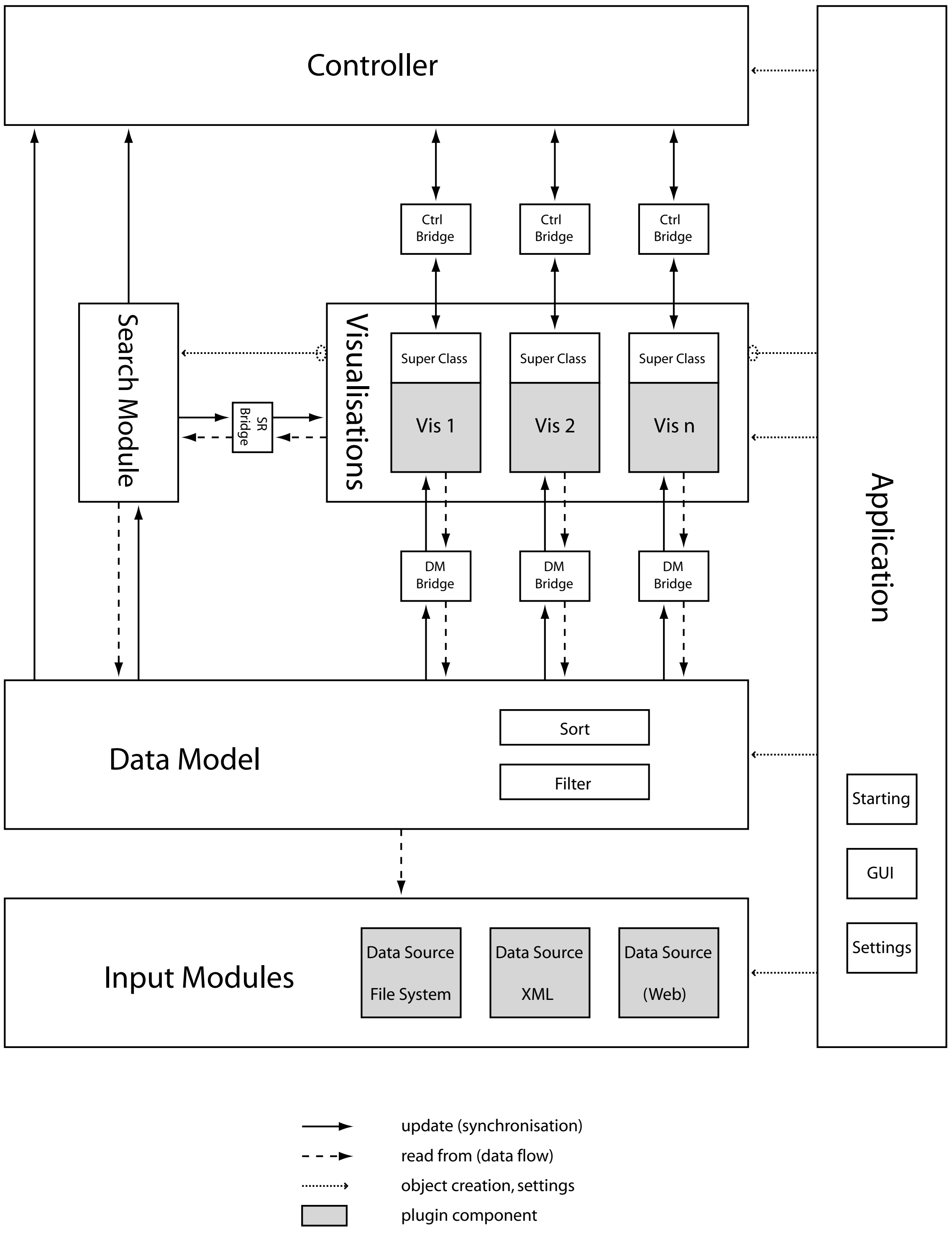


Figure 4.4: The software architecture of HVS containing the modules and plug-in components.

An important task is plug-in management. There are two types of plug-in modules, visualisations and data source modules. Both kinds must be inherited from specific super classes in order to provide an interface for the framework (see below in this Section). The application loads them from a particular directory, where they reside in common with plug-in configuration files. A configuration file for a plug-in module determines its name, the class that has to be started, and optionally any software libraries.

The application also provides the dialogues for rendering settings, which are a colour and colour scheme chooser, a font chooser, and an item selection box. A configuration comprising a data source and some visualisations can be saved and reloaded in a later HVS session.

4.2.3 Input Modules

Input modules import hierarchical data from a specific source, such as the local file system or an XML file, and create an internal data hierarchy from these data. Furthermore, they gather attributes from the data and include them in the internal data model. Since manipulating the data by user interaction is a design goal of HVS, these data can be modified after import.

To enable the framework to access these data, the input modules have to implement some abstract classes (in Java they are called *interfaces*). These are defined in the package `iicm.hvs.inputfactory`. The most important classes and method declarations include:

- `DataSource`
  This class defines access methods to the meta information about the data source. For example, the attribute keys or the available document types.
- `Node`
  A node class defines the access methods for both the inner nodes and the leaf nodes, such as a request for its name or its metadata.
- `Collection`
  A collection is inherited from a node and represents an inner node.
- `Document`
  A collection is inherited from a node and represents a leaf node.
- `InputDataModel`
  This class defines the access to hierarchical data, for example to get parents or children of a node.
- `MutableInputDataModel`
  This interface is needed for the declaration of methods which modify the tree data.
- `InputFactory`
  In order to create new instances of collections and documents, suitable methods are defined in this class.

4.2.4 Data Model Module

The data model is a wrapper class for the hierarchy data created by an input module. Thus it reads the data from the underlying module and provides them to the visualisations and the search engine. The data model acts as the *model* in the MVC design pattern.

In addition to data forwarding, filtering, and sorting are also processed in the data model. Following an user request, a command to filer or sort by the certain attributes comes from the application. The data model applies filtering and sorting on the data retrieved from the input module. Then the modified tree data are provided for the other modules. Filtering and searching are described in Section 4.1.

The interface to read hierarchy data is defined in the package `iicm.hvs.datamodel` in the class `DataModel`. The classes `Node`, `Collection`, and `Document` are equivalent to those of the input modules. The method declarations of the class *DataModel* are:

- `getRoot ()`
  The root node of the hierarchy is returned.
- `getSubCollections (Collection parent)`
  The inner nodes of the given collection are returned.
- `getDocuments (Collection parent)`
  The leaves of the given collection are returned.
- `getChildren (Collection parent)`
  The inner nodes and leaves of the given collection are returned.
- `getTreePaths (Node node)`
  The paths of the given node to the root node are returned.

An interface to modify the hierarchy data is provided in the class `MutableDataModel`, which is inherited by `DataModel`. When data have changed a notification is sent to the controller, the visualisations, and the search engine. This is done with the help of a listener mechanism, which sends the events to the registered components.

4.2.5 Search Module

A search query is entered by the user and is sent from the application to the search module. A search mechanism traverses the hierarchy in order to find nodes which match the search query. The found nodes are added to the search result list, which is also a *model* component in the sense of the MVC design pattern. If a result is complete, events are sent to the listening modules, analogously to the data model.

The class `SearchResult` in the package `iicm.hvs.search` provides the interface for the visualisations and the search result window. The views require information for each node, if it is in the search result list, and if it is the currently selected search result. Two method declarations are important therefore:

- `isSearchResult (Node node)`
  whether the node is in the search result list is returned.
- `isSelectedSearchResult (Node node)`
  whether the node is the currently selected search result is returned.

4.2.6 Visualisations

A visualisation is a pluggable module which visualises the tree data. It consists of two parts, the framework part and the plug-in part. The framework part provides the interface to the controller and the data model, provides the properties panel at the bottom of the window, and supports access to rendering and visualisation properties. The plug-in part consists of the visualisation of the hierarchy and the implementation of the interface.

The properties panel (see Figure 4.3) shows the currently selected items together with their attributes in a table. Synchronisation between the panel and the visualisation is done automatically by the synchronisation mechanism of the framework.

To implement the interface of the framework, the abstract superclass `Visualization` in the package *`iicm.hvs.visualization`* must be inherited. Furthermore it has to implement the following interfaces with their method declarations to receive synchronisation events:

- *`DataModelListener:`*
  If there are changes in the data model, an appropriate method declared in the DataModelListener is called.

- *`ControllerListener:`*
  This listener comprises the method declarations for the synchronisation of the selection and navigation state.

- *`SearchResultListener:`*
  When the search result has changed, a notification is sent to a method contained in this listener.

- *`ExpansionListener:`*
  If an inner node is expanded or collapsed by the user in one visualisation, a message is sent to all visualisations implementing this listener.

To send synchronisation events to the framework, there are a set of methods implemented in the `Visualization` class, which can be used by the plug-in part:

- *`fireFocusChanged (FocusEvent event)`*
  If the navigation sate has changed in the visualisation, this method is called to notify the other visualisations about the new navigation state.

- *`fireSelectionChanged (SelectionEvent event)`*
  If the selection state has changed, the new state is sent to the other visualisations by using this method.

- *`fireCollectionExpanded (ExpansionEvent event)`*
  This method provides the notification of an expansion of an inner node.

- *`fireCollectionCollapsed (ExpansionEvent event)`*
  This method provides the notification of a collapsing of an inner node.

To have access to rendering and visualisation properties, there are several methods implemented in the class `VisualizationProperties`.

- *`getColorForDocument (Document document)`*
  This method is used to get the colour of a document according to the type of the document.

- *`getImageForDocument (Document document)`*
  This method returns a thumbnail of the specified document, if available.

- *`getSelectionColor ()`*
  The colour for the visual representation of a selected node is returned.

- *`getSearchResultColor ()`*
  If a node is within the search result, it is drawn in the colour returned by this method.

- *`getSearchResultSelectionColor ()`*
  If a node is the currently selected node within a search result, this colour is used for the visualising.

4.2.7 Controller

The synchronisation mechanism of the visualisations is implemented in the controller module. Events which the controller receives from a visualisation are forwarded to all other visualisations. The events managed by the controller are related to user interactions and can be summarised as follows:

- *Selection:* One or more nodes can be selected in a visualisation. The selection state is sent by the active visualisation to the controller, which forwards the state to the other views.
- *Navigation:* Navigation events are related to a focused node. Various navigational actions are included, such as bringing the node into the visible area or maximising the node. This behaviour guarantees analogous views in different visualisations.
- *Expansion:* If an inner node is expanded or collapsed in a visualisation, then all visualisations are informed. However, not all of them support expanding and collapsing, for example graph-based visualisations, which always provide a view in which all nodes are expanded.

4.2.8 Synchronisation Bridges

The synchronisation bridges mechanism implements the third synchronisation type described in Section 4.1.2. There are three synchronisation modes, which can be configured by the user for each view. To implement this functionally there are three types of bridges, which act as a filter between the visualisations and the other modules (see Figure 4.4). Each view has its own instances of these bridges.

- *Controller Bridge:*
  The controller bridge can prevent the communication between a visualisation and the controller. This is used for the *Independent* mode.
- *Data Model Bridge:*
  The data model bridge can filter out nodes for the *Detailed View* mode and the *Hide Documents* mode.
- *Search Result Bride:*
  The search result bridge filter out nodes from the search result for the *Detailed View* mode and the *Hide Documents* mode.

Chapter 5

Adding a Hierarchy Browser: The Walker Layout Browser

The previous chapter (Chapter 4) discussed the Hierarchical Visualisation System and how pluggable visualisations can be integrated. Using the example of the Waker layout browser, the integration of a visualisation into HVS is demonstrated. This browser is designed to visualise the classic tree drawing laid out by the Walker algorithm (see Section 3.3).

5.1 Walker Layout Technique

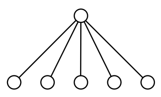
The Walker layout browser is used as a basic implementation of a HVS visualisation by the browsers discussed in the next chapters. The Magic Eye browser (see Chapter 7) and the InfoLens (see Chapter 8) are based on the Walker tree layout, which is further manipulated by their layout techniques. All the other browsers implemented the integration functionality in the same way.

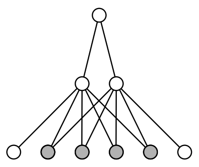
An overview of classic tree drawings is given in Section 3.3. This browser implemented the algorithm from Buchheim et al. (2002), which is an improved Walker layout algorithm. It works in a similar way and produces the same result, but in contrast to the original Walker algorithm (Walker II, 1990), it works in linear time.

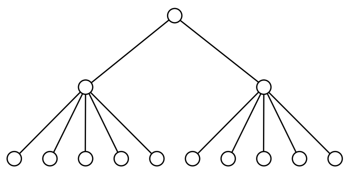
To outline the concept of the Walker layout algorithm, it can be split into three parts. In the first part, child nodes are traversed from left to right and placed at an arbitrary x-coordinate and at a y-coordinate given by their level. A fixed distance value defines the space between two neighbour child nodes. Then the parent node is placed centrally above its children (see Figure 5.1(a)). This procedure is done recursively for all nodes of the tree. In this way, a preliminary tree layout is established, which may overlapping in this phase (see Figure 5.1(b)). Each subtree is laid out independently from its position in the tree, which is a major requirement in Reingold and Tilford (1981).

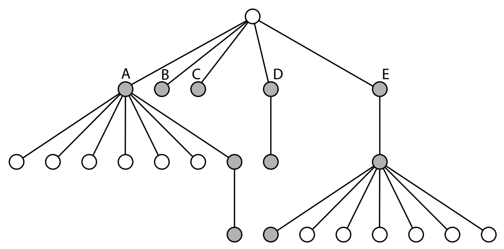
The second part of the algorithm deals with the problem of overlapping. Each subtree is shifted to the right so that it is placed as close as possible to the right contour of the left neighbour subtree. The minimum distance of two neighbour subtrees is the same fixed distance value used for nodes above. The parent of the shifted subtrees again is placed centrally above its direct children again (see Figure 5.1(c)).

At this point the method is the same as the classic Reingold-Tilford algorithm, except, that it is extended to trees of unbounded degree. However, because of the higher degree a problem arises which does not occur in binary trees. Small subtrees between larger ones are piled to the left and a gap between them and the right larger subtree emerges. In Figure 5.1(d) the subtrees B, C, and D are piled to the left, and a gap emerges between subtree D and E. This causes also a violation of the Reingold-Tilford requirement that the layout of subtrees should be independent of their location.

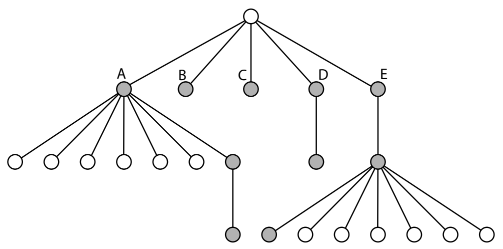

(a) The layout of a subtree.


(b) The preliminary layout of a tree.


(c) The final layout with shifted subtrees.



(d) A layout with shifted subtrees and a gap. The grey nodes indicate the contours of subtrees.



(e) A Walker layout with small subtrees, which are spaced out. The grey nodes indicate the contours of subtrees.

Figure 5.1: The Walker Layout Method shown in five steps.

The third part's task is to overcome this problem, Walker II (1990) introduced a solution for this. The number of smaller subtrees is divided by the available space between two larger subtrees, which result in a distance value for smaller subtrees. Due to this value, the smaller subtrees are placed evenly between the larger ones. Thereby they are spaced out evenly (see Figure 5.1(e)).

The algorithm consists of several single tasks, such as finding the left and right contour of a subtree and spacing out smaller subtrees. The second one, which is called *apportioning*, is improved by Buchheim et al. (2002) to run in linear time. The improvements are related to finding nodes such as the greatest uncommon ancestors or traversing a contour. However, the principle of the algorithm is the same.

Essentially, the technique of the Walker layout visualisation is the implementation of the algorithm published by Buchheim et al. (2002, Appendix A). This algorithm is used for the node placement on the 2d plane. Then the nodes are connected by lines according to their parent-child relationships. No further actions are made in the layout process.

5.2 The Walker Layout Browser

5.2.1 Rendering

After the layout algorithm has assigned a position to each node, the rendering process transforms the abstract tree to a shape with a specific appearance. Therefore graphical elements are needed, which represent the logical elements of the tree. The rendering algorithm draws the tree shape on a canvas, whereby the node positions are scaled to window size (see Figure 5.2).

The nodes are rendered as icons, either as coloured circles or as symbols according to their document type. The user can choose between these possibilities in a settings dialogue in the framework. A type

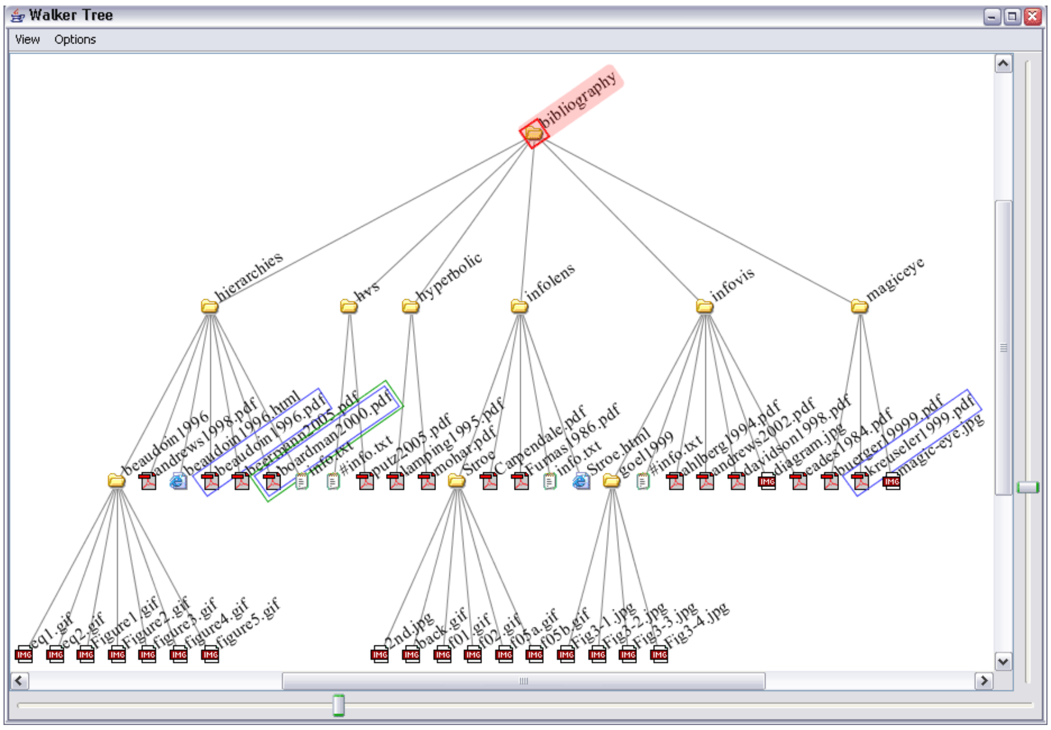


Figure 5.2: The Walker Layout Browser. Nodes are laid out side by side and subtrees are placed as close as possible to the right contour of the left neighbour subtree. Parent nodes are positioned centrally above their children.

specific representation of documents gives a rapidly understandable overview of the content information of the tree. The rendering algorithm does not influence the sizes of the icons, they are always the same, independent of their position and zoom factor. The root node can be highlighted with a configurable colour to emphasise it. This is not really necessary in this browser, however, it is implemented for compatibility reasons with the browsers described in the next chapters.

The hierarchical relations of the nodes are drawn as connecting, straight lines. The colour and the stroke width can be chosen by the user in a dialogue of this browser.

The names of the nodes are drawn as labels beside the according nodes. The font type and size can be chosen in the framework, the colour in a dialogue of the browser. In the Walker layout the nodes of the same level are arranged on a horizontal line. If the labels would be drawn horizontally beside their nodes, many of them would overlap in a normal scaled tree. For this reason, the labels can be rotated counterclockwise from zero to ninety degrees. This improves the readability of the names.

The labels can be hidden in two manners. Firstly, the user can chose this option for all labels of the tree. This may be useful to have an overview of large trees. Secondly, a dynamic distance calculation hides single labels, if nodes are too close to one another. This is an important feature, because otherwise, the view on a large tree, would be cluttered with label texts. The user can choose the distance threshold, when labels are suppressed.

Highlights of nodes are drawn as rectangular bounding boxes, whereby the colours depend on the type of the highlight. There are selection highlights, search result highlights and a selected search result highlight. The colour is chosen in a dialogue in the framework. The sizes of the highlights are different due to their type, since a node can has more than one highlight.

Selecting nodes can be done by dragging a selection box over nodes. The selection box is semi-transparent with the same base colour as the selection highlights. A semi-transparent appearance enables users to have an overview of the part of the tree they are selecting.

5.2.2 Navigation

Navigation and interaction are essential facilities in information visualisation. Layout and rendering of trees alone can often not cope with large trees. Interactive techniques are necessary to explore large hierarchies. The Walker layout browser features zoom and pan, which are very traditional tools in information visualisation.

Zoom

The browser starts with an overview of the whole tree. In small trees the details can be seen together with the overview. However, large trees can hardly be recognised, since there is no magnification or focus of a selected area supported in this browser. The Walker layout browser uses the traditional zoom, which magnifies every part of the tree with the same proportions. Thus the drawing area becomes larger than the viewport. A detailed view of a part of the tree is presented while the rest of the tree disappears outside of the drawing area.

Zooming can be done in three ways, with the scroll wheel of the mouse, by drawing a zooming rectangle, and with sliders at the bottom and the right side of the visualisation window. The mouse wheel zooms to the point where the mouse is activated. The sliders zoom to the centre of the viewport. Drawing a zooming rectangle by dragging the mouse zooms into the content of this rectangle. Figure 5.2 shows a zoomed view of a hierarchy.

Most of the labels are initially hidden because of the distance threshold. When zooming to a point of interest, the nodes are arranged wider from one another, thus the labels appear. Hence, in addition to the structural zooming, a content-based zooming is provided.

In general, zooming in these graph-based views means, that the positions of nodes and lines are changed to be closer or wider to one another. The sizes of the graphic elements remain always in the same size. A pixel-oriented geometric zooming would also magnify the node icons.

Pan

Since only a small part of a scaled tree can be shown in the viewport, panning is needed to explore the context of a zoomed area. This is done by dragging the viewport around with the mouse. Obviously this is not convenient in very large trees, because in a highly zoomed drawing it is hard to explore the tree in this way.

Focus

If a node becomes focused it is placed in the centre of the viewport. Therefore the tree drawing is moved correspondingly. Focusing is initiated by double clicking on the interesting node or by receiving a focus event from the framework.

Maximise

Maximisation is related to the selected nodes. The rectangular area on which the selected nodes are lying is zoomed in. Thus they are shown detailed in the drawing window. Maximisation is performed due to a user request in this browser window or an event from the HVS framework.

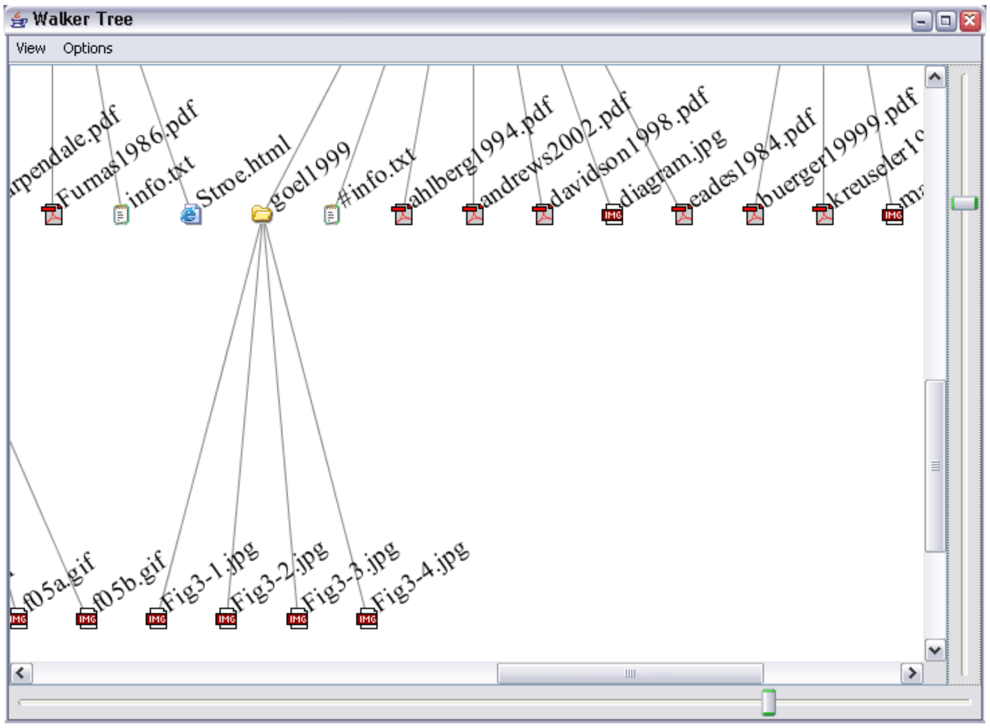


Figure 5.3: A zoomed view of a hierarchy is shown in the screen shot. The context of the magnified area has disappeared outside of the drawing window.

5.2.3 Integration into HVS

Events received from the Framework

As described in Section 4.2, message events are received from the framework for synchronisation reasons. The following events are initiated by the user in another HVS view and can be received by the Walker layout browser:

- *Expansion:* Expansion events, in order to expand and collapse an inner node, are ignored. The walker layout is always drawn with all nodes and links.
- *Selection:* Selection events change the selection state of the tree. New selections are made in the browser according to the state received in the event. The selected nodes are marked with bounding boxes in the colour provided by the HVS framework.
- *Focus:* A navigation event containing a focused node causes the layout to show this node in the centre of the window.
- *Maximise:* A navigation event containing a maximise command causes the browser to change the view, in order to show all selected nodes fully in the whole window.

This group of events occur due to user actions in the framework or changes in the data model. Reasons therefore are a complete change of the hierarchy by the user, a modification of the hierarchy data in a visualisation, filtering of the data initiated by the user, or continuous reading of data by a data source thread.

- *Node Insertion*: An inserted node in the framework is integrated into the internal data structure, then the tree layout has to be recalculated and redrawn.
- *Node Removal*: A removed node is deleted from the internal data structure, the tree layout has to be renewed.
- *Node Renaming*: Renaming a node is a simple task, only the label has to be changed in the browser.
- *Data Model Changed*: If the data model has changed completely, the tree has to be created anew.
- *Search Result*: Due to a change of a search result, all nodes in the result are surrounded by a bounding box. The bounding box of the selected search result node is drawn in a different colour.
- *Filtering*: A filter action by the user causes a *data model changed* event, which is described below in the next listing.

Events sent to the Framework

Performing an action in the visualisation causes the browser to send these actions to the framework. This is necessary to keep the other views synchronised. There are three types of synchronisation events, the first type comprises the actions performed on the navigations and selections:

- *Selection Changed*: If the user selects or deselects nodes, then the updated selection sate is transmitted to the framework, which synchronises the other views.
- *Focus Changed*: Focusing nodes is a navigational action which is sent to the framework to provide the same view on the tree in the other visualisations.
- *Maximise*: Maximising the view for the selected nodes is also a navigational action, which needs to be synchronised.

The second type is related to changes of the hierarchy data, since the user can manipulate the tree data. Unlike the type described above, the command for the action is sent to the framework, which performs the modification on the data hierarchy, then the new data is accessible for the browser. The single actions are as follows:

- *Node Insertion*: A node can be inserted as a child of an inner node. The inserted node can be a collection or a document, which enables the user to establish a hierarchy structure.
- *Node Removal*: Documents and collections can be removed. If the removed node is a collection, all descendants are also removed.
- *Node Renaming*: The name of a node, which is represented by the label, can be changed.

The third type causes an action outside of HVS, a synchronisation is not needed therefore.

- *Open Document*: A document can be opened with an external viewer. For example, images or text files are sent to a native viewer installed on the operating system for display.

Rendering Settings

There are some options dialogues in the HVS framework by which render options can be set by the user. They are implemented in the framework because these settings are applied on all visualisations. The browser requests the information in each paint process. If the user changes an adjustment, a call to repaint the drawing is sent from the framework to the browser. Thus the new setting is updated automatically in the browser. The settings can be accessed through the class `VisualizationProperties`. The following rendering information is available:

- *Colours*: Colour settings can be obtained for the bounding box marks of the selected nodes, the nodes which are in the search result set, and the node which is selected in a search result set. Furthermore, a colour is available for the drawing background.
- *Icons*: The framework provides a set of icons to represent nodes in the graph. The icons can be requested for different document types and for the inner nodes. The framework determines the icon scheme and the sizes of the icons.
- *Font*: The fonts for the labels are another graphical setting from the framework. The user can set the font type and size.
- *Tooltip Text*: Since the framework manages the hierarchical data with the attributes of the items, the tooltip text is created by the framework. The browser fetches the text and visualises it on demand for a node.

5.3 Selected Details of the Implementation

Since all hierarchy browsers discussed in this thesis are based on graph layouts, they use some parts of the implementation in common. The following selective implementation pieces are identical for these browsers.

5.3.1 Data Model

The data model of the framework provides structural and attribute information. However, the Walker layout browser and the browsers described in the next chapters need more functionality. For example, the Walker layout algorithm needs to store some temporary placement information for each node. Rendering positions and 2d extent on the drawing plane are needed to detect a clicked node. Thus an internal data model is established for the visualisations.

The class `TreeNode` is essential for the hierarchical structure. It is derived from the class `Default-MutableTreeNode`, which is part of the Java Foundation Classes (Project Swing). The DefaultMutable-TreeNode class provides all functionality to create a tree hierarchy. TreeNode classes are used for both document and collection nodes, the information about the type is stored within the class. It stores the following information, which can be accessed from outside:

- *HVS node:* The original HVS node is stored as object for data hierarchy synchronisation reasons.
- *Node type:* The information if a node is a document or a collection is retained.
- *Synchronised states:* The information if a node is selected, focused, within a search result, or selected within a search result are saved as flags.
- *Temporary layout information:* For example, the Walker layout algorithm needs six fields to calculate the positions of the nodes (preliminary position on the x-axis, modifier value, shift value, change value, thread node, and ancestor node).

```
/**
 * The method returns an object array of the selected TreeNodes.
 */
public Object[] getSelectedNodes ()
{
  return selectednodes_.toArray ();
}
```

Listing 5.1: An effective way to receive all currently selected nodes.

```
/**
 * Returns a TreeNode of the linear list with the given index.
 * @param index The index in the linear list.
 * @return The TreeNode with the given index.
 */
public TreeNode getTreeNode (int index)
{
  if (index < 0 || index >= allnodes_.size ())
    return null;

  return (TreeNode)allnodes_.get (index);
}
```

Listing 5.2: The method returns single `TreeNode` objects for an index value from an internal linear list.

- *Rendering position and extent size:* The position and extent size on the drawing canvas are stored. This is needed to detect if a node is hit by a mouse click.
- *Scale factor:* The scale factor determines the size of the drawn icon and label.

To create, access and modify the hierarchy built on TreeNode objects, there is a class named `TreeGraph`. This class provides managing methods for easy and effective usage of the hierarchy data. Its task is to hold information about the root node, the selected nodes, the focus node, the tree depth and the number of nodes.

Furthermore it contains a list of all nodes which are selected. Obtaining all selected nodes in the tree can be effectively provided, since only the list is returned by the method. Otherwise, an algorithm would have to traverse the tree recursively and collect the selected nodes. This functionality is often needed, so an effective implementation is essential. Listing 5.1 shows the implementation of the method which returns the selected nodes.

Considering an approach of storing hierarchical tree data in a table for reasons of efficiency (Fekete, 2004), the hierarchical data are also stored as a list of references to `TreeNode` objects. Thus nodes can be obtained in two ways, either by using the parent-child relationship or by accessing the list with an index of the node. Listing 5.2 shows how to get a `TreeNode` object for an index value.

A use case for traversing the tree data linearly is hit detection. When the user clicks with the mouse on a node, an algorithm has to find the node which lies under the mouse. The nodes must be queried for their position in the reverse order they were drawn. The last drawn node is put on the top if nodes are

```
/**
 * The method finds the nodes which is rendered at the given
 * position or returns null otherwise.
 * @param x The x—coordinate of the position
 * @param y The y—coordinate of the position
 * @return The node lying on the given position
 */
public TreeNode getTreeNodeAtPos (int x, int y)
{
  TreeNode treenode;
  TreeNodeIcon icon;

  // search from end to start, because the nodes were painted in
  // the other way, the later nodes are draw over the former ones.
  for (int i = treegraph_.getTreeNodeCount () − 1; i >= 0; i−−)
  {
    treenode = treegraph_.getTreeNode (i);
    icon = treenode.getIcon ();

    if (icon.contains (x, y))
      return treenode;
  }
  return null;
}
```

Listing 5.3: Hit detection using the internal linear node list.

overlapping. Obviously, performing this task recursively would be more extensive than processing a list from the tail to the head. Listing 5.3 shows this implementation.

5.3.2 Integration into HVS

The visualisations are designed to run in two modes. They may run as stand-alone applications which read the data from the local file system. Secondly they act as plug-in view in HVS. For this reasons, most parts are independent from HVS and its data and event model. Therefore the connecting class `HVSView` manages all interface functionalities to HVS, which are described in Section 4.2. Its main tasks are:

- *Derivation of the HVS* `Visualization` *Superclass:* A HVS visualisation plug-in must be derived from the `Visualization` superclass in order that it can be used by the framework.
- *Graphical User Interface (GUI):* HVSView implements the necessary GUI components, which are a drawing canvas and an options menu, and manges the mouse input.
- *Events from HVS:* Receiving of HVS events and forwarding to the appropriate component is another task of `HVSView`.
- *Event to HVS:* Events from the visualisation, such as selecting nodes, are sent to the `HVSView` class, which forwards it to the HVS framework.
- *Accessing HVS Properties:* HVSView accesses the HVS `VisualizationProperties` class and forward these values to the rendering module.
- *Creation of an internal data model* HVSView creates an internal data model for the visualisation and synchronises it with the HVS data model (see below).

```
/**
 * A new node is inserted in the internal data model.
 * @param event The HVS event containing the inserted HVS node.
 */
public void nodeInserted (DataModelEvent event)
{
  // get the hvs node from event
  Node hvsnode = event.getNode ();

  // get locations of node and insert the node into each location
  TreePath[] treepaths = event.getParentPathes ();
  for (int i = 0; i < treepaths.length; i++)
  {
    // create a new treenode using the hvs node
    TreeNode treenode = new TreeNode (
      hvsnode, hvsnode.getName () , hvsnode instanceof Collection);

    // insert treenode at correct position in the internal data model
    TreeNode parentnode = findTreeNode (treepaths[i]);
    Vector hvschildren = datamodel_.getChildren (
      (Collection)parentnode.getNodeObject ());
    for (int j = 0; j < hvschildren.size () ; j++)
      if (hvschildren.get (j) == hvsnode)
        parentnode.insert (treenode, j);

    // notify the internal data model
    treegraph_.nodeInserted (treenode);
  }

  // visualisation update
  geometryengine_.recalculate ();
  renderengine_.render ();
}
```

Listing 5.4: The method which handles the event received from HVS when a node has been inserted.

There is a second analogous class for the stand-alone mode, which fulfils the same tasks, but in a different way. The data are read from the file system and events are ignored, since the stand-alone application is not synchronised. All other parts of the visualisation have no information about its environment, they simply use an abstract superclass of them for the communication.

Listing 5.4 is an example how the data hierarchy is synchronised by *HVSView*. It shows the processing of an event from the HVS framework when a node has been inserted. *HVSView* inserts this node in the internal data model and updates the visualisation.

5.4 Outlook and Further Work

Animation is currently not implemented for automated transitions. Focusing a node by double clicking it in this or another visualisation moves the drawing in a single step to centre this node. The movement should perhaps be animated to prevent a loss of orientation. A second transition is the zooming by dragging a zooming rectangle. The zooming from the current view area to the selected zooming area can be animated. The same can be applied on the maximisation process.

Chapter 6

Hyperbolic Browser

The hyperbolic browser is an elegant and subtle solution to the problem of providing a focus plus context implementation for large hierarchies. It uses a non-euclidean geometry, hyperbolic geometry, which is mapped to the euclidean unit disc.

This chapter describes the basics of hyperbolic geometry and the mapping to euclidean space, followed by a tree layout technique using this geometric background. Afterwards it describes a browser built on this technique and details of its implementation.

6.1 Hyperbolic Layout Technique

The hyperbolic visualisation method was introduced in Lamping et al. (1995). An overview of this browser type and related work is given in Section 3.9. This thesis presents an implementation of the hyperbolic browser, which is also integrated into HVS.

The hyperbolic browser is intended to deal with large trees without loosing a detailed view on any part of the hierarchy. Due to a mapping from hyperbolic space to the euclidean unit disc, a focus plus context effect arises.

Following the discussion in Walter and Ritter (2002), Herman et al. (2000), Lamping et al. (1995), Mohar (1999), and Thruston (1997), an introduction to hyperbolic geometry is presented before explaining the hyperbolic layout technique. Hyperbolic geometry is based on an axiomatic system, which has its origins in euclidean geometry.

6.1.1 The Hyperbolic Plane (H2)

Euclidean geometry is built on five axioms and defines the space which is natural in the sense of human perception. The fifth axiom states that if a line does not intersect a point, then there is exactly one line which is parallel and intersects this point. By definition, a line is the shortest way from one point to another. Declining this axiom results in two types of homogeneous non-euclidean geometries. In both geometries, traditional trigonometric equations are no longer valid. For example, the sum of the internal angles of a triangle is no longer 180 degrees.

Depending on the angle sum there are three types of homogeneous two-dimensional geometries: *Euclidean geometry*, which has an angle sum of exactly 180 degrees, *spherical geometry*, which has an angle sum of more than 180 degrees, and *hyperbolic geometry*, which has an angle sum of less than 180 degrees. The latter is also called the *hyperbolic plane* or *H2*.

Spherical geometry can be imagined as a surface on a sphere. When drawing a triangle on a sphere, where one angle lies on the pole and the other two somewhere distantly on the equatorial circle, it can be easily seen that the angle sum is greater than 180 degrees. Unfortunately there is no intuitively

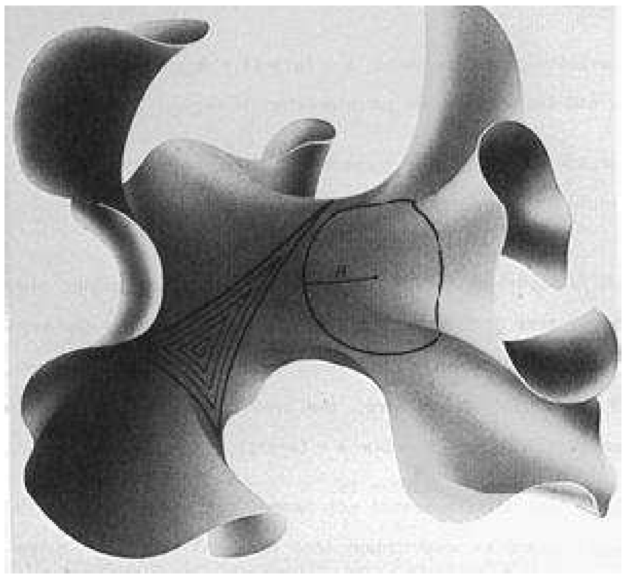


Figure 6.1: A hyperbolic plane embedded into 3d euclidean space. This illustration shows that there is more room in hyperbolic space than in euclidean space. [Figure extracted from Proc. of KDD '02. Copyright by the Association of Computing Machinery, Inc.]

understandable 3d visualisation for the hyperbolic plane. It is often visualised as a shape with a negative curvature embedded into 3d euclidean space as shown in Figure 6.1.

The hyperbolic plane is a mathematical object with unusual properties. Parallel lines diverge away from each other. This leads to the property that the circumference of a circle on the hyperbolic plane grows exponentially with its radius. Therefore the amount of space rapidly grows with the distance. The mathematical expression of this property is given by the equation of the circumference in the hyperbolic plane:

$$c\,(r) = 2\,\pi\;sinh\,(r) \tag{6.1}$$

For small radii the circumference is nearly equal to a circle in euclidean space. However, for greater radii it grows exponentially with the radius.

A mapping from 2d hyperbolic space to the euclidean plane is needed. Whereas the mapping from spherical geometry to the euclidean plane is a well known task, the mapping from H2 is a difficult problem because of the exponential growth of room. The embedding of H2 into 3d euclidean space helps to understand hyperbolic geometry, however, it is not suitable to benefit from the visualisation potential of H2. There are several mapping methods, which all have to abandon one or more characteristics, such as angles or lengths.

The best known are the Klein model and the Poincaré model. Both of them map the infinite hyperbolic plane into the euclidean open unit disc. The Klein model preserves lines and distorts angles, whereas the Poincaré model preserves angles and distorts straight lines to arcs. The following discussion concerns the Poincaré model.

6.1.2 The Poincaré Unit Disc (PD)

The hyperbolic plane is mapped into the unit disc which is also called *Poincaré unit disc* (PD) in this context. This disc shows a representation of the H2, not the hyperbolic plane per se. PD can be drawn on the euclidean plane, but has hyperbolic metric. A model means the projection of a subset of geometric characteristics into the euclidean unit disc.

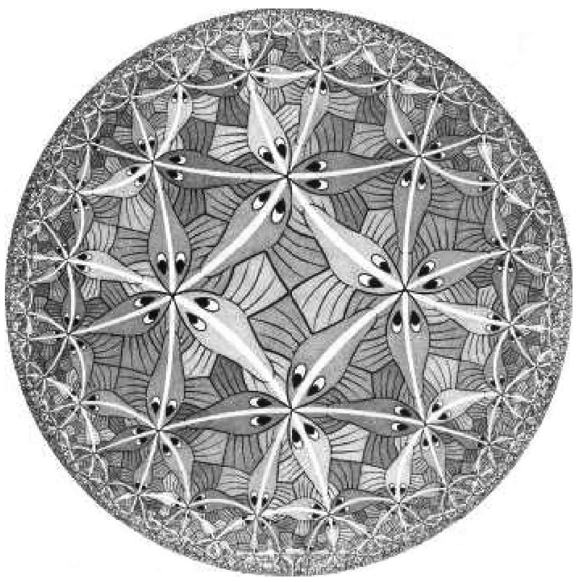


Figure 6.2: A hyperbolic unit disc illustrated by Maurits Escher (Circle Limit III). Escher was fascinated by the infinite space within the disc and influenced the beginning of the hyperbolic layout in information visualisation. The white arcs are straight lines in the hyperbolic plane. [Figure extracted from Proc. of KDD '02. Copyright by the Association of Computing Machinery, Inc.]

The properties of the Poincaré model makes this mapping convenient for layout and browsing purposes. A famous drawing by Laurids Escher illustrates some characteristics of the Poincaré disc (see Figure 6.2). The most important properties are:

- *Infinite space:* The infinite hyperbolic space is mapped entirely into the PD. That way an infinite space can be visualised on the display.
- *Circle rim:* All distant points in H2 are close to the circle rim without touching it. There is infinite space near the rim.
- *Focus plus context:* Each location in H2 can be mapped to the circle centre, which magnifies this location. All other parts are placed near the rim. A smooth transition from focus to context can be observed.
- *Lines:* Straight lines in H2 become circle arcs in PD, which orthogonally intersect the border of the disc. The lines diverge away spanning out more room than in euclidean geometry.
- *Angles:* Angles and therefore forms are preserved in the PD.
- *Circles:* Circles in H2 are mapped into circles in the PD, though they shrink in size the further they are away from the origin. Additionally, every circle in PD is also a euclidean circle.

To change the focus point in PD, a translation operation is needed, which is fulfilled by the following transformation equation. The values and parameters for this formula are complex numbers representing a location in the euclidean plane. The transformation respects the hyperbolic metric in PD when modifying position values.

$$T(z) = \frac{\Phi z + P}{1 + \overline{P} z}, \qquad |\Phi| = 1, |P| < 1 \tag{6.2}$$

The formula expresses the transformation of the point z in PD to the transformed point T(z) in PD. The complex number Φ describes the pure rotation of PD around the origin, hence the length of Φ has to be exactly one. P describes the translation of PD, whereby the origin of PD is mapped to P and -P becomes the new centre in PD. $\overline{P}$ is the complex conjugate of P.

A pure translation without rotation is affected by setting the rotation parameter Φ to one (Φ = 1 + 0i). The reduced transformation formula performs only a translation:

$$T(z) = \frac{z + P}{1 + \overline{P}z}, \qquad |P| < 1 \tag{6.3}$$

Similar to euclidean space, this translation is an isomorphic transformation, which preserves forms and metric in H2. Circles remain circles in both H2 and PD. Straight lines are preserved in H2 and therefore the corresponding circle arcs remain circle arcs in PD.

6.1.3 Tree Layout

To benefit from the potential of hyperbolic geometry, a technique is needed which makes it usable as an information visualisation method. This section describes a technique based on the approach described in Lamping et al. (1995).

The tree layout method resembles the radial tree layout described in Section 3.4. Considering the problem with the radial layout, there is too little space for large trees. Essentially, the difference to the radial layout is the underlying geometry. In hyperbolic space there is much more room for large trees. Thus this layout technique yields a useful result, whereas the same tree layout in euclidean space is not practical.

Trees grow exponentially with their depth because of their inherent structure. Considering Equation 6.1 in the last section, the circumferences of circles also grow exponentially with the radius. When laying out the tree radially with the root in the centre, the tree grows in the same way exponentially as the hyperbolic space, thus the tree can be incorporated in H2. The result is that there is enough space for nodes and subtrees at an arbitrary level. This is a main characteristic, that the available space is nearly equal for all nodes independent of their level.

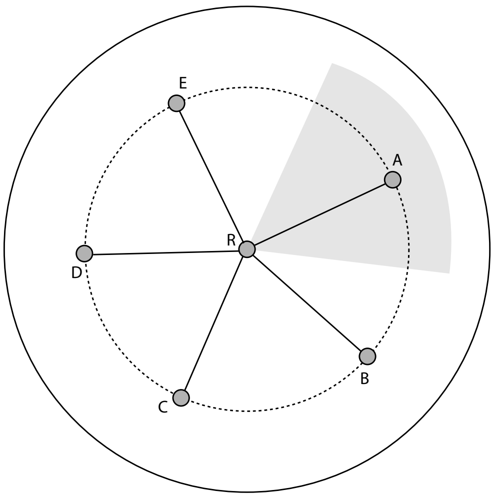
At this point the question arises how to lay out the tree in H2. The crucial point is that each node is laid out in the centre of PD and the children are placed on a circle arc which surrounds the centre. The circle arc is separated into several wedges, where one wedge is assigned to each child. Each child node is positioned in the middle of its wedge (see Figure 6.3(a)).

Now the child nodes are positioned relatively to the parent node. To determine the real position, it will be translated by using Equation 6.3 with the real location of the parent node as the translation parameter. The new position is stored for use in the initial layout.

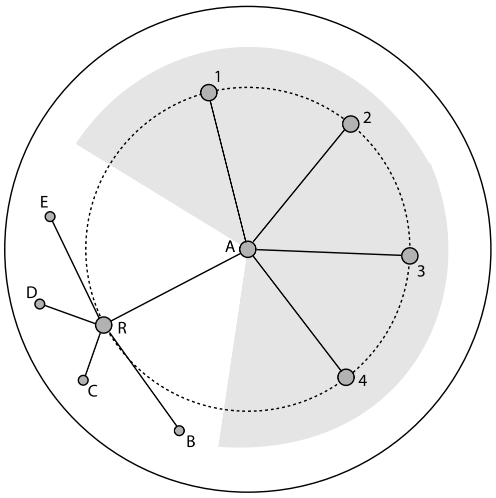
Then each child node and its wedge is translated into the centre of PD using Equation 6.3 again. By reason of hyperbolic geometry the angle of the wedge strongly increases. Then the children of the now centred child node are laid out on the magnified wedge (see Figure 6.3(b)).

This procedure is recursively applied to all nodes. Since nodes have wedges and angles are preserved in PD, the subtrees of adjacent siblings do not overlap. Circles in PD are also circles in euclidean geometry (Mohar, 1999), so the nodes can be placed on the circle arc in the same way as in an euclidean circle. The nodes keep this circle form of their arrangement.

A typical value for the length of the wedge radius is 0.7, in any case it must be between zero and one, because PD is a unit disc. For greater lengths there is more room for the nodes on the arc. On the other side the context is more pushed to the rim. For smaller radii the context is better visible, but the nodes are placed closer to each other.



(a) The root node is centred and its children are placed in a surrounding circle. The wedge of node A is drawn in grey.



(b) A translation has moved node A to the centre, increasing the wedge of A. The children of node A (node 1, 2, 3, 4) are laid out on the larger wedge.

Figure 6.3: Two schematic illustrations outline the layout of nodes in PD. A child node is drawn on a circle arc surrounding the centre, then it is translated with its wedge to the centre, which increases its wedge.

The sizes of the wedges are not evenly apportioned, but they depend on the extents of their subtrees. The nodes in a subtree are summed up logarithmically for all siblings. By comparing this value with the sum of the values of all siblings, the size of a wedge is determined for a node. That way the subtrees are spaced out evenly.

After the recursive calculations of all node positions, the nodes are placed in PD according to their location coordinates. Since PD lies in the euclidean plane, the positions are already usable for the layout (see Figure 6.4)

6.1.4 Change of Focus

Changing the focus is performed by translating the tree in PD. The same equation (Equation 6.3) is used for the translation as for the tree layout. To move a tree, the translation formula is applied on each node with the same parameter. On the euclidean display, the nodes near the centre are moved further than nodes near the rim. In this way dragging the tree can be implemented. The translation affects only the current positions of the nodes. Neither a new tree layout has to be made, nor the original locations for the first layout are needed.

To move a specific node to the centre of PD, a change of focus is performed, whereby the translation parameter consists of the negative coordinates of the point which should be focused. Due to Equation 6.3 the new centre of a translation is the negative translation parameter.

6.2 The Hyperbolic Browser

The hyperbolic browser is based in many aspects on the Walker layout browser. However, the tree layout and the navigation are essentially different. This section only discusses the features which are different or extended.

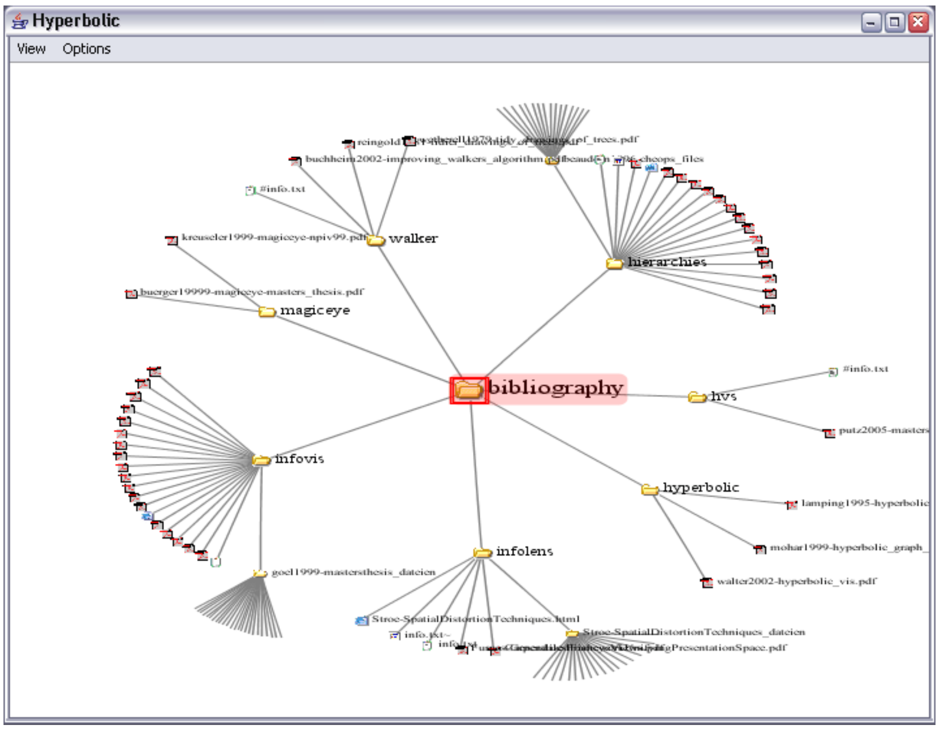

Hyperbolic
View
Options
bibliography
walker
magiceye
hierarchies
hvs
infovis
hyperbolic
infolens
reingold
buchheim2002-improving_walkers_algorithm
#info.txt
kreuseler1999-magiceye-npiv99.pdf
buerger19999-magiceye-masters_thesis.pdf
#info.txt
putz2005-masters
lamping1995-hyperbolic
mohar1999-hyperbolic_graph_
walter2002-hyperbolic_vis.pdf
goel1999-mastersthesis_dateien
Stroe-SpatialDistortionTechniques.html
info.txt~
Stroe-SpatialDistortionTechniques_dateien

Figure 6.4: An initial hyperbolic layout drawn by the HVS hyperbolic browser.

6.2.1 Rendering

The unit disc and the node locations inside are scaled to a variable window size. In contrast to the Walker layout browser, all nodes are drawn into the disc, so scrolling is not needed. Since most nodes are placed near the rim, the possible problem of cluttering is avoided by scaling and hiding the node icons and labels. Nodes near the centre are drawn in full size, near the rim they are successively scaled down with the distance, until they are completely hidden (see Figure 6.4).

The area in the disc can be divided into three parts. The inner part around the origin up to about half the radius is the focus area, where a small part of the tree is magnified to nearly its normal size. The ring around the focus area represents the contextual information of the focused tree part. It is distorted to smaller sizes and angles, but the contained structural and content information is still recognisable. All other tree parts are placed in the outer ring bordering the rim. An overview of the structure is visualised there. Thus the hyperbolic browser provides three levels of detail.

To achieve smooth panning and zooming, a *guaranteed frame rate* is provided by the rendering process when the tree is moving. The frame rate can be chosen by the user and determines the time how long the drawing procedure may take. If this time has elapsed before the rendering was finished, the algorithm stops and starts drawing a new frame at the next position. This prevents unbalanced motion with hanging frames. However, the effect is gained at the price of incomplete images during a transformation. Additionally, anti-aliasing is turned off until motion has finished. The final frame after motion has completed, is drawn in full with anti-aliasing enabled and without time constraints.

6.2.2 Navigation

Zoom and Pan

Due to the focus plus context effect, zooming and panning is performed in one user action. In the centre of the disc the visualised tree is magnified. To zoom in to a particular part of the tree, this part is dragged into the centre of the disc. The context of the newly focused subtree is also magnified and surrounds it. Using this contextual information, users can explore the tree by panning the interesting parts into the focus area (see Figure 6.5).

When parts are moved from the context into the focus area, a visual *fan out effect* can be observed. This is caused by the hyperbolic geometry in the Poincaré disc model, which distorts the area in the context more than in the centre. If a subtree is moved into the origin of PD, then the angles are visualised similarly as they are. The diminished angles and length near the rim are magnified to almost normal size, which can be observed as the visual fan out effect. For users this effect is a pleasing and aesthetic property, while exploring the information.

To change the *magnification magnitude* in the focus area, the radius of the layout wedges can be changed (see Section 6.1) using the mouse wheel. A larger radius increases the magnification, whereby the context is moved towards the rim. Details are seen better, but the context information decreases. This may be necessary, if a parent node has many child nodes. Decreasing the radius affects less magnification and more contextual information. This is suitable if the focused part of the tree is not very dense (see Figure 6.6).

Focus

Focusing a node in the hyperbolic browser means moving it to the disc centre. There a focused node is magnified together with its direct by related nodes, surrounded by its context. The corresponding label is drawn in full size and the content information can easily be gathered with the help of the tooltip.

The interesting node can be manually dragged into the disc origin, or be automatically moved by an animation task to the exact centred position. Double clicking the node or an external synchronisation event can cause a node to become focused. The guaranteed frame rate ensures a smooth transition.

Maximise

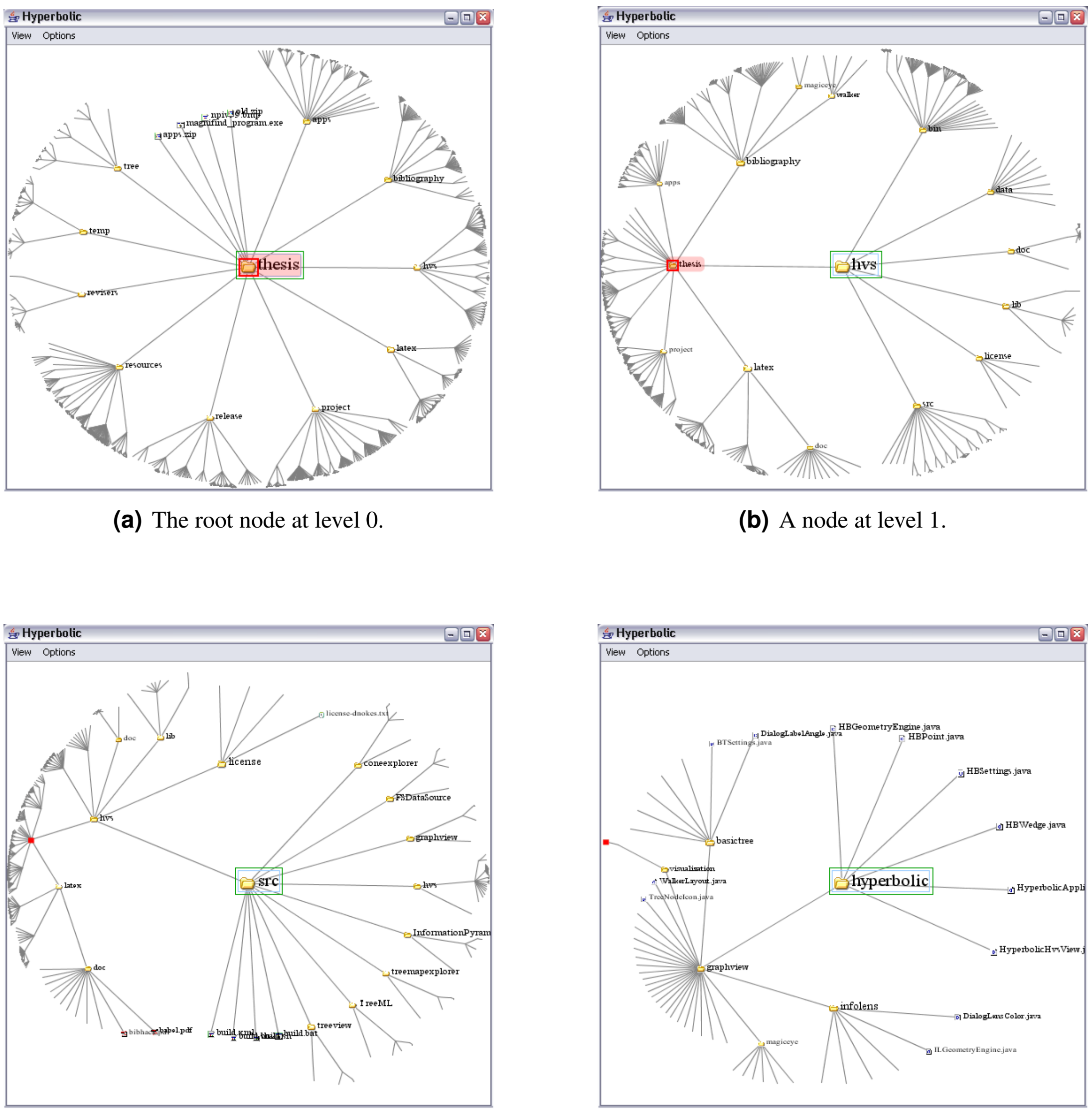
Maximisation of a number of nodes (the selected nodes), can not be fully accomplished in the hyperbolic browser if the nodes are at distant positions in the tree. Considering that only a small part of the tree can be emphasised while the rest is scaled down, the hyperbolic browser does not produce good results under such conditions. However, if the nodes are close to one another, such as siblings are, then a good result is achieved.

The maximisation algorithm collects all selected nodes and calculates the arithmetic average values of the coordinates. Then the location with this average value is focused. Therefore, it depends on how distant each node is from the average value, whether it is moved to the focus area.

6.2.3 Tree Layout

An alternate tree layout can be chosen, which draws the tree to a preferred direction instead of the default radial layout. There are four directions, right, left, top, and bottom, to which the hierarchy can be adjusted. The layout angle is restricted to ninety degrees, where the median angle of these sectors is adjusted (see Figure 6.6) . In other respects the layout is done in the PD as for the radial layout.

Navigation in an adjusted tree is performed in the preferred direction. For example, a user navigates from left to right to explore a tree from the root node to the leaves. An adjusted layout provides more



(a) The root node at level 0.

(b) A node at level 1.

(c) A node at level 2.

(d) A node at level 6.

Figure 6.5: The four figures demonstrate navigation to a particular node. Four focused nodes at different levels are shown with their child nodes. Even the node at the highest level has enough space for its children.

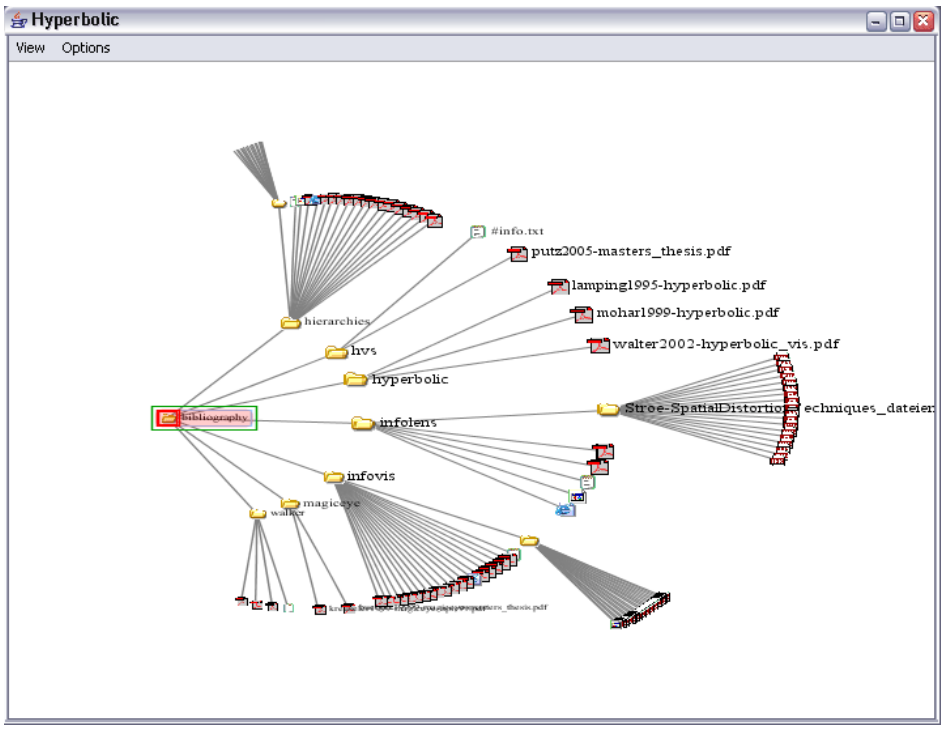

Hyperbolic
View
Options
#info.txt
putz2005-masters_thesis.pdf
lamping1995-hyperbolic.pdf
moharl1999-hyperbolic.pdf
walter2002-hyperbolic_vis.pdf
hierarchies
hvs
hyperbolic
bibliography
infolens
Stroe-SpatialDistortionTechniques_dateien
infovis
magiceye
walker

Figure 6.6: A hyperbolic layout which is adjusted to the right is shown in this figure. The magnification magnitude is low, which provides a good overview of a small tree.

uniform layout and navigation, which can help to better find hierarchical information. In a right-adjusted layout, the labels can be seen better, because the natural adjustment of words is the same.

6.2.4 Integration into HVS

The integration of the hyperbolic browser is implemented nearly in the same way as for the Walker layout browser. The same events from the HVS framework are received, which cause analogous actions. Actions performed by the user cause the corresponding events to be sent back. The rendering properties are also processed analogously.

6.3 Selected Details of the Implementation

6.3.1 Translation

Nodes are translated in PD using Equation 6.3 (see Section 6.1). The implementation is done in the class `HBNode`, which stores the location of a node and which provides the method `translate (double tx, double ty)` to apply the transformation on itself. The parameter determines the change of focus to the negative parameter coordinates, and the equation determines how far the point is moved depending on its current location. Listing 6.1 shows the implementation of the formula using complex arithmetic.

6.3.2 Tree Layout

The principle of tree layout in hyperbolic space is discussed in Section 6.1. Listing 6.2 shows the implementation of the recursive layout. The children of each node are placed around the disc origin in the

```
/**
 * Translation in PD of this point by a translation parameter
 * using the transformation formula from Lamping et al. 1995.
 * @param tx The translation parameter (x-coordinate)
 * @param ty The translation parameter (y-coordinate)
 */
void translate (double tx, double ty)
{
  // t(z) = (z + t) / (1 + z * conj(t))

  // numerator
  double numx = x_ + tx;
  double numy = y_ + ty;

  // denominator
  double denx = (x_ * tx) + (y_ * ty) + 1.0;
  double deny = (y_ * tx) - (x_ * ty) ;

  // division
  double dd = (denx * denx) + (deny * deny);
  x = ((numx * denx) + (numy * deny)) / dd;
  y = ((numy * denx) - (numx * deny)) / dd;
}
```

Listing 6.1: The method which accomplishes translation of a point in the Poincaré disc.

middle of its wedge. The node and the wedge are translated according their parent's location and this position is stored for the use in the final layout. The wedge is moved to the disc origin for the layout procedure of its children.

To achieve an evenly apportioned tree layout, weights are assigned to nodes. Nodes with wider subtrees are weighted higher than nodes with smaller ones in order to give them more extended wedges. The weight of a node is calculated from the logarithm of the sum of its children. Since this computation is recursively performed, the descendants of a node at a higher level are less important then descendants at a lower level. In other words, a node with few children and a wide subtree at a higher level has a lower weight than a node with many children and a small subtree.

Mathematically expressed, for each inner node in the tree the following equation is used for the calculation:

$$w_{node} = log_e \left(\sum_{i=1}^{n} w_{child} \right), \qquad n > 0 \tag{6.4}$$

where w_{node} is the weight of a node, w_{child} the weight of a child, and n the number of the children of the node. If a node has no children, then the weight is set to one. Listing 6.3 shows the recursive implementation.

6.4 Outlook and Further Work

6.4.1 Magnification Magnitude

In the current implementation the magnification magnitude (see Section 6.2), which determines the magnification in the focus area, is fixed for the complete layout. Though it can be changed by the user during the navigation by activating the mouse wheel, it is always applied on the whole tree. The modification is

```
/**
 * The recursive layout algorithm for the layout in the Poincare disc
 * @param treenode The treenode (and its subtree) to lay out
 * @wedge The wedge in which the node and subtree are placed
 * @length The radius of the wedge
 */
protected void layoutHBNode
  (TreeNode treenode, HBWedge wedge, double length)
{
  // layout the node
  // nothing to do for the root node
  TreeNode parent = treenode.getParentNode ();
  if (parent != null)
  {
    double angle = wedge.getBisectionAngle ();

    // We first start as if the parent was the origin.
    // We still are in the hyperbolic space.
    treenode.z.x = length * Math.cos (angle);
    treenode.z.y = length * Math.sin (angle);

    // translate the point and the wedge by the parents coordinates
    treenode.z.translate(parent.z);
    wedge.translate  (parent.z);

    // translate the wedge by the negitive points coordinates
    wedge.translate (-treenode.z.x, -treenode.z.y);
  }

  [ ....... ]

  // layout for the child nodes and their subtrees
  double currentangle = wedge.alpha_;
  for (int i = 0; i < treenode.getChildCount () ; i++)
  {
    TreeNode childnode = treenode.getChildNode (i);
    HBWedge childWedge = new HBWedge();

    // calculate the wedge for the child
    childWedge.alpha_ = currentangle;
    currentangle +=
      (wedgeangle * (childnode.weight / treenode.childrenweights));
    if (currentangle > 2 * PI)
      currentangle -= 2 * PI;
    childWedge.omega_ = currentangle;

    // layout the child node within its wedge
    layoutHBNode (childnode, childWedge, length);
  }
}
```

Listing 6.2: The tree layout algorithm which recursively lays out the hierarchy in hyperbolic space.

```
/**
 * Recursive method which computes the weights of the nodes
 */
protected void computeWeight (TreeNode treenode)
{
  for (int i = 0; i < treenode.getChildCount () ; i++)
  {
    TreeNode child = treenode.getChildNode (i);

    // compute weight recusively for all descendants
    computeWeight (child);

    // add the weight to this node
    treenode.childrenweights += child.weight;
  }

  // use the logarithmic value for further calculation
  if (treenode.childrenweights != 0.0)
    treenode.weight += Math.log(treenode.childrenweights);
}
```

Listing 6.3: The recursice method which computes the weights of the nodes.

needed if an inner node has a great number of children, because they are placed to close to each other in this case.

An improvement could be made in the layout algorithm, by assigning dynamic wedge radii to the nodes depending on the number of their children. A formula would be needed which calculates a reasonable radius value by using the number of children. In this case manual modification of magnification during navigation would be obsolete.

6.4.2 Rendering Optimisations

Label and icon sizes depend on their positions in the drawing disc. The further they are located from the origin, the smaller they are visualised. On the one hand, this represents levels of detail and on the other hand it is a necessity to avoid cluttering near the rim. An improvement could be made in the algorithm which calculates the size values. Sometimes they seem too small depending on the tree structure and the magnification magnitude. The algorithm should consider both conditions.

Moving a large tree often causes parts of the tree to disappear in order to guarantee a fixed frame rate. Drawing nodes, labels, and lines takes much more time than calculating their positions. Therefore an optimisation of rendering of moving trees might modify the decision, as to which parts are drawn and which are not. Currently a fixed order determines the sequence of nodes, labels and lines to be drawn. If the focused part of the tree is at the end of this sequence, it disappears while moving. A more sophisticated solution would be to start the drawing with the part lying near the disc origin and successively draw the elements at more distant positions. Since the magnified focus area only contains a small part of the tree, this technique would avoid the disappearance of relevant tree information.

Chapter 7

Magic Eye Browser

This chapter describes the implementation of the Magic Eye browser and its integration into HVS. The Magic Eye method is a focus plus context visualisation technique, based on a graph layout. In this chapter the visualisation technique is explained, followed by an description of the browser implementation and some implementation details.

7.1 Magic Eye Layout Technique

The Magic Eye technique is intended to have similar properties than the hyperbolic method, but uses a different approach based on spherical projection. This visualisation technique and an implementation was firstly published in Bürger (1999) and Kreuseler and Schumann (1999) and is called the *Magic Eye View*. An overview of the features and a screenshot is discussed in Section 3.15.

Essentially, the Magic Eye method is based on a mapping of a tree layout to the surface of a hemisphere. Due to the mapping of a plane with euclidean geometry to a plane with spherical geometry, a focus plus context effect is achieved. The mathematical background consists of trigonometric calculations and intersecting 3d vectors.

An orthogonal projection from the surface of a hemisphere to the equatorial disc demonstrates how a focus can be achieved by using spherical geometry. Figure 7.1 shows a projection of several rings on the hemisphere, which are equidistantly arranged from the pole to the equator. Also, the vertical angles of the vectors from the origin to adjacent rings are equidistant. The orthogonal projection to the equatorial disc achieves a focus of the rings near the pole and diminishes the rings near the equator. In this way, magnification around the pole of the hemisphere is achieved.

7.1.1 Tree Layout

The algorithm starts with a radial variation of the Walker tree layout. The x-coordinates are normalised to 2π radians and are used as angles in the layout disc. The levels of the nodes are used as the polar length, which is normalised to $\frac{\pi}{2}$. The lengths of the several node levels are equidistant from one another (see Figure 7.2(a)).

The mapping of the layout on the surface of a hemisphere is the next step. In general, each point on a hemisphere can be determined by two angles. The first angle is θ, which defines the horizontal rotation, the second is ϕ, which defines the vertical rotation. When using the polar angle of the layout disc as θ and the polar length as ϕ, a unique mapping can be determined. In this manner the circular tree layout is mapped onto the hemisphere surface (see Figure 7.2(b)).

To visualise the tree with the focus on the root node, an orthogonal projection of the tree from the hemisphere onto the equatorial disc is applied. Figure 7.1 illustrates the focus resulting from the

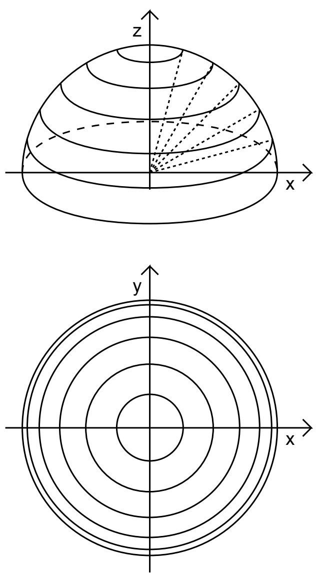


Figure 7.1: An orthogonal projection from the surface of a hemisphere to the equatorial disc is illustrated. The dashed lines symbolise the angles of the projected rings on the hemisphere.

projection. To make clear the terms in this method, the *layout disc* is the disc in which the tree is originally laid out. The *equatorial disc* or *drawing disc* is the disc onto which the tree is projected from the hemisphere.

7.1.2 Changing the Focus

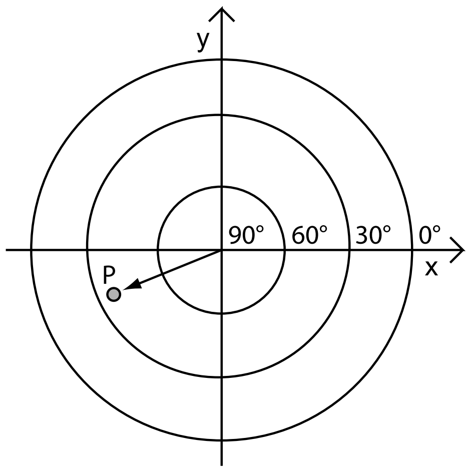
To perform a change of focus, a special point on the equatorial disc is defined, which is called the *projection centre* (PC). Initially the projection centre is placed at the origin of the hemisphere. For each node, a vector is defined which starts at the PC and points to the node. The vectors have the same vertical and horizontal angles regarding the layout disc described above (see Figure 7.3(a)).

The PC can be moved around within the equatorial disc, whereby the vectors keep their spatial directions. By reason of translating vectors, there are new intersection points of the vectors with the hemisphere, which are the new positions of the nodes. Finally, the new nodes are orthogonally projected to the equatorial disc below. A new focus emerges on the opposite side of the PC to the origin (see Figure 7.3(b)).

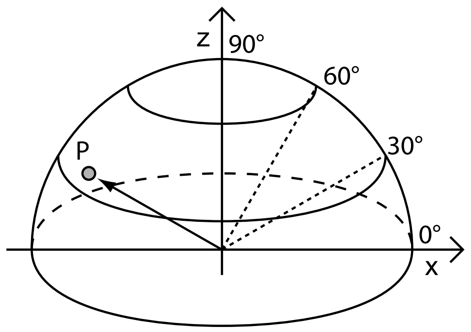
This method of changing the focus is characterised by the fact that all nodes are placed within the drawing disc, even if some nodes are concentrated near the rim. This property resemble the hyperbolic technique, which locates most of the nodes near the circle edge.

7.1.3 Focus Extensions

In addition to the original Magic Eye View three methods have been developed, which increase zooming for the items of interest. All three are applied on the layout calculated for the layout disc. The mapping onto the hemisphere and the projection onto the drawing disc is performed afterwards.

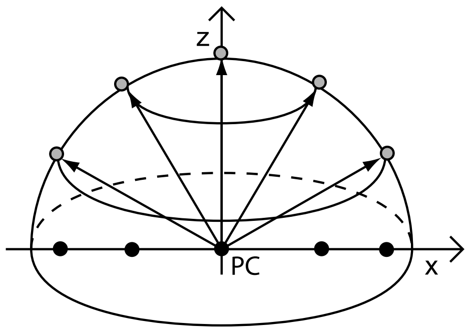


(a) The layout disc where a point is placed.

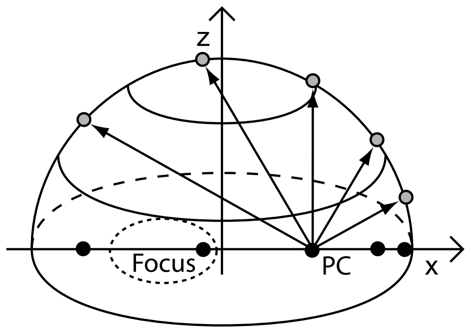


(b) The point with the same location parameters as in the layout disc on the hemisphere.

Figure 7.2: The mapping of a point from the layout disc onto the surface of a hemisphere. The polar coordinates of the point in the left figure are used for the placement on the hemisphere in right figure.



(a) A projection of the nodes with the PC in the origin. The black nodes are the projections of the grey nodes.



(b) A projection of the nodes with the translated PC. The focus has moved on the opposite side. The black nodes are the projections of the grey nodes.

Figure 7.3: Orthogonal projection of nodes. In the left drawing the projection centre (PC) is in the origin. In the right drawing the PC is translated, which causes a focus area on the opposite side of the PC.

Level Zoom

As described above, the ϕ-values in the layout disc are equidistant to each other in the original Magic Eye View method. Increasing a particular ϕ-value moves this level outwards, which leads to a greater ring corresponding to the level represented by the ϕ-value. Therefore, its circumference is enlarged and the nodes on it have more space lying side by side.

Sectoral Zoom

An angular distortion increases the magnification in the focus area. A sector in the layout disc which includes the focus area is defined by two bordering angles. Using the polar coordinate system, an angle is assigned to each node. The angles of all nodes within the sector are distorted to increase the distances of the nodes to one another. Thus the area of these nodes is enlarged and the area of the other nodes shrunk.

Subtree Zoom

A selective layout emphasises the focused node and its subtree. The nodes which are under the focused node are placed on one half of the layout disc, all other nodes on the other half.

7.2 The Magic Eye Browser

The Magic Eye browser implements the visualisation method described in the last section. Like the hyperbolic browser (see Section 6.2) it is based on the Walker layout browser regarding rendering and integration into HVS.

7.2.1 Rendering

In the rendering process most things are identical with the hyperbolic browser, such as drawing nodes, lines, and labels. The guaranteed frame rate of a moving visualisation omits parts of the tree if the time for a frame has elapsed. Nodes near the rim are represented smaller or are hidden.

The visualisation of the levels is a graphical aid in the Magic Eye browser. Since the form of the original layout on the layout disc is essentially preserved, the levels can be visualised by ellipses. They have the shape of ellipses if the PC is moved away from the origin of the equatorial disc, otherwise they are represented by circles. This gives a direct overview of the levels of nodes in different subtrees as can be seen in Figure 7.4.

7.2.2 Navigation

Pan

Panning in the Magic Eye browser means translating the projection centre within the drawing disc. The location of the PC determines the focus area, which always lies on the opposite side of the PC through the disc origin. If the PC is moved to the border, then parts at a successively deeper level are magnified (see Figure 7.4).

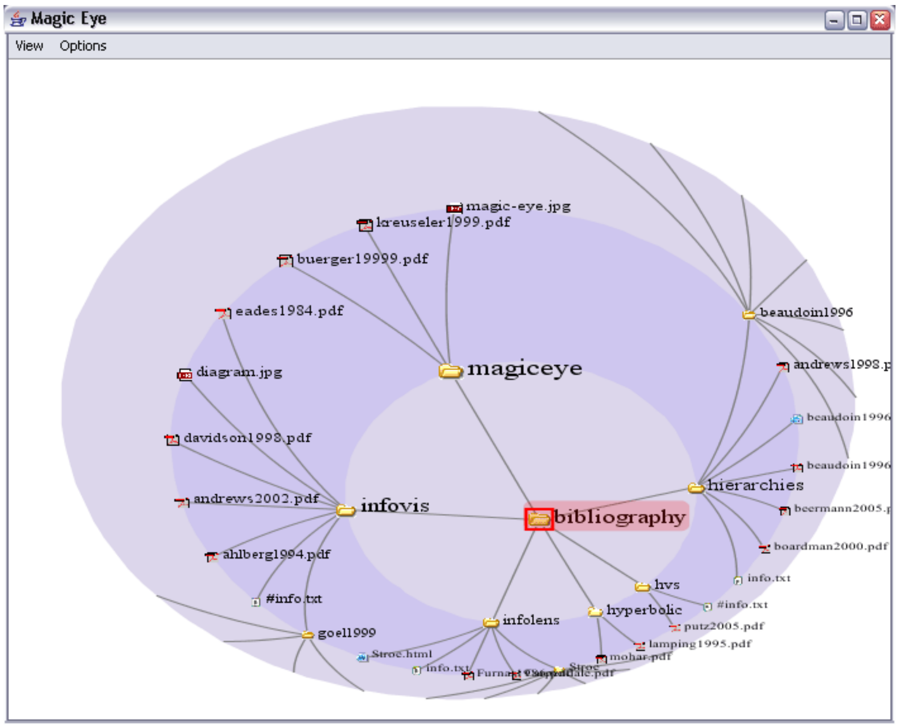


Figure 7.4: The Magic Eye browser. Different levels in the hierarchy are indicated by the bluish rings. The projection centre is translated down and to the right and therefore the focus area has moved up and to the left.

Zoom

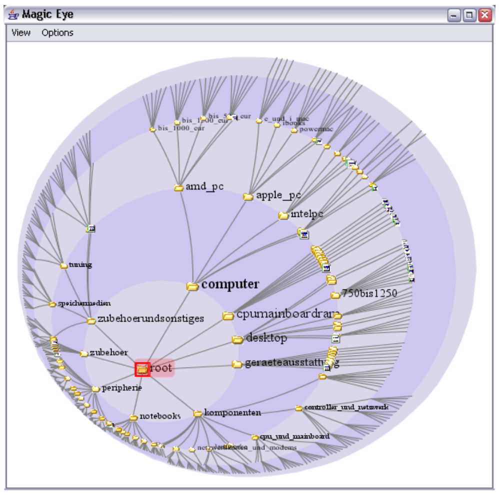
Besides emphasising the focus area by translating the PC, two additional zooming techniques are available, level zoom sectoral zoom (see Figure 7.5(b)) and (see Figure 7.5(d)). Both are extensions to the original Magic Eye View and are explained in Section 7.1.

Level zoom increases the distances between the particular level and adjacent levels. The circumference of the level is enlarged and thus there is more room for the nodes on it. To choose the level which should be magnified, a node on this level has to be focused. Then increasing a level is related to the focused node and the level of its children. Continuously increasing a level is performed with the mouse wheel. The other levels become proportionally smaller (see Figure 7.5(b)).

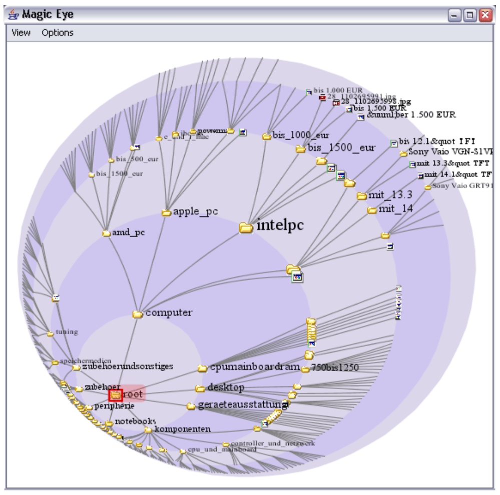
Sectoral zoom is related to the angles of the nodes in a polar coordinate system. Due to the synchronisation with the PC, nodes located in the focus area are fanned out while the other nodes are contracted. The polar lengths of the affected nodes are not modified. Sectoral zoom acts as an amplifier for the focus area, when the PC is dragged around, a higher magnification of the focus area is achieved (see Figure 7.5(d)).

Focus

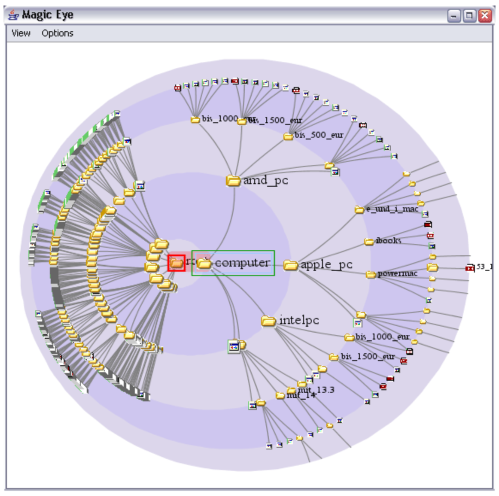
Focusing a node changes the layout, which is also an extension to the original Magic Eye View. The layout is rotated so that the focused node is on the right side of the root node. The subtree of the focus node is placed on the right half of the disc, all other nodes are positioned on the left side (see Figure 7.5(c)). This technique can be used to explore a tree by successively navigating into subtrees. When translating the PC to the left side of the drawing disc, the subtree of the focused node is allocated nearly the whole space of the disc.



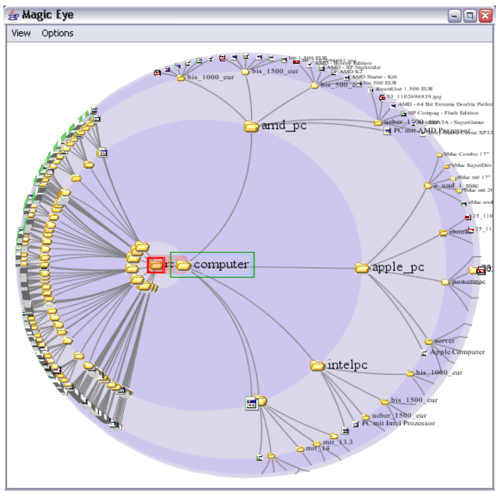
(a) Standard zoom of the focused area.



(b) Sectoral zoom of the focused area.



(c) The focused node and its subtree is zoomed.



(d) Level zoom enlarges the level of the focused node.

Figure 7.5: The Magic Eye browser zooming modes.

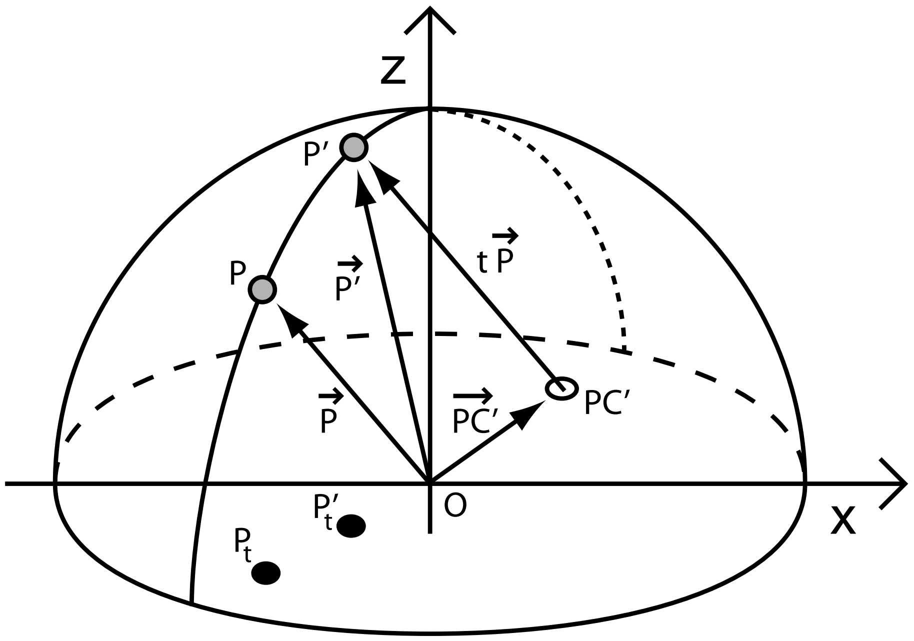


Figure 7.6: The effect on a point on the hemisphere surface after a translation of the projection centre (PC). The vectors to the points are used for the position calculations and the orthogonal projections. The grey points (P, P') are the nodes on the surface, the black points (P_t, P'_t) are the orthogonally projected nodes.

Maximise

Maximising works similar as it does in the hyperbolic browser. An "average" point of the selected nodes is calculated to which the focus area is moved. The PC is moved so that the average point is emphasised. However, it depends on the nodes, how far they are located from the average point, if they are well seen.

7.2.3 Integration into HVS

Integration of the Magic Eye browser is implemented in the same way as the Walker layout browser and the hyperbolic browser. Event handling is implemented similarly and the caused actions are analogue. Due to the graph-based rendering the rendering properties are fetched from the HVS framework and are displayed in the same way as the other browsers.

7.3 Selected Details of the Implementation

7.3.1 Translation of the Projection Centre

As described in Section 7.1, translating the PC affects the position of the nodes in the drawing disc. To obtain the exact positions of the moved nodes, calculation is done with the help of the vector analysis shown in Figure 7.6.

$\vec{P}$ is the vector from the origin to the node with the location P on the hemisphere, which was calculated by the initial layout procedure. A translation of PC to the new location PC' causes a position change of P to P'. The vector $\vec{P'}$ to the new position can be obtained by the sum of the two vectors $\vec{PC'}$ and $t\,\vec{P}$, where t is the scalar factor which changes the size of $\vec{P}$ in order to point to P':

$$\vec{P'} = \vec{PC'} + t\,\vec{P} \tag{7.1}$$

To solve this equation, the value of t is needed. Knowing that the length of $\vec{P'}$ is the same as the radius of the hemisphere, the following equation based on vector lengths can be set up, where t is the only unknown parameter:

$$|\,\vec{P'}\,| = |\,\vec{PC'} + t\,\vec{P}\,| = r \tag{7.2}$$

Since lengths of vectors are scalar numbers, this equation can be transformed to a polynomial of second degree. Then the solution for t is given by the following quadratic equation:

$$t \;=\; \frac{-b \pm \sqrt{b^2 - 4\,a\,c}}{2\,a}\;,\;\; where \tag{7.3}$$

$$\begin{aligned} a &= p_x^2 + p_y^2 + p_z^2 \\ b &= 2\,(p_x\,pc'_x + p_y\,pc'_y + p_z\,pc'_z) \\ c &= {pc'_x}^2 + {pc'_y}^2 + {pc'_y}^2 - r^2 \end{aligned}$$

At this point, the position of P' can be calculated by using the value of t in Equation 7.1. Then P' is projected to the equatorial line by setting the z-coordinate to zero.

The implementation of this algorithm is shown in Listing 7.1. The nodes are placed on the hemisphere surface according to their angles in the layout disc, then the value of t is calculated with the help of Equation 7.3 and finally the new position is computed using Equation 7.1. Projection to the drawing disc is done by omitting the z-coordinate.

7.3.2 Layer-Based Node Positioning Algorithm

Calculating the positions of the nodes in the drawing disc needs a sequence of several layout operations. The various operations, which are described in Section 7.1, are divided into five layers. In each layer the node coordinates calculated by the previous layer are used as input data. Additionally, a layer has specific parameters, which are set by the user or a synchronisation event. Due to the task of a layer, it calculates node positions from its input data using the parameters for its operation. The result is stored in a layer-related field of each node.

Initially, the control algorithm starts with the first layer and processes the layer-related operations step by step until the end. The result values of each layer are stored in the fields of the nodes for fuhrer usage. When the user interacts with the browser, type and value of the interaction is applied on the appropriate layer, whereby the input data come from the stored result of the previous layer. Then the control algorithm traverses all layers below the current one.

For example, dragging the PC with the mouse causes a new transformation and projection of the points, using the actual position coordinates of the PC as parameter. The zoom operations of the layers above are not affected, so the stored result of the previous layer is used as data input.

7.4 Outlook and Further Work

The three extensions to the original Magic Eye technique provide more zooming alternatives. Since they are applied manually and individually, the user has to adjust the view during browsing through the hierarchy. It would be an improvement to integrate them in the drag mechanism, providing automatic adjustment.

```
// a node, its parent, and their phi values
TreeNode treenode, parentnode;
double nodephi, parentphi;

// polynom parameters for the quadratic equation
double ppa, ppb;
double ppc =
  projectioncenter_.x_ * projectioncenter_.x_
  + projectioncenter_.y_ * projectioncenter_.y_
  + projectioncenter_.z_ * projectioncenter_.z_
  - RADIUS*RADIUS;

double t;
double px, py, pz;  // the postion of a node
double phi, theta;  // the angles of a node

// compute all node positions
for (int i = 0; i < treegraph_.getTreeNodeCount (); i++)
{
  treenode = treegraph_.getTreeNode (i);
  parentnode = treenode.getParentNode ();

  if (parentnode == null)
    parentnode = treenode;

  // --- node ---

  // get the angles of the current node
  phi = phicalculator_.getPhi (treenode.getLevel ());
  theta = treenode.ppsectionzoom_.angle_;

  // calculate spatial coordinates of the node on the
  // hemisphere (with pc in the origine)
  px = RADIUS * Math.cos(phi) * Math.cos(theta);
  py = RADIUS * Math.cos(phi) * Math.sin(theta);
  pz = RADIUS * Math.sin(phi);

  // do the transformation
  ppa = px * px + py * py + pz * pz;
  ppb = 2 * (projectioncenter_.x_ * px
    + projectioncenter_.y_ * py + projectioncenter_.z_ * pz);
  t = (-ppb + Math.sqrt (ppb*ppb - 4*ppa*ppc)) / (2 * ppa);

  // calculate the points in the plane for rendering purpose
  treenode.renderpos_.x = projectioncenter_.x_ + t*px + RADIUS + OFFSET;
  treenode.renderpos_.y = projectioncenter_.y_ + t*py + RADIUS + OFFSET;

  [ ... ]

}
```

Listing 7.1: Orthogonal projection of the translated projection centre.

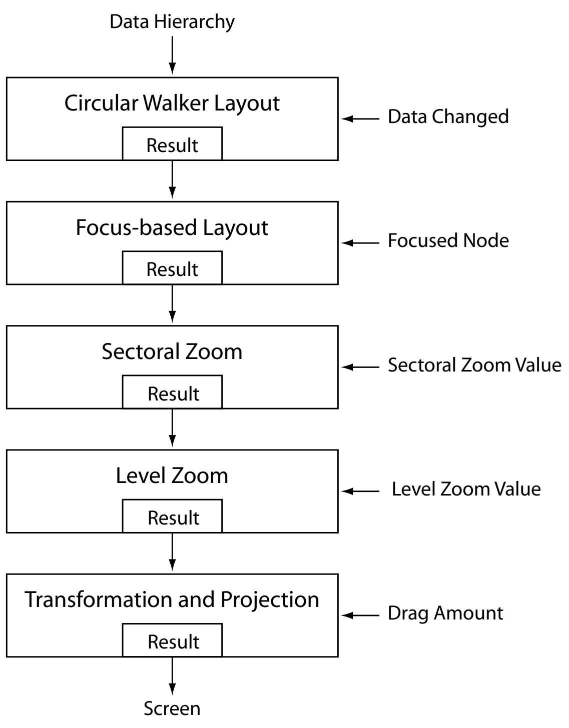


Figure 7.7: The five layers of the node positioning algorithm.

In contrast to the hyperbolic browser, there are some difficulties when navigating through a large hierarchy, though additional zoom facilities were added to the original Magic Eye technique. Maybe further research in the technique will lead to improvements of exploring large hierarchies.

Chapter 8

InfoLens

This chapter describes a new information visualisation browser called *InfoLens*. In some aspects it resembles the Magic Eye and the hyperbolic browser. However it tries to overcome some difficulties with those browsers.

Like the Magic Eye Browser it uses the technique of projecting a tree graph with radial layout onto the equatorial disc. In contrast to it, InfoLens supports dragging the graph around the hemisphere, not just the projection centre in the equatorial disc. Additionally, it uses the technique of fish-eye distortion to support a better zoom on the items of interest.

The following sections describe in detail the mathematical basics of projecting and distorting and how these techniques are used in the InfoLens browser. Integration into HVS is done similarly to the other three browsers described in the previous chapters.

8.1 InfoLens Layout Technique

The basic idea of the InfoLens browser was developed by the author of this thesis during the implementation of the Magic Eye browser. The goal was to improve the Magic Eye browser by some navigational aids which could not be integrated into the Magic Eye technique. Therefore, a new browser was created which differs from the Magic Eye browser in the following ways:

- Instead of the projection centre, the tree is dragged and rotated around the hemisphere.
- The tree layout is adjusted to the right, navigation through the tree is done in this direction.
- An improved zoom facility based on a fish-eye technique achieves better magnification.
- A fan out effect during navigation into the hierarchy is aesthetically pleasing for the user.
- The focus area is on a fixed position in the equatorial disc.
- A more intuitive navigation facility is provided.

Most of these properties are inspired from the hyperbolic browser, but they are implemented in a completely different way. In the focus area of the hyperbolic browser edges and angles are magnified. This is imitated in InfoLens by applying a fish-eye distortion to the polar lengths and angles of nodes.

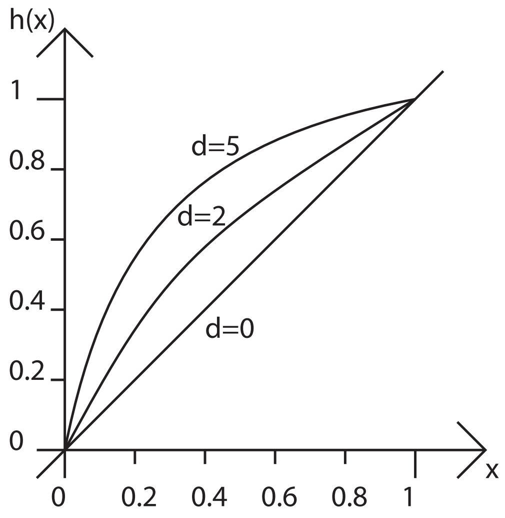


Figure 8.1: A fish-eye distortion function with different distortion factors.

8.1.1 Fish-Eye View

Initially, the term *fish-eye* was introduced by Furnas (1986). Furnas describes the problem that there is too much to show on a small display (which was a 24x80 character display at that time). As a result of moving the window around by scrolling, the user can easily loose orientation easily. His idea was to use an analogy of a zoom lens, which makes available both a global and a detailed view of a structure. He denotes the fundamental motivation of a fish-eye strategy as to provide a balance of local detail and global context. Humans also use this strategy to represent large structures in their heads. For example, people in a large company know local department heads, but only the vice presidents of remote divisions. (Furnas, 1986).

Obviously, fish-eye distortion is a *focus+context technique*, as described in Section 2.1. Fish-eye views enlarge areas of interest and show other parts in less detail. The calculation of magnification of specific areas is based on a distortion function.

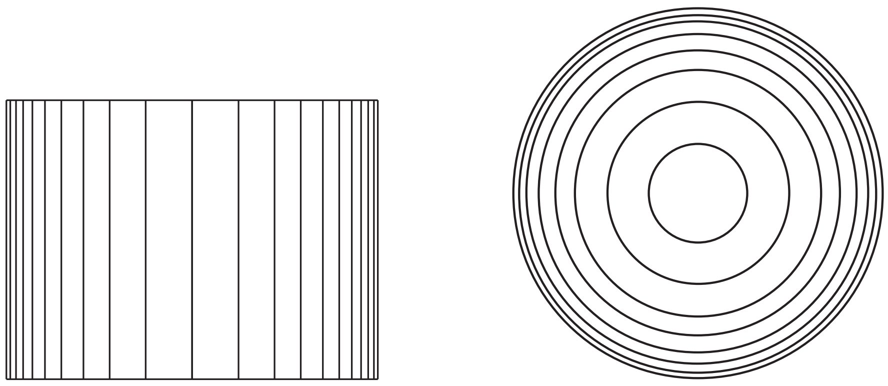
Herman et al. (2000) describe the mathematics of distortion functions. The distance of each point from the focus is distorted by a function *h(x)*. The function maps the interval [0,1] onto [0,1], whereby points which are located near zero are mapped away from zero. For example,

$$h(x) = \frac{d+1}{d+\frac{1}{x}} \tag{8.1}$$

fulfils this criterion and is plotted in Figure 8.1. The parameter *d* is the *distortion factor*, which defines the degree of magnification

The effect on images with fish-eye distortion is shown in Figure 8.2. The two images show grids with focus points in the middle. The left image is a demonstration of a distortion in one direction. The right image shows a polar distortion in two dimensions. In a cartesian fish-eye, distortion is applied independently on the x and y direction.

Carpendale and Montagnese (2001) describe six different lens types (see Figure 8.3). A lens type is mainly defined by its distortion function. The images in the figure show different degrees of magnification in the focus area and in the area near the edges of the distorted region. Also the change of magnification from the focus centre towards the edges of the focus area depends on the distortion function.


(a) Cartesian distortion in one direction.

(b) Polar distortion in two directions.

Figure 8.2: Two fish-eye distorted grids: The left grid is distorted in one dimension, the right is a polar distortion in two dimensions.

Another characteristic of a lens type is the visual integration into the non-distorted context. For example, the Gaussian lens better preserves the edges of the context than the other lens types. A smooth integration provides better orientation for the user if the focus point is moved.

Herman et al. (2000) distinguish between *graphic* and *semantic fish-eye* views. A graphic fish-eye technique is related to the level of pixels of an image. Whereas a semantic fish-eye operation is applied on the graph layout. In the second case only the position of graph elements are distorted, not the visual representations of the graph symbols (such as nodes and edges). Figure 8.4 shows a graph with a distorted layout.

8.1.2 Tree Layout

As in the Magic Eye technique the tree is laid out radially into a disc using the Walker algorithm (see Chapter 5). However, only one quarter of the layout disc is used for the layout, whereby the tree is placed on the right side of the disc origin. Then, following the Magic Eye technique, a mapping of the tree on the surface of a hemisphere is performed. The polar angles of the nodes in the layout disc are used as horizontal rotation angles in the hemisphere and the polar lengths are used as the vertical rotation angles in the hemisphere (see Section 7.1). Afterwards the tree on the hemisphere is orthogonally mapped onto the equatorial disc, which is also called drawing disc (see Figure 8.5). The orthogonal projection is contrary to the Magic Eye technique, in which the tree is projected regarding a projection centre.

8.1.3 Changing the Focus

To move the tree around is done by dragging and rotating it on the hemisphere. For the purpose of ease of use, moving on the hemisphere is restricted to two transformation types, which is illustrated in Figure 8.6. Firstly, the tree can be rotated around the z-axis with the maximum of 90 degrees in both directions (see Figure 8.6(a)). Secondly, the tree can be rotated around the y-axis with the maximum of 90 degrees in one direction (see Figure 8.6(b)).

The focus area in the drawing disc lies on the right side of the tree, which results from the orthogonal projection and from the fish-eye distortion described below. Using both rotation kinds, each part of the tree can be moved into the focus area. In the drawing disc the rotation around the y-axis appears as a translation along the x-axis of the plane and the rotation around the z-axis appears as a rotation around

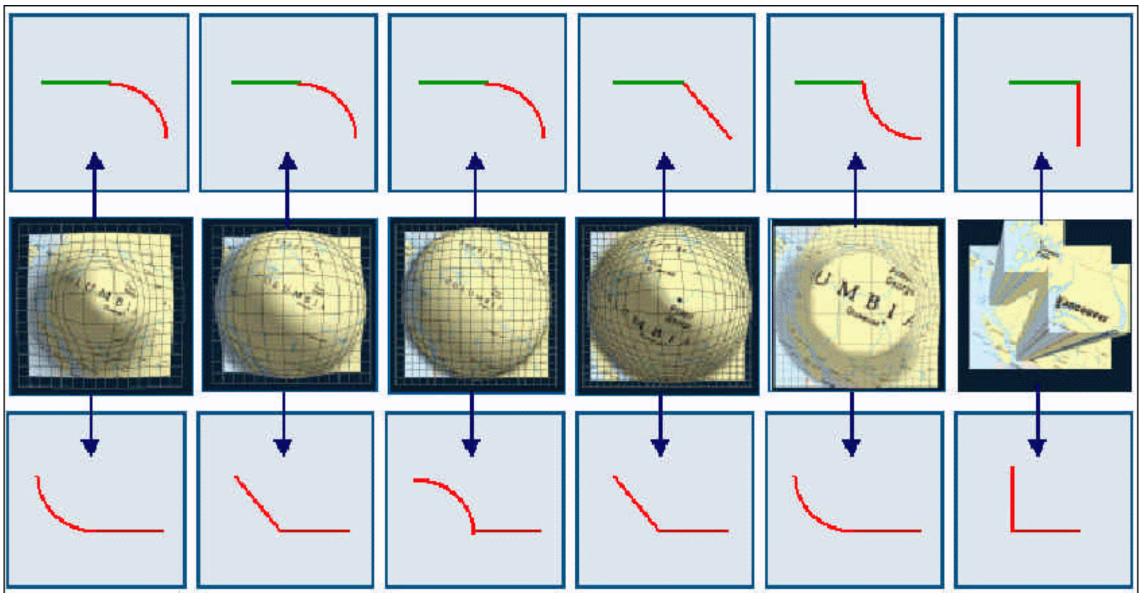


Figure 8.3: Six different lens types: from left to right Gaussian, Cosine, Hemisphere, Linear, Inverse Cosine and Manhattan. The symbols in the top row indicate the profile of the transition from focus centre to distortion area, the bottom row from the distortion area to the context. [Figure extracted from Proc. of UIST '01. Copyright by the Association of Computing Machinery, Inc.]

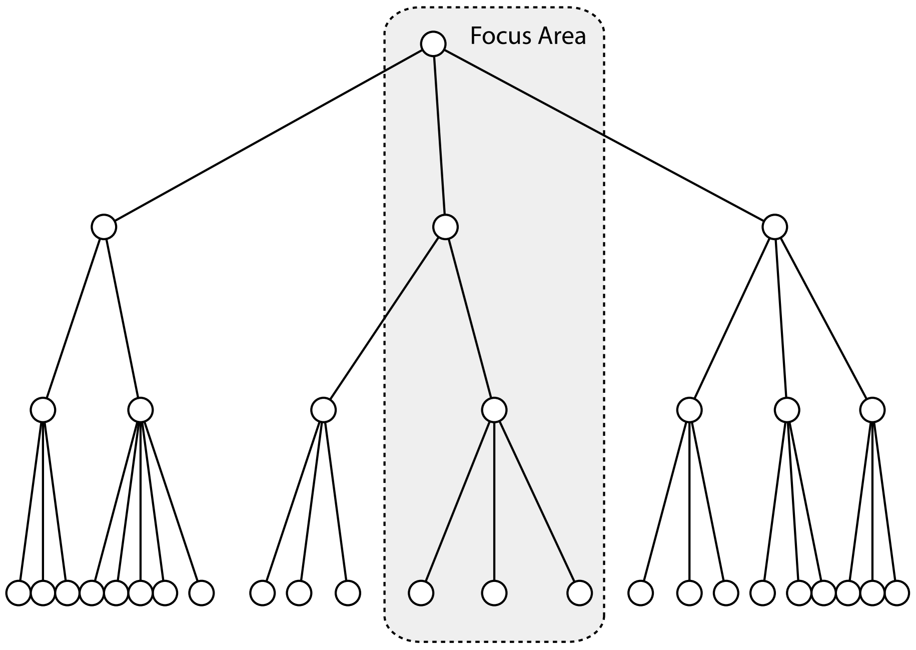


Figure 8.4: A graph with a layout which is distorted along the x-axis, which is also an example for a Cartesian distortion.

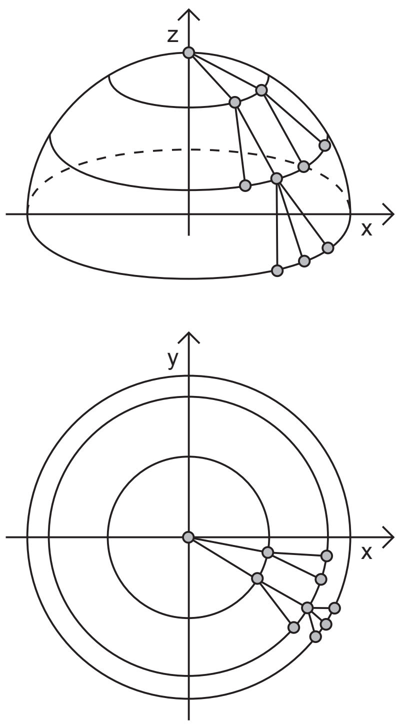
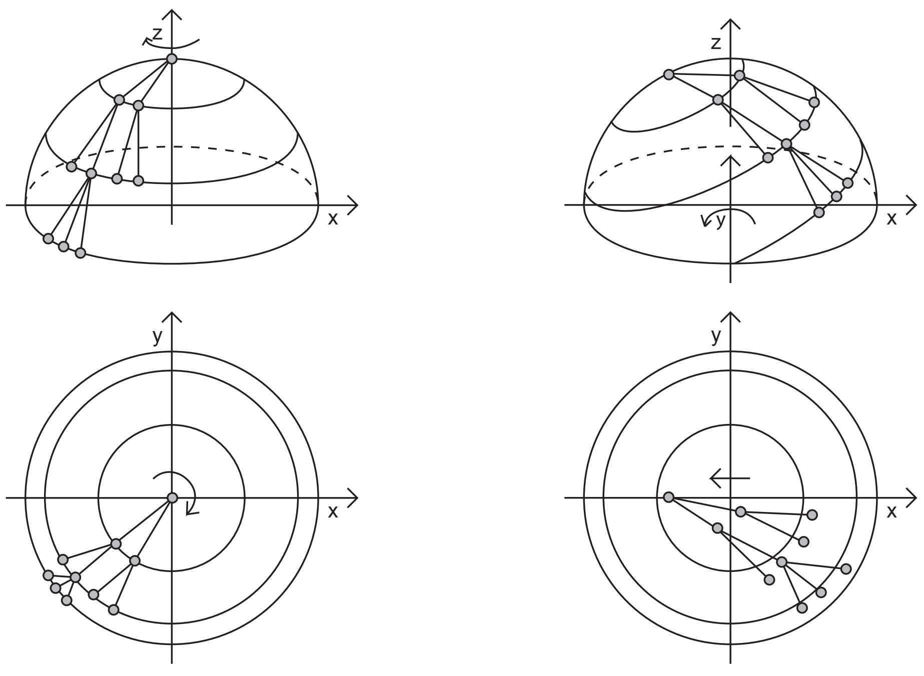


Figure 8.5: A tree laid out on a hemisphere is mapped onto the equatorial disc.



(a) Rotation around z-axis.

(b) Rotation around y-axis.

Figure 8.6: A tree is moved around the z-axis in the left image and around the y-axis in the right image.

root node. Exploring the tree is performed by using a combination of both transformation types.

8.1.4 Fish-Eye Distortion

As already mentioned above, the projected tree in the equatorial disc is distorted with a fish-eye algorithm, in order to increase the magnification in the zoom area. A semantic distortion is applied, which does not distort the graphical elements, but only the positions of the nodes. Unlike the fish-eye techniques described in Section 8.1.1, some modifications were made. For the distance metrics the distortion function described in Section 8.1.1 is used.

The fish-eye lens is spread out on the whole drawing disc, which is a similar effect as in the hyperbolic browser. In this manner the distances of nodes to one another become closer the more they are located near the rim. The magnification area makes use of approximately a quarter of the disc area and lies on the right side of the disc centre (see Figure 8.7(b)).

Basically a two-way polar distortion technique is applied on the tree in the drawing disc. Both angles and lengths of nodes are distorted with respect to a distortion origin on the negative x-axis. The distortion origin can be translated and the distortion factor can be modified by the user.

The angle of a node is related to the focus centre point in the disc (see Figure 8.7). The angular distortion centre is the line with an angle of zero, angles with a values between 0 and 180 degree are distorted towards 180 degree, angle with values between 0 and -180 degrees are distorted toward -180 degrees.

The length is defined by the distance from the focus centre (F) to the position of a node (P). Since the focus centre is not positioned in the centre of the disc, the length of the straight line from F to the intersection point (S) with the disc depends on the angle. The greater the angle, the smaller the length, if the FC lies on the left half of the disc. To provide a length distortion, a normalised length is used to be distorted:

$$ln_p = \frac{l_p}{l_s} \tag{8.2}$$

where l_p is the length from the focus origin to P, l_s is the the length from the focus origin to the intersection point S, and ln_p is the normalised length of l_p. The distorted normalised length is:

$$ln'_p = h\left(ln_p\right) \tag{8.3}$$

And the distorted absolute length l'_p is:

$$l'_p = ln'_p \cdot l'_s. \tag{8.4}$$

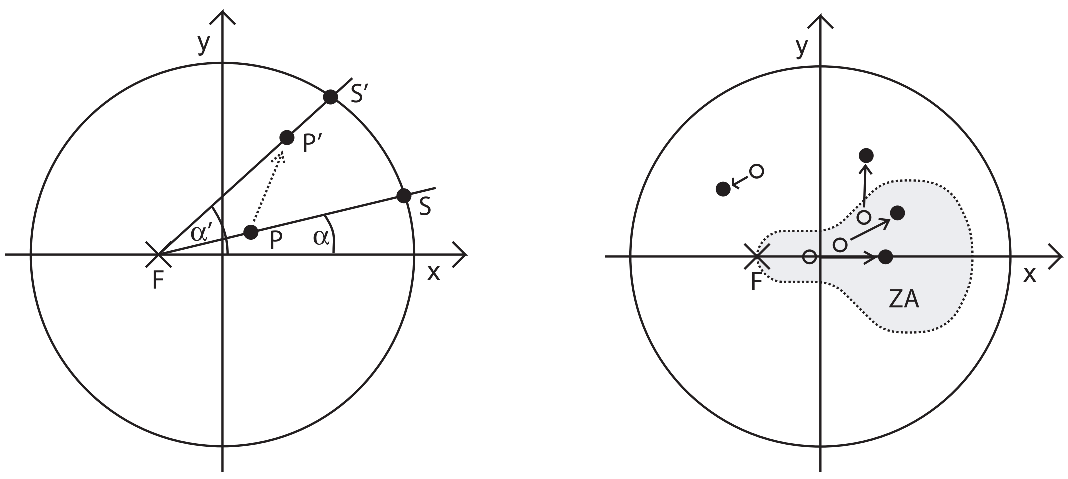
Using both distortion types, a focus area (FA) emerges in the region on the right side of the centre of the disc (see Figure 8.7(b)). Its distinctive pear-shape comes from the two-way distortion.

8.2 The InfoLens Browser

8.2.1 Rendering

Most of the rendering properties are identical with the other browsers. The visualisation of the nodes and the relationships of them are implemented in the same way. A guaranteed frame rate prohibits hanging frames while moving the tree. Label sizes and their disappearance when nodes are near the rim are also similar to the two other focus plus context browsers in the last two chapters.

An additional graphical element in this browser is the visualisation of the fish-eye lens itself. This visual element indicates the location and the magnification magnitude of the lens to the user. It is designed to act as graphical support when dragging interesting parts of the tree into the focus area. The



(a) Polar angle and length distortion. **(b)** Magnified focus area.

Figure 8.7: The InfoLens two-way fish-eye distortion.

size and the position of the lens visualisation correlate with the distortion factor and the distortion origin. It has the form of a pear and is structured with ring areas, whereby the sizes of the areas indicate the magnification magnitude (see Figure 8.9).

8.2.2 Navigation

Zoom and Pan

The InfoLens browser does not allow free transformation, three restrictions determine transformation behaviour. Firstly, the root node always lies on the negative x-axis within the drawing disc. No transformation can move the root out of this location range. Secondly, the tree can be translated to the left and to the right, unless the root resides on the negative x-axis. The tree is adjusted from left to right and the size of the tree is small enough, so that each level can be dragged to the centre of the drawing disc. This rule provides the exploration from the root to the leafs. Thirdly, the tree can be rotated around the root node, depending on its actual position. The maximum rotation angle is from -90 degrees to $+90$ degrees. That way each node can can be rotated onto the positive x-axis. Figure 8.8 shows a tree mapped onto the drawing disc without distortion applied to it.

Owing to the focus plus context property of the InfoLens, there is a focus area on the right side of the disc origin. In order to zoom into a particular part of the tree, this part has to be moved to the focus area using the navigation facilities described above. They provide the ability that every part can be moved into the focus area (see Figure 8.9).

The focus area is a virtual lens realised with a fish-eye technique. There are two properties of the lens which define the distortion, the distortion factor and the distortion origin. As described in the last section, points are distorted according to their lengths and their angles with respect to the distortion origin. The distortion factor determines the magnification magnitude of both the lengths and the angles. This factor can be modified by the user with the mouse wheel, in oder to zoom in deeper into interesting parts of the tree. The distortion origin determines the location of the focus area. Both properties are visualised with the pear-shaped lens contour described above.

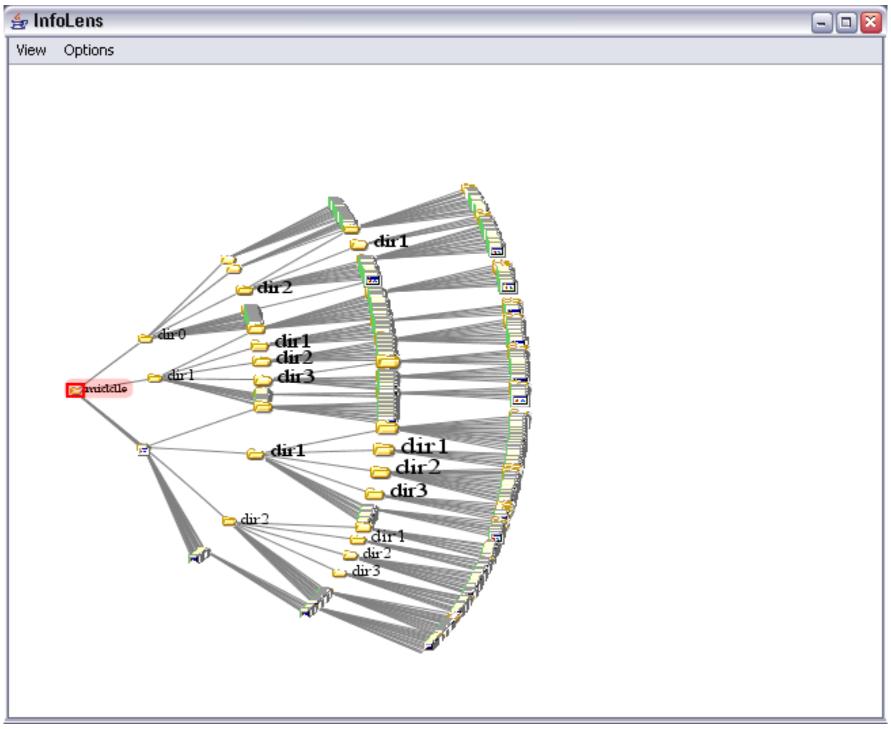


Figure 8.8: A tree mapped onto the drawing disc without distortion.

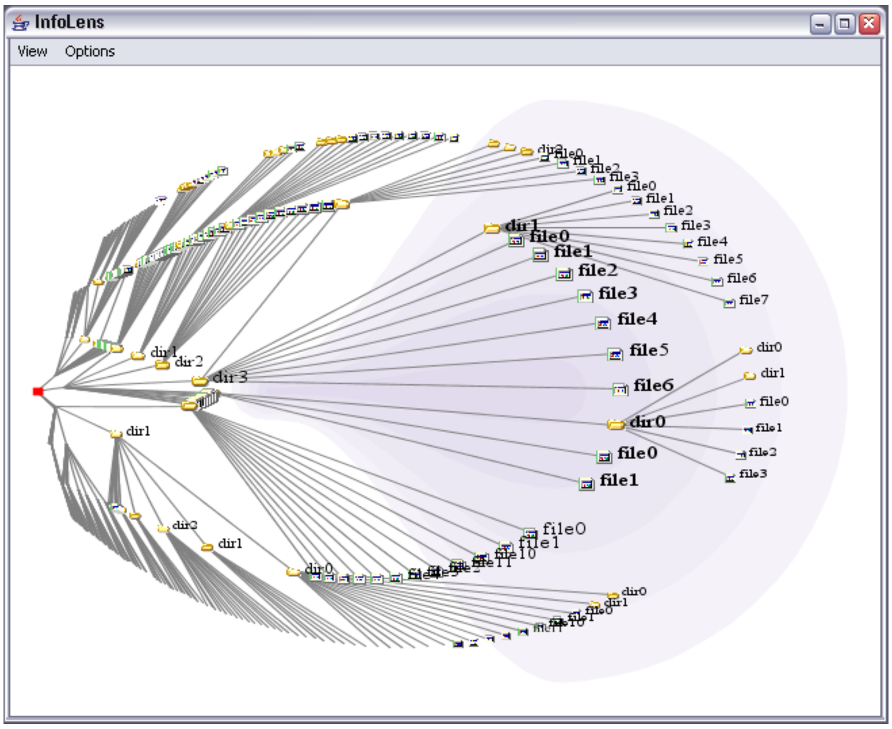


Figure 8.9: A tree mapped onto the drawing disc and distorted by the fish-eye lens. The extent of the lens is visualised by "rings" which also define the zoom area.

Focus

The node which is focused by the user is moved into the centre of the focus area. This is done by a smooth animation, so users do not loose orientation. The automatic moving behaves in the same way as the moving caused by user interaction. The tree is translated along the x-axis and rotated around the root node, in order to position the focused node into the lens centre.

Maximise

The maximisation of selected nodes is based on the same principle as the Magic Eye browser and the hyperbolic browser. An average position of the selected nodes is calculated, then the tree is transformed in order to place the average position into the focus centre.

8.2.3 Integrating into HVS

The InfoLens is integrated into HVS in the same way as the other browsers discussed above. Event handling and rendering properties are processed analogously. Navigation, selection, focus, search result, and data model changes are synchronised with the opened visualisations.

8.3 Selected Details of the Implementation

8.3.1 Fish-Eye Distortion

As described in Section 8.1, the hierarchy in the drawing disc is distorted twice. Considering that nodes are determined by polar coordinates, the angle and the length of each node is translated using the distortion function. The implementation of the transformation of a point is done in the method *distortPoint* in the class *ILGeometryEngine* shown in Listing 8.1.

Firstly the angle of the straight line from PC to P is calculated, followed by the intersection point S of this line with the circle. Regarding S, the normalised length from PC to P is computed, which is distorted just as the angle. Then new new intersection point S and its length from the PC are calculated. The new length of P is obtained from the multiplication of the normalised length with the length to S. The now available angle and length of P are used to compute the cartesian coordinates.

```
/**
 * Distorts a point by using the distortValue method.
 * The result is stored in the parameter point
 * @param ppoint The point to be distorted, will contain the result.
 */
protected void distortPoint (PolarPoint2D ppoint)
{
  double sx, sy, k, d, sflength, pflength, pfnormlength;

  // get cartesian coordinates of the original point
  double px = ppoint.getPosX ();
  double py = ppoint.getPosY ();

  // get angle of the original point
  double alpha = Math.atan (
    Math.abs (py) / Math.abs (px - fisheyecenterx_));
  if (px - fisheyecenterx_ < 0)
    alpha = Math.PI - alpha;
  if (py < 0)
    alpha = -alpha;
    // calculate intersection point s on the circle
    k = py / (px - fisheyecenterx_);
    d = py - (px * py) / (px - fisheyecenterx_);
    if (Math.abs (alpha) < Math.PI / 2.0)
      sx = (-k*d + Math.sqrt (k*k - d*d + 1)) / (k*k + 1);
    else
      sx = (-k*d - Math.sqrt (k*k - d*d + 1)) / (k*k + 1);
    sy = k*sx +d;

    // calculate the normalised length of p regarding the circle
    sflength = Math.sqrt (
      (sx-fisheyecenterx_)*(sx-fisheyecenterx_) + sy*sy);
    pflength = Math.sqrt (
      (px-fisheyecenterx_)*(px-fisheyecenterx_) + py*py);
    pfnormlength = pflength / sflength;

    // distort angle
    alpha /= Math.PI;
    if (alpha >= 0)
      alpha = distortValue (alpha, TYPE_ANGLE);
    else
      alpha = -distortValue (-alpha, TYPE_ANGLE);
    alpha *= Math.PI;

    // distort normalised length
    pfnormlength = distortValue (pfnormlength, TYPE_LENGTH);

    // calculate absolute length of p
    if (Math.abs (alpha) < Math.PI)
      px = 1;
    else
      px = -1;
    py = px * Math.tan (alpha);
    px += fisheyecenterx_;
    k = py / (px - fisheyecenterx_);
    d = py - (px * py) / (px - fisheyecenterx_);
    if (Math.abs (alpha) < Math.PI / 2.0)
      sx = (-k*d + Math.sqrt (k*k - d*d + 1)) / (k*k + 1);
    else
      sx = (-k*d - Math.sqrt (k*k - d*d + 1)) / (k*k + 1);
    sy = k*sx +d;
    sflength = Math.sqrt (
      (sx-fisheyecenterx_)*(sx-fisheyecenterx_) + sy*sy);
    pflength = sflength * pfnormlength;

    // calculate cartesian coordiantes of p and store it in polar point
    px = pflength * Math.cos (alpha) + fisheyecenterx_;
    py = pflength * Math.sin (alpha);
    ppoint.setCartesian (px, py);
}
```

Listing 8.1: The implementation of the two-way fish-eye distortion.

The implementation of the distortion function is shown Listing 8.2. The distortion factor is a member variable, which is modified with the mouse wheel. The distortion can potentially be influenced by a type, such as angle and length. However, this is currently not implemented.

```
/**
 * The distortion function. Used for a single value in [0, 1].
 * The function uses the private member zoomlevel_ as factor.
 * @param value The value to be distorted.
 * @param type The type, which influences the distortion strength
 *   (currently not implemented).
 * @return The distorted value.
 */
protected double distortValue (double value, int type)
{
  if (value < EPSILON && value > -EPSILON)
    return 0;

  return (zoomlevel_ + 1) / (zoomlevel_ + (1.0 / value));
}
```

Listing 8.2: The implementation of the distortion function.

8.4 Outlook and Further Work

The zoom factor of the lens can be modified, but there is a magnification limit, which determines the maximum amount of distortion. Trees grow exponentially with the depth, whereas the circumference of the layout disc grows only linearly with the radius. Thus the nodes of a higher level are placed more closely to one another. A goal of future work would be to find a dynamic solution, which places nodes automatically distantly enough independent of their levels.

Another improvement could probably done in applying the fish-eye distortion. Currently, the distortion is applied on the projected tree, whereby node positions are changed without knowledge about the structure of the tree. Applying the distortion on the tree in the layout disc depending on the tree structure could bring a different result. In this case the distortion would be applied before the mapping onto the hemisphere and projection onto the layout disc.

Chapter 9

Comparison of Hierarchical Browsers

This chapter gives an overview of the strengths and weaknesses of the visualisations described in this thesis. Since all visualisations are implemented as views running in HVS, usability studies could be performed. However, here the browsers are compared regarding factors such as tree layout, focus plus context effect, handling of large trees, and navigation. An overview of the most important properties, strengths, and weaknesses of the four browsers is given in Table 9.1.

9.1 Walker Layout Browser

The Walker layout is a classic tree drawing, which lays out the tree from top to bottom without distortion. The layout algorithm is designed to be aesthetically pleasing, which is significant in terms of human perception. Nodes and subtrees are spaced out evenly and parents are placed centrally above their children. This helps users to better understand the tree structure. In general, the classic tree drawing facilitates an intuitive understanding of the tree.

As long as trees are small enough, users can see the whole tree and every detail at the same time. If trees are large, then the nodes have to be drawn very close to each other and structural details disappear. To increase a particular part of the tree, the whole tree is magnified, whereby most parts are outside the viewing plane, because the tree is zoomed evenly. In large trees either an overview without detail, or detail without overview and context can be seen.

The traditional navigational tools zoom and pan are implemented in the Walker layout browser. Zooming is done with the mouse wheel, sliders at the sides, or by drawing a zooming rectangle over the interesting part of the tree. Since the drawing area of the tree is increased, the disappearing parts have to be explored with the pan functionality. The user can pan the tree with the mouse in order to explore it.

9.2 Hyperbolic Browser

The hyperbolic browser uses a radial layout, whereby the space allocated to subtrees depends on the number of nodes. In euclidean geometry this approach is not as effective as the Walker layout regarding the usage of space. However, the tree is laid out in hyperbolic space, which offers sufficient room even for this layout. The exponential growth of space with the radius is the main property of this browser, because it can contain an exponentially growing tree.

The Poincaré disc offers infinite space, therefore trees can be completely drawn inside this disc, independent of their size. In contrast to the Walker layout browser, trees are not scaled evenly, but they are hardly scaled in the central area and strongly scaled near the rim. In this way, a focus plus context effect is achieved.

	Waker Layout	Hyperbolic	Magic Eye	InfoLens
Radial layout		○	○	○
Tree always fully in drawing area		○	○	○
Intuitive overview of small trees	+	+	+	+
Overview of large trees	−	○	+	○
Focus plus context	−	+	+	+
Permanent focus area		+	−	+
No manual zoom needed	−	+	−	−
Fan out effect	−	+	−	+
Exploring very large trees	−	+	−	−

Table 9.1: The table shows an overview of the compared items of the browsers. A plus sign (+) indicates a strength, a minus sign (−) a weakness, and a circle (○) the availability of a property. An empty cell means that this property is not applicable to the respective browser.

Due to the focus plus context property of the hyperbolic geometry, zoom and pan are combined to one pan tool. When the user drags the tree around, the part of the tree lying in the centre of the disc is magnified, the rest is scaled down. An automatic zoom while dragging is achieved in this way, whereby the zoom is always strong enough to visualise all details. Subtrees which are dragged into the focus area fan out their children the more they are placed centrally, which is an aesthetic and elegant way of zooming into details. If a node has a very great number of children, then they are placed very close to each other, because even in hyperbolic space a circle has only a finite amount of room on it.

9.3 Magic Eye Browser

Layout in the Magic Eye browser is performed radially, whereby the Walker layout algorithm is used. The more effective layout is needed since room is not as available as in hyperbolic geometry. A focus plus context effect is achieved by an orthogonal projection from a hemisphere.

The Magic Eye technique is suitable for small and medium-sized trees, but has problems with large trees. Since the tree is always mapped onto a constant-sized hemisphere and projected to the equatorial disc, large trees must be laid out closer. The focus area achieved by the orthogonal projection magnifies a part of the tree, however, this magnification factor is not sufficient for large trees. Some zooming techniques are integrated into the original Magic Eye technique, however, they require manual interaction by the user.

In contrast to the hyperbolic browser, the focus area is moved instead of the tree. Zooming is performed by panning the focus area, in this way the tree can be explored. While navigating, the global overview of the tree is preserved better than in the other browsers. To overcome the problem of a too low magnification magnitude, the tree layout can be modified by focusing particular nodes. Then the subtree of this node obtains half the disc for the layout of its children. This method can be used for successively browsing into a subtree. The disadvantage of this method is the loss of global overview and the need for additional manual interaction with the browser.

9.4 InfoLens

The InfoLens layout technique is similar to the Magic Eye technique. The tree is laid out radially using the Walker layout algorithm, whereby the tree is adjusted to the right side using only a quarter of the layout disc. Then it is mapped onto the hemisphere and projected onto the equatorial disc. Additional to the Magic Eye technique, a polar two-way fish-eye distortion is applied on the projected tree to increase magnification of the focus area.

Due to the fish-eye distortion, larger trees can be explored with this method. The user can control the magnification magnitude with the mouse to adapt it for the tree size. However, the InfoLens also has a limit regarding the tree size, since magnification can not be increased infinitely.

Navigation is performed by dragging the tree from right to left. The part which lies currently in the central part of the drawing disc is magnified. Similar to the hyperbolic browser, the lengths and the edges of the focused part of the tree are increased, which produces a fan out effect. Zooming into a particular part is done by dragging this part into the focus area, which is always located in the middle of the drawing disc.

Chapter 10

Concluding Remarks

Information visualisation provides an interface to abstract information spaces, which have no natural 2d or 3d geometry. It transforms abstract data into a form which can be recognised and understood by humans. Information is classified regarding the structure of data, which can be linear, hierarchical, networked, multi-dimensional, or content-based vector spaces. There are numerous visualisation techniques for each type, whereby this thesis focused on hierarchical information visualisation.

An overview of diverse hierarchy visualisation methods indicates the comprehensive and lively nature of research in this topic. Each visualisation technique is qualified with one or more classification terms and explained using implementation examples. For instance, there are outliners, graph-based diagrams, radial views, inclusive and space-filling techniques, distorted oriented methods, and visualisations in 3d space.

To ease the implementation of hierarchy viewers the Hierarchical Visualisation System (HVS) was developed as an extensible framework. HVS fulfils several tasks which are not directly related to the visualisation, such as managing hierarchal data and loading and synchronisation of the multiple visualisations. Viewers which are developed to be integrated into HVS benefit from theses properties, their implementation focuses on the visualisation method and the interface to HVS.

The development of four graph-based viewers are presented in the second part of this thesis. For each technique the technical background is explained, followed by an detailed description of the applied technique. The navigational and rendering features resulting from the according visualisation method and implementation details are explicated.

The Walker layout browser is a classic tree drawing viewer using the Walker layout algorithm. Navigation is implemented with the traditional zoom and pan tools, which leads to navigational difficulties for large trees. Furthermore this browser is an implementation basis for the other browsers, since the development of the HVS interface and the rendering of the node-link model is equal to all other browsers.

The hyperbolic browser implements a technique based on hyperbolic geometry. A simple tree layout leads to an excellent visualisation result when it is laid out in the hyperbolic plane. A focus plus context effect is achieved, whereby the focus can be moved to every part of the tree using a transformation formula. Nodes can be fanned out at each level in order to explore the hierarchy.

The Magic Eye browser makes use of a focus plus context technique based on the spherical projection of a tree onto the 2d equatorial circle. The focus is changed by moving the projection centre within this circle. An overview of the tree structures is always kept, however, exploring large trees leads to difficulties. Three additional techniques are presented, which advances the original Magic Eye View.

The InfoLens essentially implements the ideas of the Magic Eye browser, but modifications are made to achieve better navigation properties with large trees. A fan out effect similar to the hyperbolic browser is achieved by the use of a two-way fish-eye distortion, which is applied to the projected tree.

Appendix A

User Guide

This chapter describes the usage of the visualisations within HVS. Since all browsers are plug-in visualisations, HVS has to be installed and started first. A user guide for HVS was published in Putz (2005, Appendix A).

A.1 HVS Integration

A.1.1 Installation

To install the four browsers in HVS, a folder with arbitrary name has to be created in the HVS *plugin* directory. Then the two files *graphview.jar* and *HvsPlugin.xml* have to be copied (or extracted) into this folder. The file *graphview.jar* contains all Java class files of the four applications. The file *HvsPlugin.xml* is the plug-in configuration file, which connects the visualisations with HVS and which has the following content (see Lising A.1).

A.1.2 Starting

In the File menu of HVS a dialogue box can be opened which lists all plug-in visualisations (see Figure A.1). Each listed browser can be chosen, and is then opened. A data source has to be opened first.

A.1.3 HVS Options Menu

Global rendering settings can be chosen in the HVS *Options* menu (see Figure A.2). These settings affect all currently opened visualisations in HVS. The graph-based views process these settings in the same way.

Colour: This menu item opens the colour dialogue, where colours for the selection boxes, the search result boxes, the selected search result boxes, and the node types can be chosen (see Figure A.2).

Font: The Font menu item opens die font dialogue, where the font type and size of the labels can be set (see Figure A.3).

Icons: The Icons menu item opens the icons dialogue, where the icons for the document types can be chosen in a list box (see Figure A.3).

Display Icons: This menu item controls whether the nodes are represented by coloured circles or by document type icons.

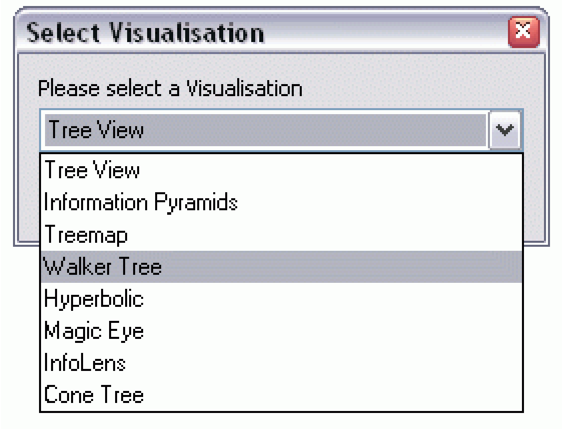
```
<plugin>
  <runtime>
    <library name="graphview.jar"/>
    <path name="./" />
    <extension
      name="Walker Layout"
      point="iicm.hvs.visualization.Visualization"
      class="iicm.hvs.visualization.graphview.basictree.BasicTreeHvsView"
    />
  </runtime>

  <runtime>
    <library name="graphview.jar"/>
    <path name="./" />
    <extension
      name="Hyperbolic"
      point="iicm.hvs.visualization.Visualization"
      class="iicm.hvs.visualization.graphview.hyperbolic.
          HyperbolicHvsView"
    />
  </runtime>

  <runtime>
    <library name="graphview.jar"/>
    <path name="./" />
    <extension
      name="Magic Eye"
      point="iicm.hvs.visualization.Visualization"
      class="iicm.hvs.visualization.graphview.magiceye.MagicEyeHvsView"
    />
  </runtime>

  <runtime>
    <library name="graphview.jar"/>
    <path name="./" />
    <extension
      name="InfoLens"
      point="iicm.hvs.visualization.Visualization"
      class="iicm.hvs.visualization.graphview.infolens.InfoLensHvsView"
    />
  </runtime>
</plugin>
```

Listing A.1: The content of the plug-in configuration file HvsPlugin.xml, which determines the integration of a visualisation into HVS.



Select Visualisation

Please select a Visualisation

Tree View

- Tree View
- Information Pyramids
- Treemap
- Walker Tree
- Hyperbolic
- Magic Eye
- InfoLens
- Cone Tree

Figure A.1: The dialogue to open a new visualisation. The four graph-based browsers are listed beside other visualisations.

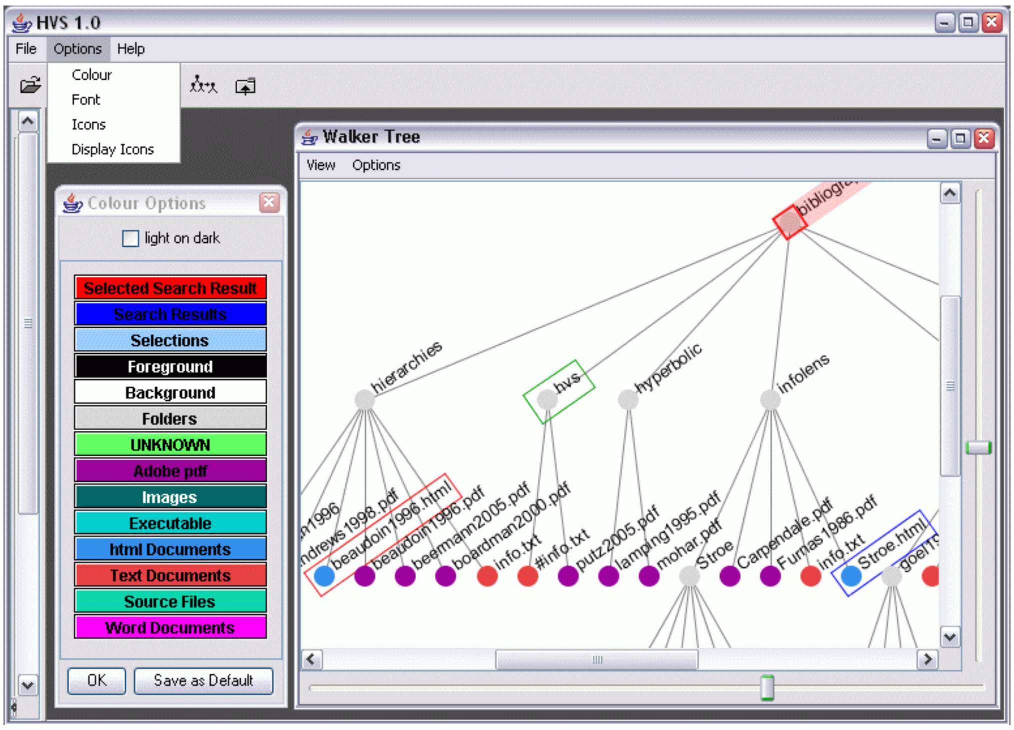


Figure A.2: The HVS colour settings dialogue and the effect on the nodes. Colours for the various node types and the selection boxes can be chosen in the dialogue. In the visualisation window these colours are used for the rendering.

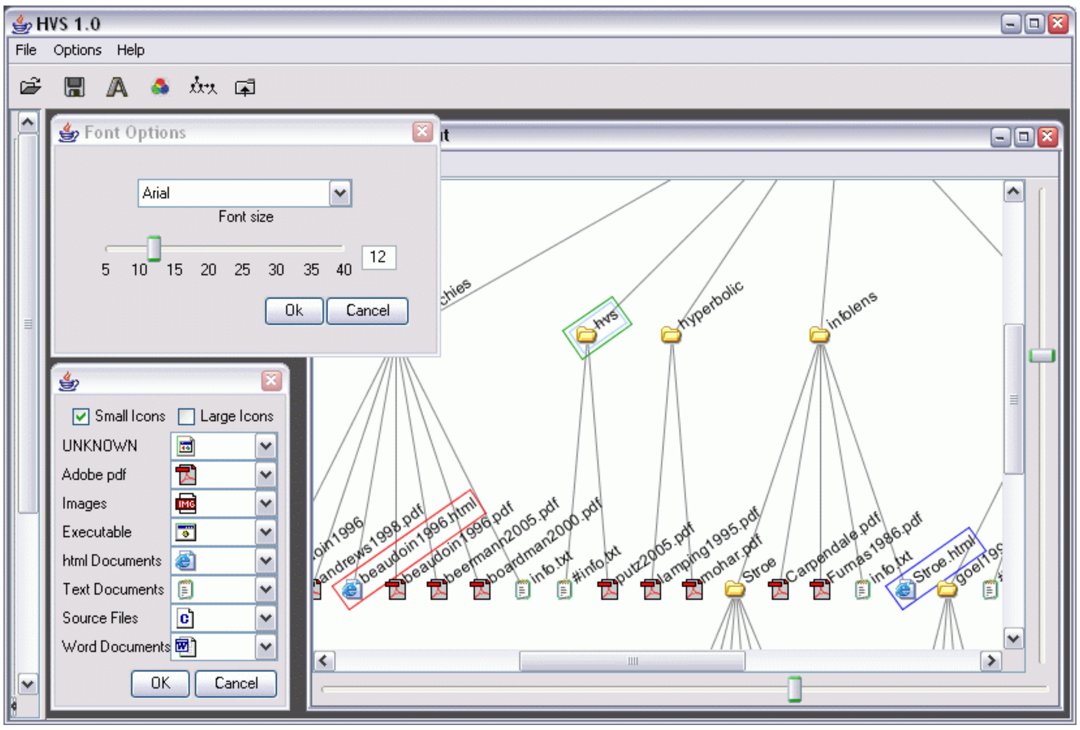


Figure A.3: The HVS font and icon settings and the effect on the nodes. Font type and size and icons for the various node types can be chosen in the dialogues on the left side. In the visualisation window these settings are used for the rendering.

A.1.4 Visualisation View Menu

The *View* menu of the visualisation frame is provided by HVS and is the same in all browsers (see Figure A.4(a)). The menu items and their meanings are as follows:

Independent: Determines the synchronisation mode of the respective visualisation. If the mode is turned on, navigational actions in other visualisations do not affect this one, otherwise the view is fully synchronised.

Detailed View: Determines the filtering state of the hierarchy data. If it is turned on, then a new root is set which is the smallest common parent of all currently selected nodes, otherwise the root is the root node of the chosen data source.

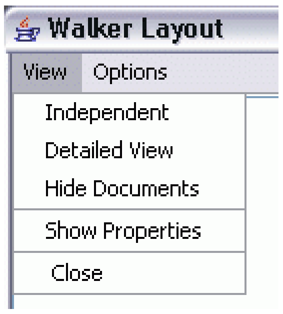
Hide Documents: Determines weather documents should be displayed or not. If turned on, the document nodes are temporary removed from the hierarchy data.

Show Properties: Determines if the properties panel on the bottom of the visualisation frame should be shown or not. The properties panel contains a table of all selected nodes and their attributes.

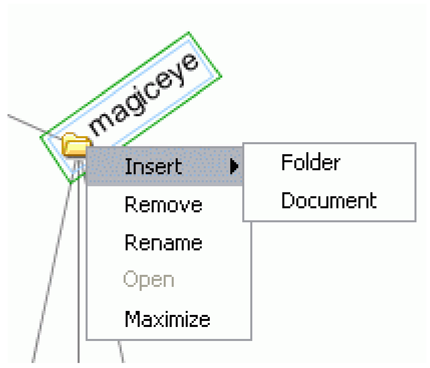
Close: Closes the view.

A.1.5 Visualisation Context Menu

The context menu of the visualisation is provided by HVS to modify the hierarchy data (see Figure A.4(b)). They are enabled, if the user clicks with the right mouse button on a node. If the click



(a) The view menu is used for synchronisation and filter settings.



(b) The context menu is used for data manipulation and maximisation of the currently selected nodes.

Figure A.4: The view and context menus of a visualisation are provided by HVS.

hits no node, the menu items are disabled. The last item *Maximise* is handled differently by the respective visualisation.

Insert: A new folder or document can be inserted under the clicked folder. If a document was clicked, HVS refuses this operation.

Remove: The clicked node will be removed from the hierarchy data.

Rename: The clicked node can be renamed in a dialogue.

Open: If the hit node is a document, then this document will be opened by an external program.

Maximise: A maximisation of the selected nodes will be done by the visualisation (see the section of the mouse buttons for each view.

A.2 Walker Layout Browser

The Walker layout browser lays out a hierarchy using the classic tree drawing algorithm of Walker (see Figure A.3).

A.2.1 Options Menu

The *Options* menu of the Walker layout browser is shown in Figure A.5. The particular menu items are explained in the following list:

Show Tooltips: If this option is turned on, a tooltip are shown, when the mouse is moved over a node. The tooltip gives information about the name, the size, and other attributes of nodes.

Show Labels: If this option is turned on, the names of each node is drawn beside the node icons.

Highlight Root Node: If this option is turned on, the root node is highlighted, in order to keep orientation.

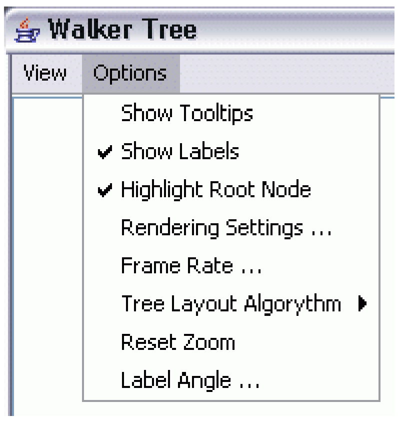


Figure A.5: The options menu of the Walker layout browser is used for rendering and navigation settings.

Rendering Settings: Opens the rendering settings dialogue, in which the following render settings can be chosen (see Figure A.6): line colour, line width, label colour, root highlight colour, and label distance threshold value. The last one determines the distance value, when labels are hidden if the nodes are too near.

Frame Rate: Opens a dialogue with the render frame rate setting, which determines the guaranteed number of frames to be drawn per second.

Tree Layout Algorithm: Beside the Walker layout algorithm, there is a second tree layout algorithm called *Simple* for demonstration purposes. In this layout, each node allocates the same space independent of the number of its descendants. Obviously the result is rather bad, which should show the advantages of the Walker layout.

Reset Zoom: This menu item resets the current zoom to the default value, which shows the whole tree in the window.

Label Angle: A dialogue is opened in which the rotation angle of the labels can be set. The same angle can be changed with the mouse wheel.

A.2.2 Mouse Functions

Left Button:

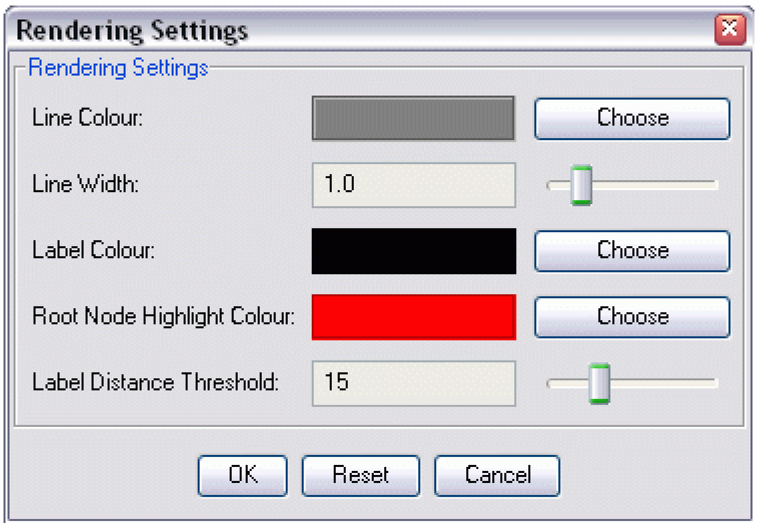
Single Click: Selects the clicked node and deselects all other nodes.

Control + Click: Toggles the selection state of the clicked node, the selection states of the other nodes are not affected.

Double Click: Focuses the clicked node, which means that this node is moved to the centre of the window and a synchronisation event is sent to the HVS framework.

Drag: A selection box is painted from the start point to the current mouse point. When the mouse button is released, all nodes within this box are selected and the other nodes are deselected (see Figure A.7).

Middle Button:


Rendering Settings

Rendering Settings

Line Colour:	▨	Choose
Line Width:	1.0	
Label Colour:	■	Choose
Root Node Highlight Colour:	🟥	Choose
Label Distance Threshold:	15	

Figure A.6: The rendering settings dialogue, which is provided by all graph-based browsers to modify label and line rendering.

Single Click: If the clicked node is a document, it will be opened with an external program.

Drag: A zooming box is painted from the start point to the current mouse point. When the mouse button is released, this area is zoomed.

Right Button:

Single Click: The context menu is popped up, which is described in Section A.1. The *Maximise* menu item activates the maximisation of the selected nodes. In this browser, the area used by these nodes is zoomed.

Drag: Moves the tree around.

Mouse Wheel:

Normal Rotation: A normal rotation zooms in to the point where the mouse is currently located.

Control + Rotation: Rotates the labels.

A.2.3 Scroll Bars and Sliders

Scroll Bars: The vertical and horizontal scroll bars are used to move the tree. The size of the bars indicates the current zoom factor.

Sliders: The sliders are used to zoom the tree, whereby the zoom centre is the centre of the window.

A.3 Hyperbolic Browser

The hyperbolic browser lays out the tree in the Poincaré disc, which offers infinite space. Figure A.7 shows a hyperbolic layout with a currently drawn selection box.

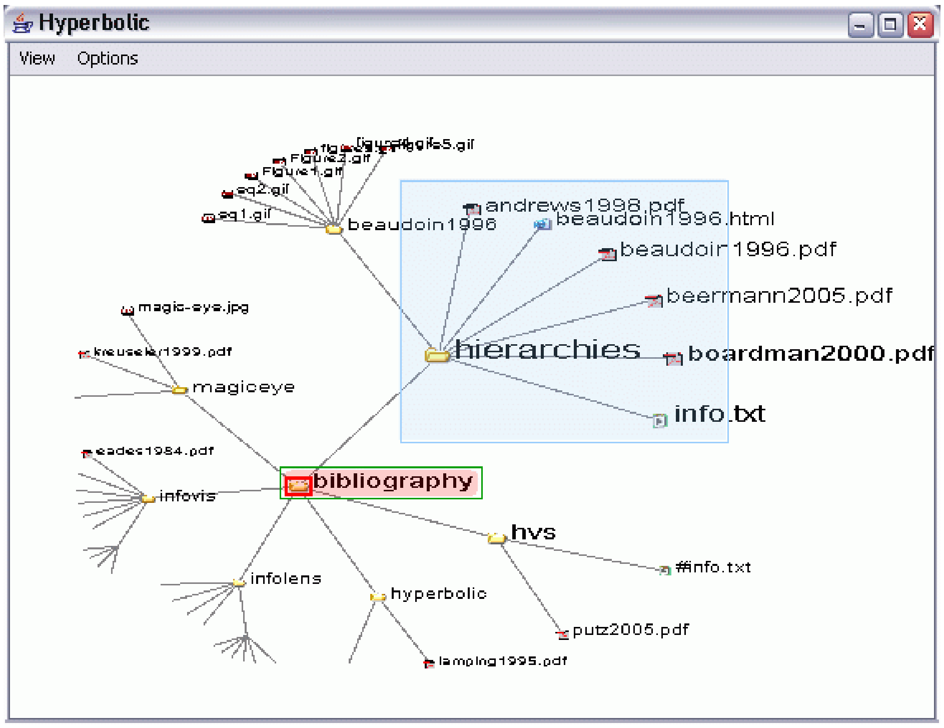


Figure A.7: A tree laid out in the hyperbolic browser. The box is the selection box, which is currently dragged over some nodes.

A.3.1 Options Menu

The *Options* menu of the Hyperbolic browser is shown in Figure A.8. The particular menu items are explained in the following list:

Show Tooltips: If this option is turned on, a tooltip are shown, when the mouse is moved over a node. The tooltip gives information about the name, the size, and other attributes of nodes.

Show Labels: If this option is turned on, the names of each node is drawn beside the node icons.

Highlight Rootnode: If this option is turned on, the root node is highlighted, in order to keep orientation.

Render Settings: Opens the render settings dialogue, in which the following render settings can be chosen (see Figure A.6): line colour, line width, label colour, root highlight colour, and label distance threshold value. The last one determines the distance value, when labels are hidden if the nodes are too near.

Framerate: Opens a dialogue with the render frame rate setting, which determines the guaranteed number of frames to be drawn per second.

Tree Layout Direction: Determines the adjustment of the tree. The radial layout means, that the children are placed 360 degree around the root. If one of the other layout options are chosen, the children are placed on a quarter of a disc around the root, either to the right, to the left, to the top, or to the bottom.

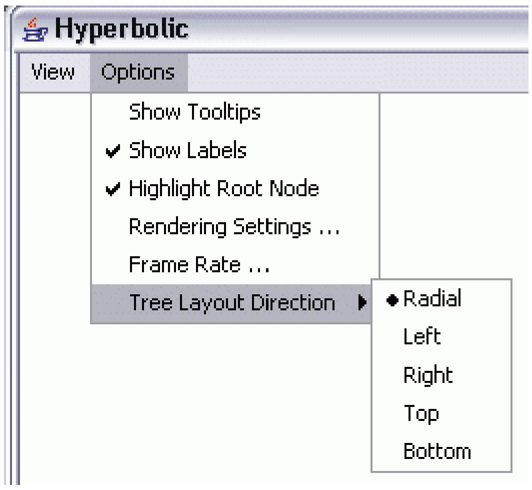


Figure A.8: The options menu of the hyperbolic browser, where rendering and tree layout settings can be made.

A.3.2 Mouse Functions

Left Button:

Single Click: Selects the clicked node and deselects all other nodes.

Control + Click: Toggles the selection state of the clicked node, the selection states of the other nodes are not affected.

Double Click: Focuses the clicked node, which means that this node is moved to the centre of the window and a synchronisation event is sent to the HVS framework.

Drag: A selection box is painted from the start point to the current mouse point. When the mouse button is released, all nodes within this box are selected and the other nodes are deselected (see Figure A.7).

Middle Button:

Single Click: If the clicked node is a document, it will be opened with an external program.

Right Button:

Single Click: The context menu is popped up, which is described in Section A.1. The *Maximise* menu item activates the maximisation of the selected nodes. In this browser, the geometric average point is moved to the centre.

Drag: Moves the tree around.

Mouse Wheel:

Normal Rotation: A normal rotation changes the zoom factor, which means that the length of the lines between the nodes are increased or decreased. The longer a length is, the wider nodes are placed from each other. The currently focused node is fixed on its position, the other nodes change their position according the modification of the zoom factor.

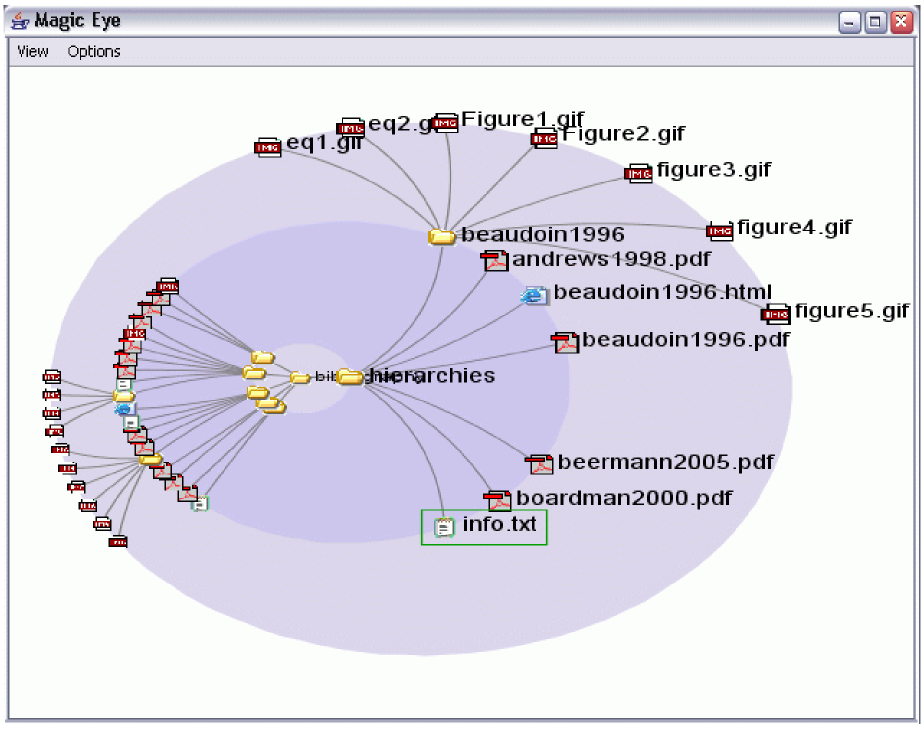


Figure A.9: A tree is laid out in the Magic Eye browser. A node is focused and its descendants allocate the right side of the drawing circle.

A.4 Magic Eye Browser

The Magic Eye browser visualises a hierarchy using a modified spherical projection (see Figure A.9).

A.4.1 Options Menu

The *Options* menu of the Magic Eye browser is shown in Figure A.10. The particular menu items are explained in the following list:

Show Tooltips: If this option is turned on, a tooltip are shown, when the mouse is moved over a node. The tooltip gives information about the name, the size, and other attributes of nodes.

Show Labels: If this option is turned on, the names of each node is drawn beside the node icons.

Highlight Rootnode: If this option is turned on, the root node is highlighted, in order to keep orientation.

Render Settings: Opens the render settings dialogue, in which the following render settings can be chosen (see Figure A.6): line colour, line width, label colour, root highlight colour, and label distance threshold value. The last one determines the distance value, when labels are hidden if the nodes are too near.

Framerate: Opens a dialogue with the render frame rate setting, which determines the guaranteed number of frames to be drawn per second.

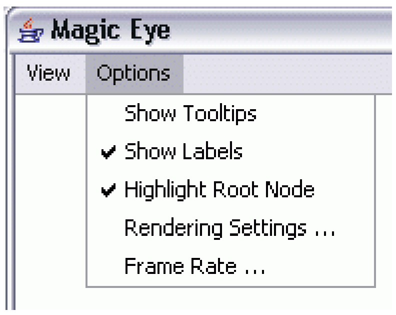


Figure A.10: The Options menu of the Magic Eye Browser, where rendering settings can be made.

A.4.2 Mouse Functions

Left Button:

Single Click: Selects the clicked node and deselects all other nodes.

Control + Click: Toggles the selection state of the clicked node, the selection states of the other nodes are not affected.

Double Click: Focuses the clicked node, which means that this node is moved to the centre of the window and a synchronisation event is sent to the HVS framework. Furthermore, the descendants of the focused node are placed on the right side and all other nodes are placed on the left side of the focus node.

Drag: A selection box is painted from the start point to the current mouse point. When the mouse button is released, all nodes within this box are selected and the other nodes are deselected (see Figure A.7).

Middle Button:

Single Click: If the clicked node is a document, it will be opened with an external program.

Right Button:

Single Click: The context menu is popped up, which is described in Section A.1. The *Maximise* menu item activates the maximisation of the selected nodes. In this browser, the geometric average point is moved to the centre.

Drag: Moves the projection centre and the root node, which changes the location of the focus. In this way the tree can be explored.

Mouse Wheel:

Normal Rotation: Zooms the level of the focused node.

Control + Rotation: Zooms the angle of the focus area.

A.5 InfoLens

The InfoLens browser visualises a hierarchy using a spherical projections and a two-way fish-eye distortion afterwards (see Figure A.11). The fish-eye distortion area is represented as a pear-shape on the right side of the browser window.

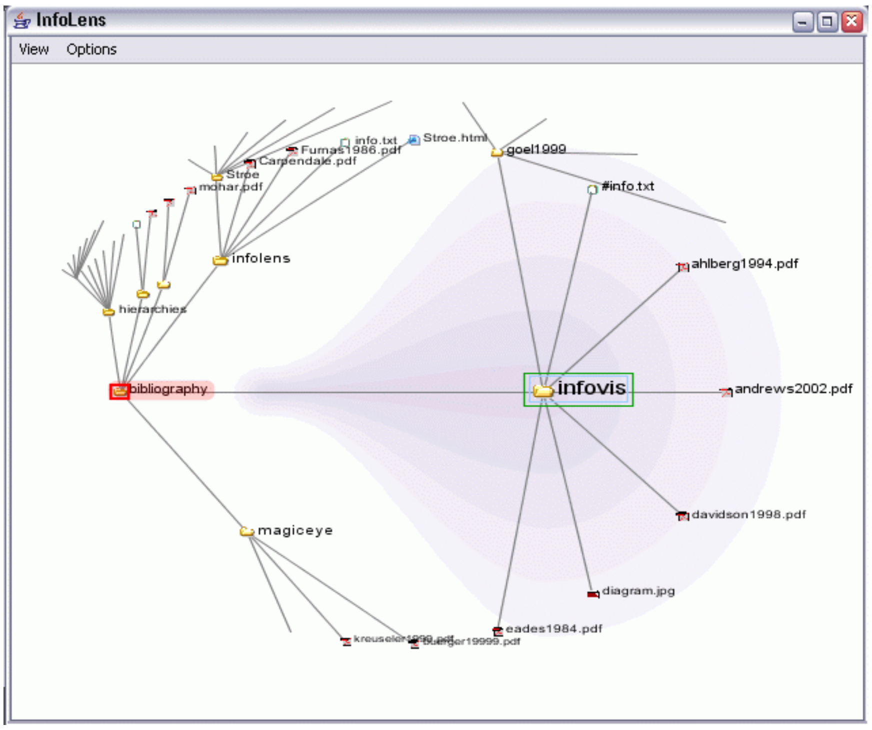


Figure A.11: A tree is laid out in the InfoLens browser. The lens is visualised as a pear-shaped contour outline to indicate the magnification in this area.

A.5.1 Options Menu

The *Options* menu of the InfoLens is shown in Figure A.12. The particular menu items are explained in the following list:

Show Tooltips: If this option is turned on, a tooltip are shown, when the mouse is moved over a node. The tooltip gives information about the name, the size, and other attributes of nodes.

Show Labels: If this option is turned on, the names of each node is drawn beside the node icons.

Highlight Rootnode: If this option is turned on, the root node is highlighted, in order to keep orientation.

Render Settings: Opens the render settings dialogue, in which the following render settings can be chosen (see Figure A.6): line colour, line width, label colour, root highlight colour, and label distance threshold value. The last one determines the distance value, when labels are hidden if the nodes are too near.

Framerate: Opens a dialogue with the render frame rate setting, which determines the guaranteed number of frames to be drawn per second.

Lens Colour: A dialogue in which the colour of the lens can be set.

Reset Zoom: This menu item resets the current zoom to the default value, which leads to an average magnification of the lens area.

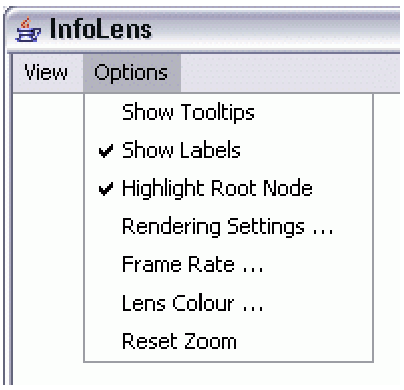


Figure A.12: The Options menu of the InfoLens, where rendering and navigation settings can be made.

A.5.2 Mouse Functions

Left Button:

Single Click: Selects the clicked node and deselects all other nodes.

Control + Click: Toggles the selection state of the clicked node, the selection states of the other nodes are not affected.

Double Click: Focuses the clicked node, which means that this node is moved to the centre of the window and a synchronisation event is sent to the HVS framework.

Drag: A selection box is painted from the start point to the current mouse point. When the mouse button is released, all nodes within this box are selected and the other nodes are deselected (see Figure A.7).

Middle Button:

Single Click: If the clicked node is a document, it will be opened with an external program.

Right Mouse:

Single Click: The context menu is popped up, which is described in Section A.1. The *Maximise* menu item activates the maximisation of the selected nodes. In this browser, the geometric average point is moved to the centre.

Drag: Moves the tree around. A drag to the left or the right, moves the tree in this direction. A drag to the top or the bottom rotates the tree around the root node.

Mouse Wheel:

Normal Rotation: A normal rotation zooms the lens magnification magnitude and therefore the focus area, which is represented as a pear-shapes contour outline.

Bibliography

Christopher Ahlberg and Ben Shneiderman. *Visual Information Seeking using the FilmFinder*. In: *Conference Companion on Human Factors in Computing Systems (CHI '94)*, pages 433–434. ACM Press, Boston, Massachusetts, USA (1994). DOI:10.1145/259963.260431. (Cited on page 8.)

Keith Andrews. *Visualising Information Structures: Aspects of Information Visualisation*. Professorial thesis, Graz University of Technology, Austria (2002). `ftp://ftp.iicm.edu/pub/keith/habil/visinfo.pdf`. (Cited on pages 4 and 29.)

Keith Andrews. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria (2004). `ftp://ftp2.iicm.edu/pub/keith/thesis/thesis.zip`. (Cited on page iii.)

Keith Andrews and Helmut Heidegger. *Information Slices: Visualizing and Exploring Large Hierarchies using Cascading, Semi-Circular Discs*. In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '98)*, pages 9–11. IEEE Computer Society Press, Research Triangle Park, North Carolina, USA (1998). `ftp://ftp.iicm.edu/pub/papers/ivis98.pdf`. (Cited on page 24.)

Keith Andrews, Michael Pichler, and Peter Wolf. *Towards Rich Information Landscapes for Visualising Structured Web Spaces*. In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '96)*, pages 62–63. IEEE Computer Society Press, San Francisco, California, USA (1996). `ftp://ftp.iicm.edu/pub/papers/ivis96.pdf`. (Cited on page 28.)

Luc Beaudoin, Marc-Antoine Parent, and Louis C. Vroomen. *Cheops: A Compact Explorer for Complex Hierarchies*. In: *Proceedings of the 7th conference on Visualization (Vis '96)*, pages 87–92. IEEE Computer Society Press, San Francisco, California, USA (1996). `http://portal.acm.org/citation.cfm?id=245014`. (Cited on page 24.)

Dale Beermann, Tamara Munzner, and Greg Humphrey. *Scalable, Robust Visualization of Very Large Trees*. In: *Proceedings of the Joint Eurographics and IEEE VGTC Symposium on Visualization 2005 (VisSym 2005)*, pages 37–44. Eurographics Association, Leeds, UK (2005). `http://www.cs.virginia.edu/~gfx/pubs/TJC/tjc.pdf`. (Cited on page 20.)

Richard Boardman. *Bubble Trees: The Visualization of Hierarchical Information Structures*. In: *Extended Abstracts on Human Factors in Computing Systems (CHI 2000)*, pages 315–316. ACM Press, The Hague, The Netherlands (2000). DOI:10.1145/633292.633484. `http://www.iis.ee.ic.ac.uk/~rick/research/pubs/bubbletree-chi2000.pdf`. (Cited on page 18.)

Mark Bruls, Kees Huizing, and Jarke J. van Wijk. *Squarified Treemaps*. In: *Proceedings of the Joint Eurographics and IEEE TVCG Symposium on Visualization (VisSym 2000)*, pages 33–42. Eurographics Association, Amsterdam, The Netherlands (2000). `http://www.win.tue.nl/~vanwijk/stm.pdf`. (Cited on page 21.)

Christoph Buchheim, Michael Jünger, and Sebastian Leipert. *Improving Walker's Algorithm to Run in Linear Time*. In: *Revised Papers from the 10th International Symposium on Graph Drawing (GD 2002)*, pages 344–353. Springer, Irvine, California, USA (2002). `http://portal.acm.org/citation.cfm?id=647554.729576`. (Cited on pages 16, 41 and 42.)

Thomas Bürger. *Magic Eye View: Eine neue Fokus + Kontext Technik zur Darstellung von Graphen*. Master's Thesis, Institute for Computer Graphics, University of Rostock, Germany (1999). `http://wwwicg.informatik.uni-rostock.de/Diplomarbeiten/1999/Thomas_Buerger/`. (Cited on pages 27 and 63.)

Sheelagh Carpendale and Catherine Montagnese. *A Framework for Unifying Presentation Space*. In: *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST 2001)*, pages 61–70. ACM Press, Orlando, Florida, USA (2001). DOI:10.1145/502348.502358. (Cited on page 74.)

George S. Davidson, Bruce Hendrickson, David K. Johnson, Charles E. Meyers, and Brian N. Wylie. *Knowledge Mining with VxInsight: Discovery Through Interaction*. *Journal of Intelligent Information Systems*, 11(3), pages 259–285 (1998). DOI:10.1023/A:1008690008856. `http://www.cs.sandia.gov/projects/VxInsight/pubs/jiis98_prepub.pdf`. (Cited on page 10.)

Peter Eades. *A Heuristic for Graph Drawing*. *Congressus Numerantium*, 42, pages 149–160 (1984). `http://www.cs.usyd.edu.au/~peter/old_spring_paper.pdf`. (Cited on page 6.)

Peter Eades and Kozo Sugiyama. *How to Draw a Directed Graph*. In: *Journal of Information Processing*, 13(4), pages 424–437. Information Processing Society of Japan (1991). (Cited on page 6.)

Martin Eyl. *The Harmony Information Landscape: Interactive, Three-Dimensional Navigation Through an Information Space*. Master's Thesis, Graz University of Technology, Austria (1995). `ftp://ftp.iicm.edu/pub/theses/meyl.pdf`. (Cited on page 28.)

Kim M. Fairchild, Steven E. Poltrock, and George W. Furnas. *SemNet: Three-Dimensional Representations of Large Knowledge Bases*. In: *Cognitive Science and its Application for Human-Computer Interface*, pages 201–233. Lawrence Erlbaum (1988). (Cited on page 7.)

Jean-Daniel Fekete. *The InfoVis Toolkit*. In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2004)*, pages 167–174. IEEE Computer Society Press, Austin, Texas, USA (2004). DOI:10.1109/INFVIS.2004.64. `http://www.lri.fr/~fekete/ps/ivtk-04.pdf`. (Cited on page 48.)

George W. Furnas. *Generalized Fisheye Views*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23. ACM Press, Boston, Massachusetts, USA (1986). DOI:10.1145/22627.22342. (Cited on page 74.)

Nahum Gershon, Stephen G. Eick, and Stuart Card. *Information Visualization*. *ACM Interactions*, 5(2), pages 9–15 (1998). DOI:10.1145/274430.274432. (Cited on page 3.)

Amit Goel. *Visualization in Problem Solving Environments*. Master's Thesis, Virginia Polytechnic Institute and State University, USA (1999). `http://www.amitgoel.com/vizcraft/docs/masters_thesis.html`. (Cited on page 8.)

Jeffrey Heer, Stuart K. Card, and James A. Landay. *prefuse: A Toolkit for Interactive Information Visualization*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2005)*, pages 421–430. ACM Press, Portland, Oregon, USA (2005). DOI:10.1145/1054972.1055031. `http://jheer.org/publications/2005-prefuse-CHI.pdf`. (Cited on page 7.)

Bob J. Hendley, Nick S. Drew, Andrew M. Wood, and Russell Beale. *Narcissus: Visualising Information*. In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '95)*, pages 90–96. IEEE Computer Society Press, Atlanta, Georgia, USA (1995). DOI:10.1109/INFVIS.1995.528691. (Cited on page 7.)

Ivan Herman, Guy Melancon, Maurice M. de Ruiter, and Maylis Delest. *Latour — A Tree Visualisation System*. In: *Proceedings of the 7th International Symposium on Graph Drawing (GD '99)*, pages 392–399. Springer, Stirin Castle, Czech Republic (1999). `http://homepages.cwi.nl/~ivan/AboutMe/Publications/LatourGD99.pdf`. (Cited on page 16.)

Ivan Herman, Guy Melancon, and M. Scott Marshall. *Graph Visualisation and Navigation in Information Visualisation: A Survey*. *IEEE Transactions on Visualization and Computer Graphics*, 6(1), pages 24–43 (2000). DOI:10.1109/2945.841119. `http://homepages.cwi.nl/~ivan/AboutMe/Publications/StarGraphVisuInInfoVis.pdf`. (Cited on pages 4, 5, 51, 74 and 75.)

Matthew Holton. *Strands, Gravity and Botanical Tree Imagery*. In: *Computer Graphics Forum*, 13(1), pages 57–67. Eurographics Association (1994). (Cited on page 30.)

Java. *Java Technology*. Sun Microsystems, Inc (2005). `http://java.sun.com/`. (Cited on page 35.)

Brian Johnson and Ben Shneiderman. *Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures*. In: *Proceedings of the 2nd conference on Visualization 1991 (Vis '91)*, pages 284–291. IEEE Computer Society Press, San Diego, California, USA (1991). `http://www.cs.umd.edu/hcil/treemaps/`. (Cited on pages 13 and 21.)

Wolfgang Kienreich, Vedran Sabol, Michael Granitzer, Frank Kappe, and Keith Andrews. *InfoSky: A System for Visual Exploration of Very Large, Hierarchically Structured Knowledge Spaces*. In: *Workshop on Knowledge and Experience Management (FGWM 2003)*. Karlsruhe, Germany (2003). `http://km.aifb.uni-karlsruhe.de/ws/LLWA/fgwm/Resources/FGWM03_02_Wolfgang_Kienreich.pdf`. (Cited on page 10.)

Ernst Kleiberg, Huub van de Wetering, and Jack van Wijk. *Botanical Visualization of Huge Hierarchies*. In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2001)*, pages 87–94. IEEE Computer Society Press, San Diego, California (2001). `http://www.win.tue.nl/~vanwijk/botatree.pdf`. (Cited on page 29.)

Matthias Kreuseler and Heidrun Schumann. *Information Visualization using a New Focus+Context Technique in Combination with Dynamic Clustering of Information Space*. In: *Proceedings of the Workshop on new Paradigms in Information Visualization and Manipulation (NPIVM '99)*, pages 1–5. ACM Press, Kansas City, Missouri, USA (1999). DOI:10.1145/331770.331772. `http://www.informatik.uni-rostock.de/~mkreusel/SInVis/infovis.html`. (Cited on pages 27 and 63.)

Krista Lagus, Samuel Kaski, and Teuvo Kohonen. *Mining Massive Document Collections by the WEBSOM Method*. *Information Sciences*, 163(1–3), pages 135–156 (2004). DOI:10.1016/j.ins.2003.03.017. `http://websom.hut.fi/websom/doc/ps/Lagus04Infosci.pdf`. (Cited on page 10.)

John Lamping, Ramana Rao, and Peter Pirolli. *A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*, pages 401–408. ACM Press, Denver, Colorado USA (1995). DOI:10.1145/223904.223956. `http://www.ramanarao.com/papers/startree-chi95.pdf`. (Cited on pages 21, 51 and 54.)

Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. *The Perspective Wall: Detail and Context Smoothly Integrated*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pages 173–176. ACM Press, New Orleans, Louisiana, USA (1991). DOI:10.1145/108844.108870. (Cited on page 5.)

Guy Melancon and Ivan Herman. *Circular Drawings of Rooted Trees* (1998). `http://ftp.cwi.nl/CWIreports/INS/INS-R9817.pdf`. (Cited on page 18.)

Bojan Mohar. *Drawing Graphs in the Hyperbolic Plane*. In: *Proceedings of the 7th International Symposium on Graph Drawing (GD '99)*, pages 127–136. Springer, Stirin Castle, Czech Republic (1999). `http://portal.acm.org/citation.cfm?id=647551.729255`. (Cited on pages 51 and 54.)

Tamara Munzner. *H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space*. In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '97)*, pages 2–10. IEEE Computer Society Press, Phoenix, Arizona, USA (1997). DOI:10.1109/INFVIS.1997.636718. `http://graphics.stanford.edu/papers/h3/`. (Cited on page 22.)

Tamara Munzner, Francois Guimbretiere, Serdar Tasiran, Li Zhang, and Yunhong Zhou. *TreeJuxtaposer: Scalable Tree Comparison using Focus+Context with Guaranteed Visibility*. In: *Transactions on Graphics (TOG 2003)*, 22(3), pages 453–462. ACM Press (2003). DOI:10.1145/882262.882291. `http://www.cs.ubc.ca/~tmm/papers/tj/`. (Cited on page 20.)

prefuse. *prefuse: An Interactive Visualisation Toolkit*. UC Berkeley Group for User Inteface Research (2004). `http://prefuse.sourceforge.net/`. (Cited on page 18.)

Werner Putz. *The Hierarchical Visualization System*. Master's Thesis, Graz University of Technology, Austria (2005). `http://www.iicm.edu/thesis/wputz.pdf`. (Cited on pages 31 and 91.)

Ramana Rao and Stuart K. Card. *The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*, pages 318–322. ACM Press, Boston, Massachusetts, USA (1994). DOI:10.1145/191666.191776. (Cited on page 8.)

Edward M. Reingold and John S. Tilford. *Tidier Drawings of Trees*. *IEEE Transactions on Software Engineering*, 7(2), pages 223–228. (1981). (Cited on pages 16 and 41.)

George G. Robertson and Jock D. Mackinlay. *The Document Lens*. In: *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology (UIST '93)*, pages 101–108. ACM Press, Atlanta, Georgia, USA (1993). DOI:10.1145/168642.168652. (Cited on page 5.)

George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. *Cone Trees: Animated 3D Visualizations of Hierarchical Information*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pages 189–194. ACM Press, New Orleans, Louisiana, USA (1991). DOI:10.1145/108844.108883. (Cited on page 23.)

Gerard Salton, A. Wong, and C. S. Yang. *A Vector Space Model for Automatic Indexing*. *ACM Communications*, 18(11), pages 613–620 (1975). DOI:10.1145/361219.361220. (Cited on page 10.)

Jürgen Schipflinger. *The Design and Implementation of the Harmony Session Manager*. Master's Thesis, Graz University of Technology, Austria (1998). `http://www.iicm.edu/thesis/jschipf.pdf`. (Cited on page 6.)

Jinwook Seo and Ben Shneiderman. *Interactively Exploring Hierarchical Clustering Results*. *IEEE Computer*, 35(7), pages 80–86 (2002). `http://www.cs.umd.edu/local-cgi-bin/hcil/rr.pl?number=2002-10`. (Cited on page 20.)

Nihar Sheth and Qin Cai. *Visualizing MeSH Dataset using Radial Tree Layout*. *Spring 2003 Information Visualization Class Project* (2003). http://iv.slis.indiana.edu/sw/papers/radialtree.pdf. (Cited on page 16.)

Ben Shneiderman. *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*. In: *Proceedings of the IEEE Symposium on Visual Languages (VL '96)*, pages 336–343. IEEE Computer Society Press, Boulder, Colorado, USA (1996). DOI:10.1109/VL.1996.545307. (Cited on pages 3 and 4.)

John Stasko and Eugene Zhang. *Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations*. In: *Proceedings of the IEEE Symposium on Information Vizualization (InfoVis 2000)*, pages 57–65. IEEE Computer Society Press, Salt Lake City, Utah, USA (2000). DOI:10.1109/INFVIS.2000.885091. http://www.cc.gatech.edu/~john.stasko/papers/infovis00.pdf. (Cited on page 26.)

Bernhard Tatzmann. *Dynamic Exploration of Large Graphs*. Master's Thesis, Graz University of Technology, Austria (2004). http://www.iicm.edu/thesis/btatzmann.pdf. (Cited on page 5.)

Joel D. Tesler and Steven L. Strasnick. *FSN: The 3D File System Navigator,*. Silicon Graphics, Inc. (1992). ftp://ftp.sgi.com/sgi/fsn. (Cited on page 27.)

William P. Thruston. *Three-Dimensional Geometry and Topology*. Princeton Mathematical Series 35. Princeton University Press (1997). ISBN 0691083045. (Cited on page 51.)

Lisa Tweedie, Bob Spence, David Williams, and Ravinder Bhogal. *The Attribute Explorer*. In: *Conference Companion on Human Factors in Computing Systems (CHI '94)*, pages 435–436. ACM Press, Boston, Massachusetts, USA (1994). DOI:10.1145/259963.260433. (Cited on page 8.)

John Q. Walker II. *A Node-Positioning Algorithm for General Trees*. In: *Software—Practice and Experience*, 20(7), pages 685–705. John Wiley and Sons, Inc. (1990). http://portal.acm.org/citation.cfm?id=79026. (Cited on pages 16, 41 and 42.)

Jörg A. Walter and Helge Ritter. *On Interactive Visualization of High-dimensional Data using the Hyperbolic Plane*. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mmining (KDD 2002)*, pages 123–132. ACM Press, Edmonton, Alberta, Canada (2002). DOI:10.1145/775047.775065. http://www.techfak.uni-bielefeld.de/~walter/pub/Walter02-kdd.pdf. (Cited on page 51.)

Michael Welz. *The Java Pyramids Explorer: A Portable, Graphical Hierarchy Browser*. Master's Thesis, Graz University of Technology, Austria (1999). ftp://ftp.iicm.edu/pub/thesis/mwelz.pdf. (Cited on page 29.)

Charles Wetherell and Alfred Shannon. *Tidy Drawings of Trees*. *IEEE Transactions on Software Engineering*, 5(5), pages 514–520 (1979). (Cited on page 16.)

Josef Wolte. *Information Pyramids: Compactly Visualising Large Hierarchies*. Master's Thesis, Graz University of Technology, Austria (1998). ftp://ftp.iicm.edu/pub/thesis/jwolte.pdf. (Cited on page 29.)