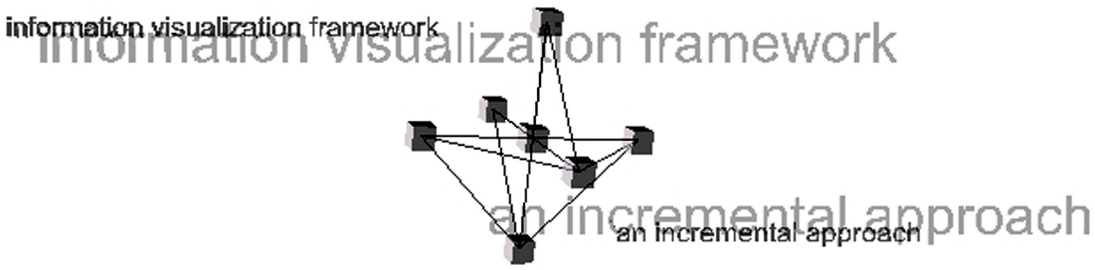
Visualization framework for information graphs

an incremental approach



Visualization Framework for Information Graphs an Incremental Approach

Diploma Work by Patrick Bichler

February 2002

Department of Computer Science, Database Research Group
Swiss Federal Institute of Technology (ETH) Zurich

Supervision: Kai Jauslin, Dr. Roger Weber
Professor: Prof. Dr. Hans-Jörg Schek

To my daugther Lenya

I would like to thank the many people who have helped me on the path towards this diploma work during and before my time at ETH. Especially, the acknowledgment to Kai Jauslin and Roger Weber for providing data, advice and support.

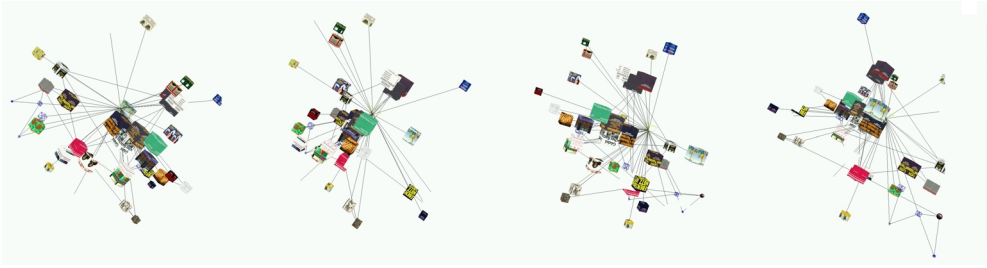
Abstract

The research in information visualization brings a wide variety of different visualization techniques. But there is no integrated solution which makes it possible to use these visualizations on different data sources. We developed a general framework for the visualization of huge graphs by an incremental exploration. In each step we generate a logical frame containing a well defined region of the whole information graph.

The interface between the data providers and the visualization algorithm is realized by the construction of an information graph. An information graph is a general model to describe relationships in information spaces and is generated by neighborhood queries on the data sources.

Our application 'InfoGraph' is an implementation using this framework with several visualizations and data sources.

Keywords: information visualization, graph drawing, incremental approach, framework, clustered graphs, navigation/interaction, three-dimensional, focus+context, clustering, Java3D



Contents

Chapter 1

Introduction

1.1 Motivation

Information visualization has a wide range of applications in the design of user interfaces, information or content management, data warehousing and scientific data visualization, and numerous other fields.

> *"The growing amount of data and information leads to the creation of virtual environments. We can only make sense of it all if we provide suitable interfaces and aids." [3]*

The research in information visualization brings a wide variety of different visualization techniques in each data domain to aid comprehension of these complex connections. But there is no integrated solution which makes it possible to use these visualizations on different data sources. Therefore we aim to develop a general framework for many data sources and visualizations.

We need a standardized scheme to access data sources for integrating several types of them. We try to represent the information structure and the inter-relationship of items in an information graph and discuss the advantages of this attempt. Thus the information is first expressed as a graph and then a layout algorithm is used to create a visual representation of this graph.

We discuss the needs and problems of such a framework and implement it as a proof of concept. Our goal is not to invent yet another visualization technique, but to build an integrated framework.

1.2 Incremental Approach

To reach the goal of a generalized framework we will model the structure of a data source by an information graph and take an incremental approach to visualize parts of the graph. So in each step we just look at a slight portion of the available information and therefore we can reduce the complexity in layouting. Figure 1.1 shows three possible extracts from an information graph. We attempt to layout these frames with different visualizations and provide functions to explore the whole graph.

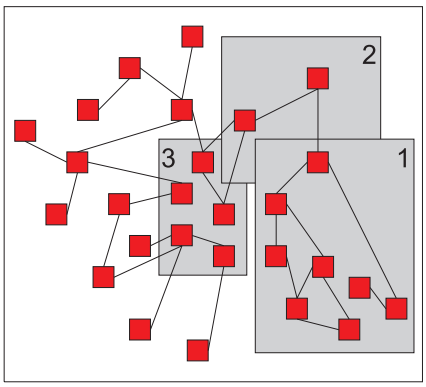


Figure 1.1: Incremental Approach

The implementation of such a framework has to meet the criteria of being fast, nice looking, user friendly and flexible. It would not be possible to meet all of these demands, but it is practicable to build a general framework for large and semi-structured data sources with the ability to calculate nice looking screen representations in short time.

Chapter 2

Terminology

Before beginning to describe the related work in the field of information visualization, we have to define several terms for the better understanding of this work and to provide a uniform source language.

2.1 Information

The first definitions are about data, information and knowledge, because we have a lot of data sources storing the fundamental components of our information society. These data pieces are combined and stored with additional information about their relationships. This leads to the definition of information. In conjunction with the human being we can define knowledge. The definitions are taken from [28] and translated.

Data Facts and illustrations of reality, called data, are the fundamental components of information science. These character chains are unstructured and independent of any context.

Information Information is the arrangement of data into meaningful structures. They mark objective content and are subjectively perceptible and usable.

Knowledge Knowledge is based on data and information, but is, in contrast to them, independent of a person. It comprises theoretical discoveries well as practical ways of acting. Knowledge stands for the totality of discoveries and abilities which is put into play by individuals to solve problems. It is developed from the processing of information by the integration of information into the context of experience. Therefore, knowledge is extremely subjective.

Finally, the data access to a collection of data pieces is defined:

Data Source A data source is in our context an accumulation of electronicly available facts, typically from a certain domain and with uniform data access methods. It is the provider of data objects with a well known interface for querying.

2.2 Graph Theory

2.2.1 Graphs

Graph A graph is a pair (V, E), where V is the set of vertices and E the set of edges. An edge is a couple of vertices u, v with u and v contained in V.

Not all graphs are simple. Sometimes a pair of vertices is connected by multiple edges yielding a multigraph. At times, vertices are even connected to themselves by edges called loop, yielding a pseudograph. Finally, edges can also be given a direction yielding a directed graph (or digraph).

In this work we will consider all the graphs ro be simple, meaning there exist neither multiple edges nor self-loops. We define n as the size of the graph (=number of vertices), and the degree of a vertex is the number of edges that are incident with it.

2.2.2 Weighted Graphs

Often it is desirable to attach information to the vertices or edges. These graphs are called weighted graphs. The pieces of information attached to the vertices and edges are called weights, capacities, attributes, or labels depending on the context within which they are used.

2.2.3 Connected Graphs

A graph is connected if there is a path connecting every pair of vertices. Otherwise it can be divided into connected components (disjoint connected subgraphs).

2.2.4 Tree, Spanning Tree

Tree A tree is a connected, acyclic graph.

spanning tree

The *spanning tree* of a graph G is a subgraph T of G which is a tree and satisfies |E(T)|=|V(G)| - 1 and |V(G)| = |V(T)|.

2.2.5 Clustering

Clustering Clustering is the automatic grouping of 'close' vertices, where 'close' can be defined by geometric distance or graph-theoretic distances. So in general, a cluster is defined as a set of similar objects.

2.2.6 Clustered Graphs

Clustered Graph A clustered graph C = (G, T) consists of an undirected graph G = (V, E) and a rooted tree T such that the leaves of T are exactly the vertices of G. Each node v of T represents a cluster of vertices of G consisting of the leaves of the subtree rooted at v. The tree T describes an inclusion relation between clusters; it is the cluster tree of C. [10]

The clustering mechanism can be used to model hierarchies, classifications and categorization of the data objects in the data source. There are two kinds of

Semantic clustering
Structural clustering

clustering approaches. *Semantic clustering* uses the knowledge of the data domain to define the similarity of data objects, whereas *Structural clustering* only needs the structure of the information graph for building the clusters.

In several applications, the user has to define clusters even if he doesn't know about anything about clustering or clusters. For example we store files onto the hard disk in a tree structure of folders. Thus the files will be divided into several groups. That is exactly what a clustering algorithm would do. Therefore we call this a *natural clustering*.

natural clustering

Natural clustering is used to simplify the graph or to store additional information about the structure in the graph. Alternatively, clustering algorithms can be used to generate a dynamic partitioning of the entities.

For large and complicated relational structures, it is useful to introduce a hierarchy of abstractions above the elementary relationships between nodes. One model for such abstractions is called CIgraph. [30] More information about CIgraphs can be found in section 3.1.

2.3 Information Space

Information Space An information space is a collection of data objects generated and edited by several application programs with unique access patterns. The relations between data objects are calculated by similarity measurements or user defined associations.

2.4 Information Graph

We combine the concept of a clustered graph with an information space to get the definition of an information graph.

Information Graph An information graph is a clustered graph, where the vertices are the entities in the information space, edges represent similarities among them and the hierarchy of abstractions is a natural or dynamic clustering of vertices. The nodes are weighted with respect to their importance in the information space and the edges as similarity values normalized to]0,1]. ('0' is defined as high similarity and strong importance.)

There is one special vertex. The vertex which represents the starting point is called *center object* in the information graph. This object is initially placed at the center of the virtual environment.

center object

2.5 Visualization

No particular geometry is associated with an information graph. A visualization algorithm is the projection of the information space or information graph to a virtual environment. This means the placement of the objects and therefore the calculation of coordinates in this virtual room. The goal is to find an arrangement such that the picture is easy to understand and fits into the viewing area of the display device. [38] There is a wide range of different visualizations mainly for specific applications or data sources. In chapter 4.2 we present different types of them.

2.5.1 Graph Layout

The graph layout is a specialized visualization algorithm which can be applied to graphs. This methods are subject of the research in *graph drawing*. "The problems to achieve a good layout are to minimize edge crossings and the display of symmetries existing in the graph. Furthermore, edges should have as few bends as possible, and the deviation on their lengths should be small. The area used for drawing should be as small as possible, while the vertices and edges should be evenly distributed in the area. Connected vertices should be close to each other." [15] In the following chapter we describe several approaches for graph layouts.

graph drawing

2.5.2 Information Visualization

We define the term information visualization to be an extension of graph layouting. The short term for it is 'visualization'.

Information Visualization Information Visualization is defined as the visualization and navigation of abstract data structures. [22]

In our context, the abstract data structure will be an information graph.

Level of Detail A level of detail (LOD) function assigns to each entity in the information space the amount of additional information which should be drawn in the visualization with respect to its position and the view point.

The calculation of this function can take the distance from the viewpoint, the position in the drawing or other parameters as input. It can influence the visibility, scaling or the whole representation of an information object.

2.5.3 Signs

Sign A sign is the representation of an entity in the information space, drawn at the position calculated by the visualization.

Applying this definition to an information graph, it is the representation of a vertex drawn at the position calculated. These signs can be images, thumbnails, geometric figures or strings and can be influenced by a LOD function.

2.5.4 Brushing

To each object in the information space there has to be assigned a unique id for the system to distinguish it from others and a marking which is called brushing in the context of information visualization. If we look at the filesystem as data source, the location (host + path + name) can be used as id and the name for the brushing.

Brushing Brushing is the annotation of entries in the information space with a typical expression or keywords.

In the spatial arrangement the brushing is shown near the data object with respect to the level of detail.

2.6 Framework

What is an object-oriented framework? A short answer is: A framework is a set of related classes which can be specialized and/or instantiated to implement an application or subsystem. It is a kind of class library but also defines a generic design.

Framework A framework is a reusable design expressed as a set of abstract classes and the way their instances collaborate. It is a reusable design for all or part of a software system. By definition, a framework is an object-oriented design. It does not have to be implemented in an object-oriented language, though it usually is. The framework provides a context for the components in the library to be reused. [25]

Chapter 3

Related Work

We have a look at other publications concerning subjects related to this work. First we discuss clustered graphs as structure for an information graph, visualizations in two or three dimensions for general graphs or trees and how to improve the algorithms for large graphs. Further we look at the incremental approach and several focus+context concepts. Finally there are certain publications about navigation and interaction in virtual environments.

3.1 Clustered Graphs

There have been several attempts to model clustered graphs. A comparison of them is made by Lai and Eades. [46]

Higraph Harel presents Higraphs for the representation of complex relations. They involve multi-level blobs that can enclose or intersect each other. The blobs are similar to clusters in our context but the clustering is not forced to be a tree structure. "Higraphs are suited for a wide array of applications to databases, knowledge representation, and, most notably, the behavioral specification of complex concurrent systems using the Higraph-based language of statecharts" [21] (see Figure: 3.1)

Compound Digraph Sugiyama and Misue present compound digraphs. The compound digraph is an extension of the directed graph with a set V of primitive nodes, a set F of frames with a nesting relation $I \subseteq (V \cup F) \times (V \cup F)$ and a set of primitive edges $E \subseteq (V \cup F) \times (V \cup F)$. Since no frame can be nested into a primitive node or into itself, the nesting relation can be seen as a tree $T = (V \cup F, I)$ with $f \in F$ as inner nodes and $v \in V$ as leaves. [43] (see Figure: 3.2)

CIgraph (Compound graphs with Integrated layout functions) The CIgraph definition corresponds to our definition of a clustered graph. Additionally it is allowed to add layout functions to each cluster for the presentation in a virtual environment. "The aim of a CIgraph is to represent combinatorial relationships and attach layout functions to each node in the hierarchy; in this way, any algorithm aimed at classical graphs is available for compound graphs and different parts of the hierarchy can use different algorithms." [30] (see Figure: 3.3)

We took the main ideas from these concepts to define an information graph (see 2.4). In the context of information visualization we want to define the blobs or frames as results of a clustering algorithm or a categorization specified by the user. Therefore we named them data cubes. The assigned layouting functions in the CIgraph would be too confusing for the user, thus we take one visualization algorithm for each logical frame we generate during the exploration of the graph. And drawing of blobs, which can intersect in arbitrary ways, will be very difficult to realize. [46]

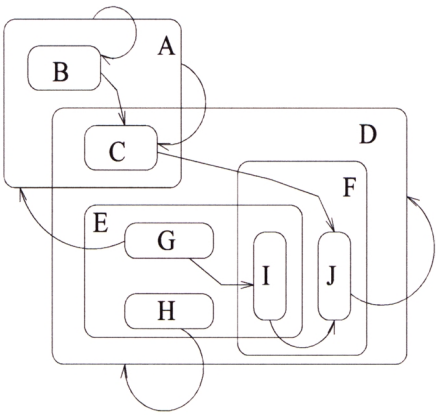


Figure 3.1: Higraph [46]

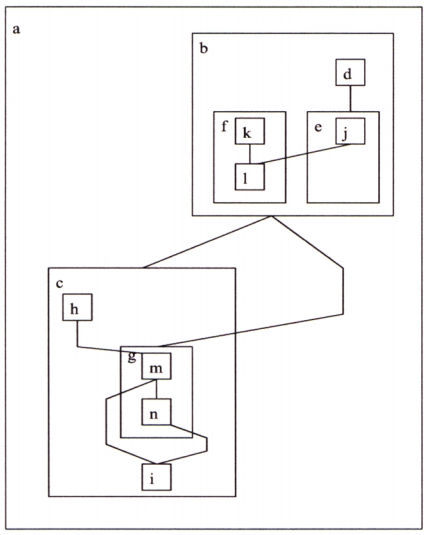


Figure 3.2: Compound Digraph [46]

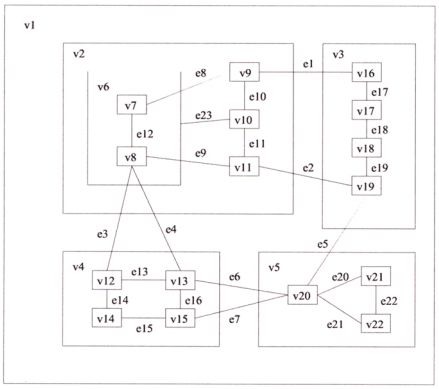


Figure 3.3: CIgraph [46]

3.2 Visualizations

3.2.1 Graph Drawing

Graph drawing is very popular, because it can be applied to a wide range of engineering activities. For example in network design, physics, chemistry, traffic planning, biology, production planning. Often these pictures are drawn manually and the resulting image shows the underlying relationships among the considered objects. [17] [13] The basic graph drawing problem can be described as: "Given a set of vertices with a set of edges (relations), calculate the position of the nodes and the curve to be drawn for each edge." [22]

3.2.1.1 Drawing Clustered Graphs

Higres stands for a "visualization system for clustered graphs and graph algorithms" [31]. It is a visualization tool and editor for attributed clustered graphs combined with the execution and animation of graph algorithms. The algorithms executed by Higres are implemented in external modules, such that the user can create modules supporting the wished semantics.

Another method to draw clustered graphs was introduced by Eades and Feng in the paper "Drawing Clustered Graphs on an Orthogonal Grid" [12]. They present an algorithm which produces planar drawings of clustered graphs with the constraint of orthogonal grid rectangular cluster drawings. This constraint appears in applications like VLSI circuit design and diagrammatic interfaces for relational information systems, where the graph is mapped onto a grid with edges drawn as sequence of horizontal and vertical segments, vertices on the grid plots and region boundaries for clusters as rectangles.

3.2.1.2 Force-Directed Algorithms

In this approach to visualizing graphs, we take a physical system of springs and optimize an energy function over this model. The first algorithms published by Quinn and Breur [40] named "force-directed placement algorithm" and the equivalent spring embedder model of Eades [11] start with a random embedding of the graph in the virtual environment. These concepts can easily be applied to each undirected graph.

Some examples of algorithms in this field of research are the algorithm of Fruchterman and Reingold [16], the algorithm by Kamada and Kawai [26], the simulated annealing approach of Davidson and Harel [9], the Tunkelang incremental algorithm [45] and the graph embedder GEM by Frick et al.[15] These five methods were compared by Brandenburg, Himsolt and Rohrer [2] with the result that each of these algorithms has its advantages and disadvantages, but all of them are usable.

3.2.1.3 Force-Directed Algorithms with Constraints

The spring algorithm can be implemented with mechanical constraints such that the vertices have to lie on simple 3D surfaces. The inclusion of such mechanical constraints on the system can improve the readability of the layout and is described in [38].

3.2.1.4 Self-Organizing Graphs

"Self-organizing graphs are a novel approach to graph layout based on a competitive learning algorithm. This method is an extension of self-organization strategies known from unsupervised neural networks, namely from Kohonen's self-organizing map." [34] The arrangement function is equivalent to force-directed algorithms. One of the main differences is the approximation for the displacement vectors in each round of computation. In the self-organizing graph algorithm the optimization problem is the byproduct of a stochastic self-organization process, whereas in force-directed methods an energy minimization is calculated.

3.2.2 Three Dimensional Visualizations

The drawing in three dimensions has been studied by several persons. We just looked at the most important visualizations. In [29] Kumar and Fowler discuss the extension of the spring-force model from two to three dimensions. "The simplest approach is to generalize classical 2D layout algorithms for 3D. [...] Most force-directed methods are also described in dimension independent terms, which allows them to be generalized to 3D. [...] In spite of their apparent simplicity, displaying graphs can also introduce new problems. Objects can occlude one another and it is also difficult to choose the best 'view' in space." [22] To weaken this problematic nature, transparency can be used to avoid occluded objects. And changing the point of view can minimize the edge-crossings.

The application designer gains more space by using the third dimension and also real world metaphors can be considered for visualizations: Perspective Wall [32], Web Book [5]. But in general, the three-dimensional algorithms are quite different from those in two dimensions, because the representation is viewed through a single point perspective projection. [6] This affects the integration of signs within the 3D-drawing. The application developer has to carefully think about showing images and texts on a plane or as texture.

In the literature we can find the term 2.5D visualization, where the visualization arranges the items in two dimensions and the third direction is used for meta information (e.g. brushing). Examples: landscape visualizations - since they are built by interpolation of two-dimensional geometric constellations, 2.5D Model of Cugini, Laskowski and Sebrechts [8]

3.2.3 Tree Drawing

If the information graph to be visualized is a tree, the layouting algorithm can be more complicated because of the reduced graph complexity. Creating aesthetically pleasing drawings is possible in polynomial time. [44]

3.2.3.1 Planar Tree Drawing

Planar Tree Drawing methods take the tree structure and calculate coordinates in two directions. The paper by Herman, Melancon and Marshall [22] shows four variants of them. These algorithms point out the hierarchical structure in the graph.

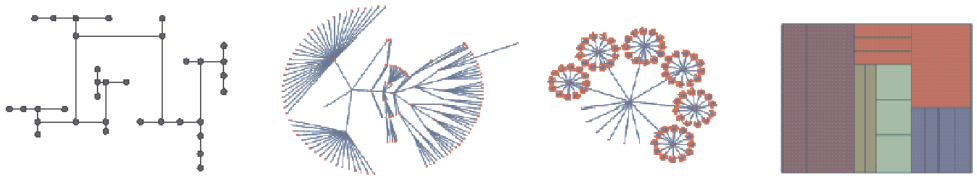


Figure 3.4: Planar Tree Drawing Algorithms
from left to right: H-Tree Layout, Radial View, Balloon View, TreeMap [22]

3.2.3.2 ConeTree

ConeTrees were first described in [20] as a three-dimensional extension to the more familiar 2D hierarchical tree structures. Cone Trees show the root node at the top of the screen and place the child nodes at equal distances along the base of a cone and will be repeated recursively.

Carrière and Kazman [7] extended this algorithm with a bottom-up estimation process which calculates an optimal radius for each cone based on the space which is used by the sub-cones. The resulting ConeTree layout is shown in figure 3.5. The same idea of a bottom-up estimation process is used by Munzner for the Hyperbolic Tree.

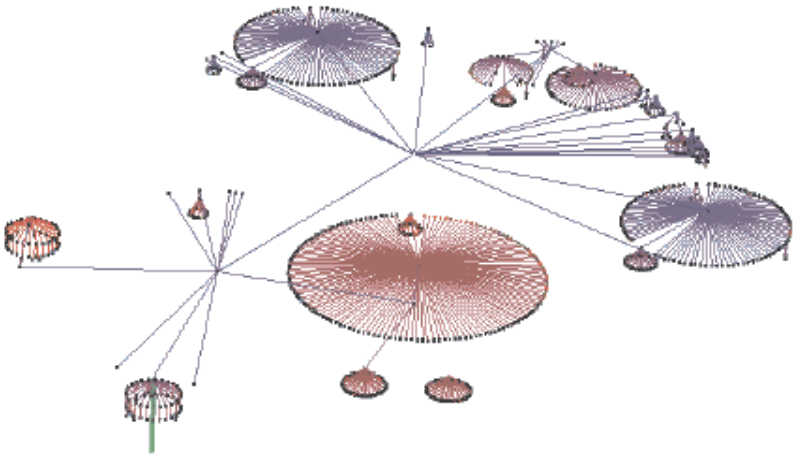


Figure 3.5: Extended ConeTree Visualization [7]

3.2.3.3 Hyperbolic Tree

Tamara Munzer [36] worked out many ideas about the interactive visualization of large graphs and networks. Very well known is the 3D-visualization of Hyperbolic Quasi-Hierarchical Graphs and the implementation called H3 Viewer (see figure 3.6). An improvement is the handling of very large graphs with rendering at a guaranteed frame rate independently of the graph size. The difference between the H3 algorithm by Munzner and the previous work in distortion-based graph drawing is the consideration of navigation and layout in the 3D hyperbolic space. This idea was taken by the Cooperative Association for Internet Data Analysis (CAIDA) into the Walrus - Graph Visualization Tool [4]. It uses the hyperbolic geometry for visualizing large directed graphs, where large is in the order of a million nodes, and about as many links in the domain of network topology graphs.

3.2.4 FastMap - a Clustering Algorithm

"A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets" [14] - The input to this clustering algorithm is a distance matrix between all data objects in the data source. These distances were calculated by k-dimensional feature extraction functions and are mapped to a point in the k-dimensional space. By projections the dimensionality is reduced untill the coordinates are two- or three-dimensional vectors. For the information visualization we can take these coordinates, because the resulting distances among the objects show the relationship between them.

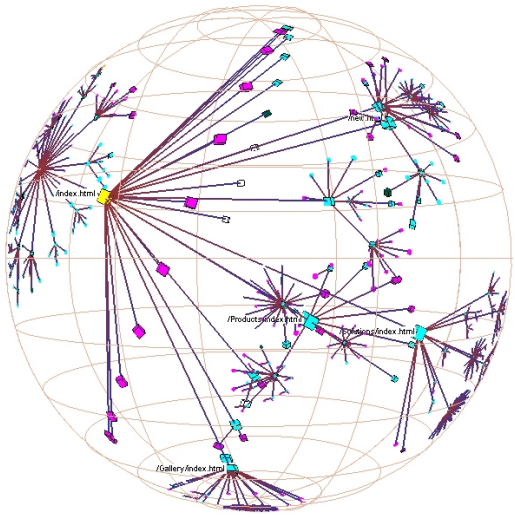


Figure 3.6: Hyperbolic Tree [36]

3.2.5 Drawing Large Graphs

Imagine a graph with thousands of nodes an edges. None of the force-directed algorithms can handle this amount of data, because of there complexity in calculating the attractive and repulsive forces. Even the drawing becomes impractical because of the screen resolution, or the user cannot recognize the relationships in the mess. There are several approaches to avoid these problems: Fish-Eye views show an area of interest enlarged and detailed while showing the surrounding regions smaller. [42] The Fish-Eye concept is also part of the Focus + Context method we describe in the next section. Multi-level views allow the user to see large graphs at multiple abstraction levels. Finally we can take an incremental approach and showing only a slight portion of the whole graph in each step.

The paper 'Fast Multi-Dimensional Algorithm for Drawing Large Graphs' [18] describes all these problems and suggests a vertex filtration to reduce the complexity and combines it with an intelligent initial placement of the vertices. The algorithm can even be extended to calculate positions in a k-dimensional environment. Gajer, Goodrich and Kobourov [18] stated that the drawings are nicer in 3D if the calculation is made for four dimensions and the result is projected into the 3D-coordinates.

3.2.6 Incremental Layouts

An alternative to fit an entire graph into one view is to provide interactive exploration of subregions of the graph. "The advantage of such an incremental approach is that, at any given time, the subgraph to be shown on the screen may be limited in size, hence the layout and interaction times may not be critical any more." [22] [23] [35]

Here, subgraphs can be layouted in short time, interaction will be faster and not the whole graph has to be fetched. In several cases, it is not even possible to get all data objects from the data source (e.g. www, dynamic content).

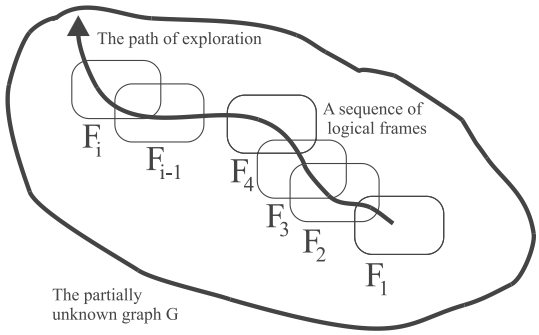


Figure 3.7: Exploration of a Huge Graph [22]

But there are some problems to be solved. We have to answer the following questions, because the user only sees a small extract from the whole information graph: How to create new logical frames? What is the influence in the navigation? How to animate the transition between two frames? These questions have to be answered by a developer for each application during the conception of the system.

3.3 Focus+Context

The term "focus+context" has been used to describe a technique where the user can focus on some detail without losing the context.

3.3.1 Fish-eye Distortion

"A fish-eye camera lens is a very wide angle lens that magnifies nearby objects while shrinking distant objects. It is a valuable tool for seeing both 'local detail' and 'global context' simultaneously." [42] This idea can be adapted to any visualization algorithm. The distortion creates a mapping from the original coordinates to distorted ones, which will be presented to the user. This enlarges the area of interest and shows the environment with less information, which is the basic idea of all distortion models. These concepts are summarized in [22] [39]. Several different mechanisms can be used to implement the fish-eye lens effect. It is not in general necessary to construct an explicit lens model, rather the effect can be achieved through simple geometric transformations. Such a mechanism can be found in [42].

3.3.2 3D Distortion

In a perspective framework the two-dimensional surface can be placed on a plane. As in the 2.5D visualizations, the third dimension can be used for the focus+context view. The surface will be manipulated such that the focal region is raised.[6]

In general three-dimensional layouts the methods can be adapted to a spatial distortion in x/y/z coordinates by applying the distortion function to all dimensions. On the other side, "the 3D visualization provides focus and context through the operation of linear perspective. Objects seen in the foreground have more detail than those further away. By changing the viewpoint, the foreground, and hence the focus, can be changed." [39]

3.3.3 Elision

"Elision is a technique where parts of the structure are hidden until they are needed. Typically, this is achieved through collapsing a node that contains a sub-graph into a single node'." [39] This method is very useful in the case of clustered graphs, because entire sub-graphs can be collapsed.

3.3.4 Multiple Windows

Several visualization systems use more than one window - one to give an overview and the other for the details. [39] [1] A system implementation has to consider about the content synchronization for each user interaction, such that the user never loses the orientation. This corresponds to the level of detail concept by showing providing two levels in separated windows.

3.3.5 Hyperbolic Layout

The hyperbolic geometry is a natural way to produce a focus+context representation. This approach allows to map the infinite Euclidean space into a finite disk so that the objects near the center is enlarged and near the periphery they are shrinked. "Second, we can allocate the same amount of room for each of the nodes in a tree while still avoiding collisions because there is an exponential amount of room available in hyperbolic space." [36] Thus the hyperbolic space provides infinite space for the drawing of large graphs and a projection in a finite part of the Euclidean space.

3.4 Interaction/Navigation

Interactive navigation consists of changing either the viewpoint, the positions of objects in a scene or the fetching of new data. We present the available concepts for these issues.

3.4.1 Zoom and Pan

Zooming is easy to implement with graph visualization in mind by simply adjusting the size of the visible objects in the screen representation. We can find two types of zooming. [22] Geometric zooming provides a blow up of the graph content; semantic zooming means that the information content changes and more details are shown when getting closer to a particular region of the graph. The problem is not the zooming operation itself, but rather to assign an appropriate level of detail.

Panning is a translation in the virtual environment to bring a certain area of interest in front of the eye position and only a part of the data objects is visible through the viewing window. A well-known problem of zooming is that if one zooms on a focus, all contextual information is lost. This can be resolved by using focus+context techniques as described earlier in this chapter.

3.4.2 Exploration

Considering the navigation of an incremental visualization method, the exploration of the graph and therefore the updating of the current window is also a kind of user interaction. In virtually any system, this kind of navigation is supported by menu buttons and checkboxes to make choices, define queries or set further properties of the visualization till the user gets what he looks for.

3.4.3 Rotation

In three dimensions the user can additionally change the point of view by rotation. This operation is quite useful to get an overview of the main structure of the visual encoding.

3.5 Animation

Animations may reduce the cognitive effort of the user in recognizing changes in the information graph or the structure of the layout by keeping the mental map. [10] There are several types of animation.

3.5.1 Animated Addition and Deletion

When a vertex is deleted or added to the abridgment, animated shrinking or growing can be used to help the user identify nodes that are appearing or disappearing.

3.5.2 Animated Translation and Scaling

Depending on the focal point, the distortion function recalculates the size and position of the entities. The transition can be animated in an information visualization system. "Viewers have a much easier time retaining their mental model of an object if changes to its structure or its position are shown as smooth transitions instead of discrete jumps." [19]

The interactive exploration of a graph with the radial layout described by Yee, Fisher, Dhamija and Hearst [47] shows a concept for the animated transition to a new layout changing the center of interest.

3.5.3 Animated Cluster Opening/Closing

This is the extension of the elision techniques with an animation to provide a smooth transition from one state to another.

3.5.4 Camera animation

Camera animation moves the whole drawing. It can be used, for example, to move specific nodes of interest to the center of the screen.

Chapter 4

Concept

In this chapter we describe our concept for an incremental visualization system. The main idea behind this framework is the representation of a small portion of the information graph in each drawing. By changing this logical view, we will fetch another part of the graph. Each section of this chapter is illustrated with the filesystem and an image database as examples. For the presentation of the subjects concerning visualizations we used the spring-force algorithm (3.2.1.2) and the ConeTree (3.2.3.2) concepts.

4.1 Link to ETHWorld Infrastructure

The ETHWorld infrastructure provides several data sources for which we implement a generic framework to visualize all kinds of data available. The goal is to have a spatial representation of the data in a three-dimensional space. The data sources deliver an information graph describing the similarity of entries in the information space or the search results to a query. This system can be used for ergonomical and social studies. The ETHWorld infrastructure will be the main source of data. Additionally other data sources like search engines should be wrapped into our visualization framework.

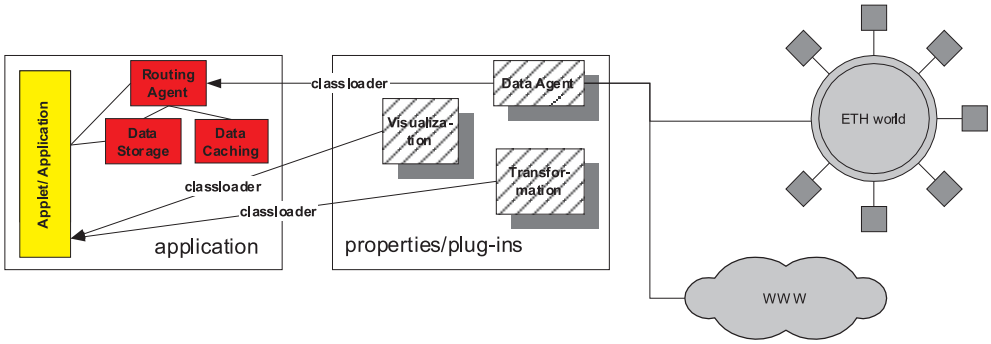


Figure 4.1: Link to ETHWorld Infrastructure

4.2 Data Sources

As defined in the first chapter, the data source can be structured independently from the domain as an information graph. We consider the graph to be undirected. Each piece of information is represented by an object in the data source and has an unique object identification. These data objects can be divided into three types.

data unit

First there is the *data unit* as reflection of entities in the information space (e.g. picture, URL, database entry). All data units have a marking which can be used for

data cube

the brushing and an URL pointing to the subject itself. Secondly we define a *data cube* as a cluster of data units. This grouping is the semantic clustering within the data source. The data cubes build a hierarchical structuring of the data units which can be described with a tree (e.g. yahoo directory, folders of file system). Finally

data query

the *data query* is an imaginary object standing for the initial query to the data source.

To completely describe an information graph, we need to have edges between the data objects representing their relationship (e.g. image similarity, www linkage). To each link a metric value can be assigned standing for the strength of the relationship. For generality this value has to be normalized to the interval]0,1] ('0' corresponds to near and '1' to far away).

Even the query generation, domain specific transformations, the available similarity features and the sign to plot data objects depend on the data domain. We describe the needs in section 4.2.3. The representation of data objects in the visualization is a method to plot themselves. We plan the optional assignment of a scene graph node to each data object. This is the representation displayed as sign.

example

	filesystem	image database
data units	files	images
data cubes	folders	
similarity	hierarchy	image features
signs	if file is image: textured sphere	textured cubes with image

4.2.1 Data Access

In our concept of the incremental information visualization we carefully had to think about the data access. These methods have to fulfill our claim to generality with respect to several visualizations and data sources. It wouldn't make sense to retrieve more data objects than necessary because the querying can be a performance bot-

neighborhood queries

tleneck. For these reasons we decided to plan '*neighborhood queries*' for each data object. The whole range of possible access patterns is shown in the figure 4.2.

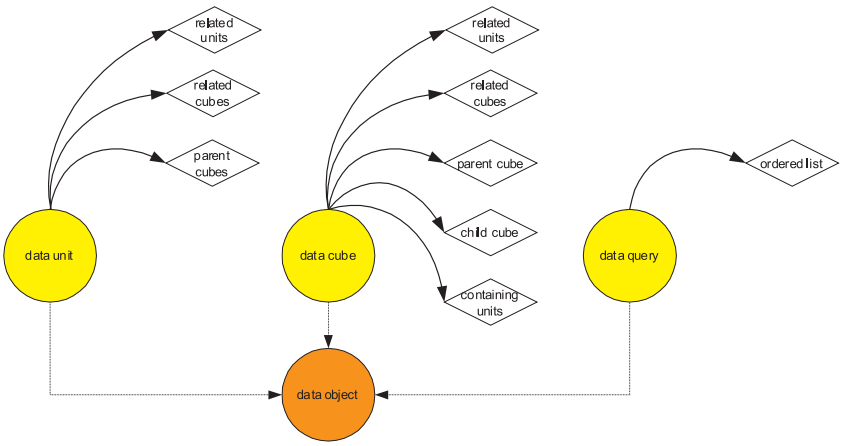


Figure 4.2: Data Access Methods

For each data source, these access methods have to be defined independently. The difficulty is that queries should have individual parameters depending to the data domain. So we'd like to implement a class called *query* and a *query interface*. These classes are extended for each data source with respect to their semantic and the system engineer has the ability to prepare different methods for searching. (see 4.2.3)

query
query interface

In the query interface, there are options encoded for which the user should specify the parameters. Then the system creates a query with these values. To fetch the necessary data objects the data source has to recursively perform several neighborhood queries with different observation points in the graph. The resulting graphs are composed to a complete graph by the routing agent. (see 4.3)

example

	filesystem	image database
similar to data cubes	files, sub-folders, parent folder	
similar to data units	comprised folder	images
query features/options	path	similarity features of image database

4.2.2 Types of Information Graphs

Each data source is modelled by an information graph. But depending on the internal structure, the resulting graph can be more or less connected. For our purpose, we choose among three alternatives: ordered *list*, *tree*, *network*. With the following questions we decide which kind of information graph is available.

list
tree
network

If there are hierarchical structures in the data source, these always have to build the data cubes. Doing so gives us the possibility to assign the interactions 'fold' and 'unfold' (see 4.6) to each data cube. Afterwards we look at the available neighborhood to define the structure of the resulting information graph. The type of an information graph can be evaluated by the answers to the following questions.

Network Is there one of the following relations in the data source: related cubes, related units from cubes or units, parent cube of a unit? If yes we must take the type 'network'.

Tree If only the parent and child relation of a cube is provided and units are only contained in one cube, then the type has to be chosen as 'tree'.

List If none of this methods is available, we have an ordered 'list'.

example

		filesystem	image database
data unit	related units		√
	related cubes		
	parent cubes	√	
data cube	related units		
	related cubes		
	parent cube	√ (only one)	
	child cube	√	
	contained units	√	
resulting	type of graph	tree	network

4.2.3 Data Agents

The data agents transform the knowledge of applications to information graphs. Mainly the data source doesn't provide the data in the demanded way. So we must have a converter. The data agents take queries for the data source, map the data access method to internal functions and send back the results well formed in an information graph. A data agent has to answer a simple question: *What are the data objects in the direct neighborhood of a vertex in the information graph?* This information is encoded in a small information graph containing a vertex for the requested point and all neighbors connected with an outgoing association. This association shows with its assigned measurement the similarity to the querying point. For generality this value also has to be normalized to the interval]0,1].

Figure 4.3 shows a possible information graph to the query for 'c:/documents'. It contains the subfolders 'project' and 'readings' with a similarity value of 0.5 and the file 'content.xml' also with the same measurement. We define that the parent folder is also in the information graph, but give a similarity value of 1.0, standing for far away.

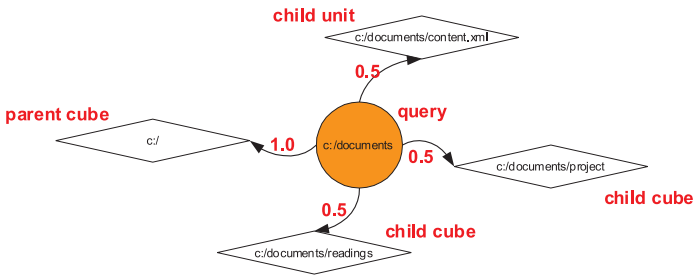


Figure 4.3: Example: Data Access in the Filesystem

Query In the query class we specify the routing type as 'count' or 'radius'. In the counting mode, the routing agent should fetch data till the number of data objects is reached and if 'radius' is defined, data objects are collected till the distance from the starting point in the information graph reaches the value of the parameter 'Radius'. The query has to be extended for each data source with properties and functions so that the data agent can perform it.

Query Interface and Similarity Features The query interface is a panel with the querying options corresponding to the data source. It also contains the input controls to specify the parameters in the extended query.

Observation Point A query is sent several times to the data agent to recursively fetch the data objects in the information graph. In each iteration, the observation point is set. This is the vertex from which to perform the neighborhood query.

Signs To each data object, the data agent can add a sign which is a Java3D scene graph node. This sign is drawn by the visualization. If none is specified, the visualization takes a standardized sign.

example

	filesystem	image database
routing type	radius	count
similarity features	hierarchy	eg. texture, histogram, color
observation point	path	object id
signs	folders: wired cube files: wired sphere, textured if an image	textured cube with image

4.3 Routing Agent, Data Storage

The data storage is the container where the query results from the data agent are collected. To fill this container, we need a routing agent which sends the queries to the data agent and collects the resulting data objects. The answer to a query describes a small part of the information graph and has to be integrated into the data storage. So the routing agent incrementally builds the desired information graph.

Incremental visualizations show the environment of the data object currently observed. Because of speed and overview criteria the radius of data objects in the context has to be limited. We define the radius of neighborhood as the weighted sum from the centric object to another object on the shortest path. Looking at search engines as data sources, there isn't a radius to calculate. Instead of this mechanism we can limit the data objects in the response.

The whole graph in the data storage will be taken by the visualization and rendered in the virtual environment. If there are some transformations, these will be applied to the information graph. Because of our incremental approach we must build an event model to tell the transformations or visualizations about changes in the data storage. We distinguish between the following events:

events

Start Routing This event indicates the the routing agent starts with a new query to fetch the graph. All objects are marked in the data storage as not visited.

Add Vertex If the routing agent has found a vertex which is not yet in the data storage, this event can be used to incrementally add the data objects in the virtual environment.

Update Vertex The event 'update vertex' is used when the routing agent fetches a data object which already is in the data storage. This includes the situation when changes in associations occur or the sign is changed.

Remove Vertex This event indicates that a data object has been removed from the data storage, the transformation and the visualization have to be informed.

Clear All objects are removed in the data storage.

Set Center Object This usually occurs when a new query is launched. Then the visualization typically has to recalculate the spatial layout.

End Routing Now the routing agent has reached the radius or counting limitation and the whole graph is available. All data objects which were not visited are deleted now in the data storage.

4.4 Visualizations

A visualization algorithm is the projection of the information space or information graph to a virtual environment. This means the placement of the objects and therefore the calculation of coordinates in this virtual room. We want to support visualizations with one connected component and the center node placed in the root of the spatial environment.

The introduction of a hierarchy of abstractions makes the graph drawing problem far more difficult, but gives further possibilities to assist the comprehension of the structures in the graph. As in the data sources, each visualization has special demands with respect to the data. The visualization has to specify which data structures can be displayed based on its layouting possibilities.

example

	ConeTree	spring-force algorithm
supported type	tree	network

For the framework we have to think about standard functions for all visualizations to simplify an implementation. The most important functions are the creation of objects in the scene graph for the rendering, to place them at the calculated coordinates and the implementation of a 3D-Library.

4.4.1 Signs

If the data agent doesn't assign a sign to the data objects, the visualization has to plot standard representations for data cubes and units.

4.4.2 3D-Library

In all tasks of the implementation we need components from a 3D-Library. It will be much easier if the main functions are available in a library. Thus the programmer doesn't have to care about the details of the 3D-Library. The most important components are lines, signs, lighting and the texts for brushing. Another thing to take serious is the behavior for navigating, rotating and select objects in the concept of 'focus+context' (further information: 4.5, 5.11.2).

4.5 Navigation

For the navigation in the virtual environment, we want to provide the zooming and rotation functionality. This goes very well with three-dimensional visualization and is very intuitive to make use of it. Further the focus+context concept is implemented. So the viewer can navigate the graph by selecting nodes to become the focal node. Afterwards the spatial layout is rearranged with the selected node in the center and the other objects are distorted in the space. The description of distortions can be found in 4.8. Supplementary a visualization has to gain control of the navigationbehavior. For the implementation of special navigation opportunities, the framework contains callback functions: *mouseOver*, *mouseOut* or *clicked*.

mouseOver
mouseOut
clicked

4.6 Interaction

An interaction is an operation assigned to a data object. The list of interactions will be shown by clicking on the sign in the virtual environment and the user can select one of them. The visualization framework provides some generic interactions like 'open', 'fold' and 'unfold'.

open

fold
unfold

The function *open* takes the subject-URL of the data object and passes it to the standard browser. A speciality of clustered graphs is the built-in tree structure. This can be used to *fold* and *unfold* recursively all children of a data cube. The advantage of this elision technique is the hiding of uninteresting regions of the information graph and the simplification with fewer objects.

The other interactions are executed by the data agent itself. An important function is 'new query' to change the visible part of the information graph. Even other possibilities can be realized by an interaction, like feedback-driven similarity features.

4.7 Animation

morphing

For a visualization it can be necessary to implement animations for a smooth transition between two sights. This can be useful during the incremental building of the screen representation to show at each moment the data available till then. We call this animation *morphing*. Each visualization should implement this function. Further animations can be the rotation of all children in the ConeTree, animated scaling and translation or camera animations like fly-through. The framework provides a class animation which can be added to a visualization and is activatable at runtime. So the visualization algorithm can execute the necessary operations.

4.8 Distortions

The distortion is the manipulation of object positions independently of the visualization and is part of the focus+context concept. The underlying distortion function maps the original coordinates to new ones. This function enlarges objects near the focal region and moves the other objects towards the borderland. The distortion value is a factor regulating the amount of change. A distortion function maps the interval [0,1] to [0,1].

4.9 Transformations

The possibilities to combine data sources and visualizations are limited by the types of information graphs they support. Mainly for search results with no inter-unit similarity the presented visualizations wouldn't be applicable. Therefore we introduce transformation methods for the generated graph.

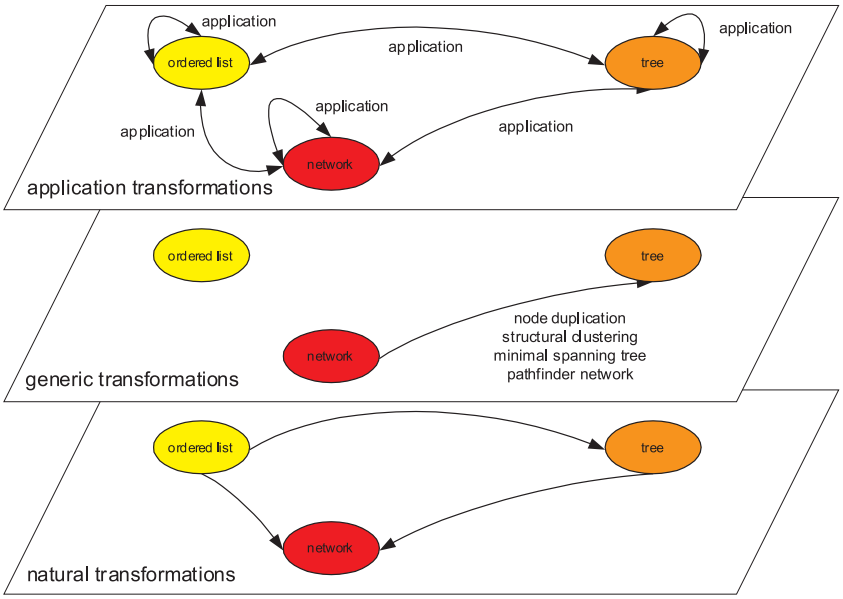


Figure 4.4: Layered Scheme of Transformations

The transformation methods in Fig. 4.4 make it possible to present the data with more visualizations. Converting from one structure to another always goes hand in hand with a loss of precision. It is obvious that a tree can be visualized with an algorithm for a general graph. Therefore we support natural transformations. On the other hand several transformation methods can be applied to an information graph independently of its domain. These methods are in the layer of generic transformations. Finally each data source and the application which generated the data has a kind of semantic knowledge about the information graph. The methods

in the layer of application transformations are defined by the data source at runtime.

A transformation is described by an input and an output type of information graph. The available transformations are those of the data agent including the natural and generic transformations supported by the framework.

Some examples how to use transformations:
- clustering
- minimal spanning trees
- filtration
- node duplication for a pseudo-tree structure
- pathfinder network

With these transformations we are able to have a wide range of possibilities to present the data. Now the system can find paths from the original graph layout to other structures. Each matching visualization can be presented to the user.

4.10 Use Cases

The use case diagram 4.5 shows the operations to be performed after a user interaction.

4.11 System Architecture

We aim to implement an application with several components to be loaded as plug-ins. These dynamic classes are visualizations, transformations, distortions and data agents. The application is divided in three layers. One for the data handling, the representation layer with the generation of a 3D scene graph and the application layer containing the user interface. An overview of the architecture is given in Fig. 4.6.

4.11.1 Plug-Ins and Properties

The system needs a properties file containing a list of all dynamic classes and other system settings. For the first implementation it is sufficient to have four lists stored as XML document in the filesystem. The client can request these lists and select one of the plug-ins, which is loaded afterwards by the classloader.

4.11.2 User Interface

In designing the user interface we will try to respect six rules developed by Nigay and Vernier [37].

R1: The representational systems must be easy to change(direct access).

R2: Temporal continuity must be guaranteed while changing the representational systems in order to provide visual continuity. The user should not be lost in the space because she/he switched from one representational system to another one.

R3: The representational system used to present the focus of interaction must be precise and not rely on a distortion function.

R4: If it is not possible to present the whole information space using one precise and global representational system, two representational systems can be combined, one being (partial, precise) and one (global, vague).

R5: If two representational systems are combined to present the information space, spatial continuity must be guaranteed between the two representations in order to provide visual continuity.

R6: The navigational tools must be uniform along the user interface (reusability of the navigation tools).

4.12 Summary

We have defined a general framework for the visualization of information graphs. We started with modeling data sources as information graphs and specifying the access to the data sources with neighborhood queries. The considerations about clustered graphs gave many ideas on how an information graph can be modeled. With this abstract interface we are ready to think about the visualizations. Some of them use trees or networks as input and the output is the arrangement of the nodes in the information graph onto a virtual environment.

Our platform is able to contain every kind of visualizations. This is possible because of the three-dimensional representation, the event model for the communication during the incremental approach to fetch data from the sources. An implementation has to point out the performance reachable by an application following our approach.

In each screen of the visualization we see a well defined part of the whole graph. This is done by the definition of a query on the data source and the number of vertices in the neighborhood related to the query. Looking at the navigation issues, we defined a event mechanism allowing the data source to provide any interactions. One special kind of interaction is the fetching of a new portion of the graph by setting a new query point. The transition from one logical frame to another can be animated to keep the mental map.

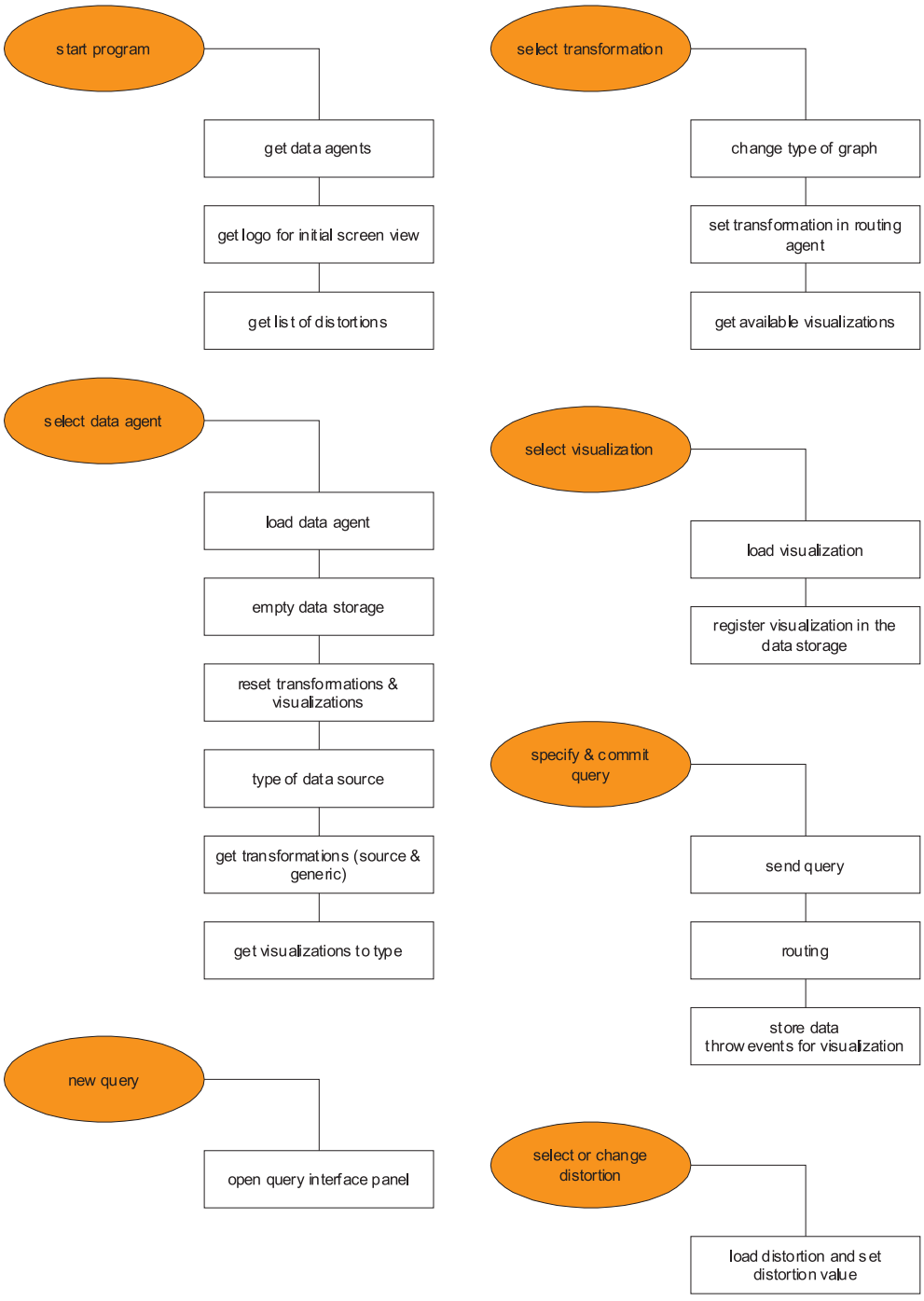


Figure 4.5: Use Cases

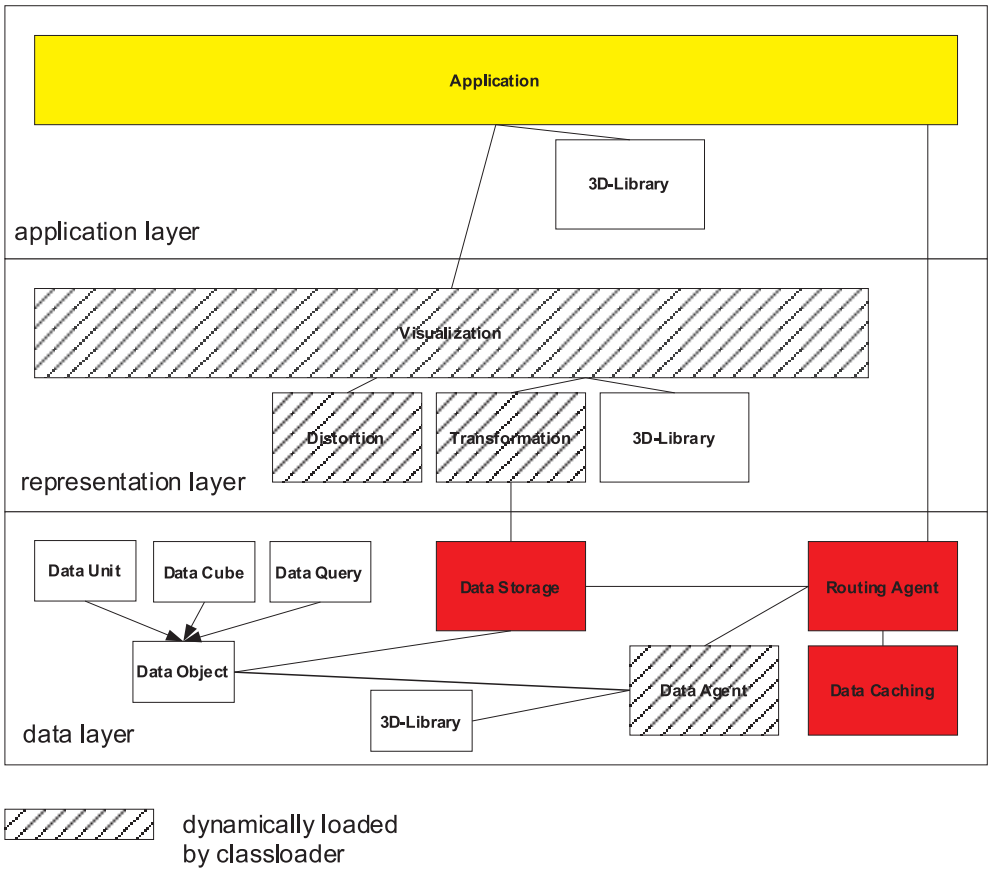


Figure 4.6: Architecture of the Framework (with its three layers)

Chapter 5

Implementation

In this chapter we describe the specialities of the implementation and give a developer an overview which tasks have to be done in writing extensions or new components to our framework. We also use the same examples as in the previous chapters to continue the illustration of the concepts. You can find introductions in almost each section describing the tasks to be done during an implementation.

5.1 Introduction

The current version of the information visualization framework is developed in Java, using a Windows2000 workstation, the Java3D and JAI-libraries for the three-dimensional visualization and image preparation for textures. More information about the libraries and infrastructure can be read in 5.2.3. The system has also been tested on RedHat Linux 6.2 using the Java3D implementation by BlackDown [*B*].

Now look at the scheme 5.1 of the data flow in the system. This will be the foundation of our explanations in this chapter.

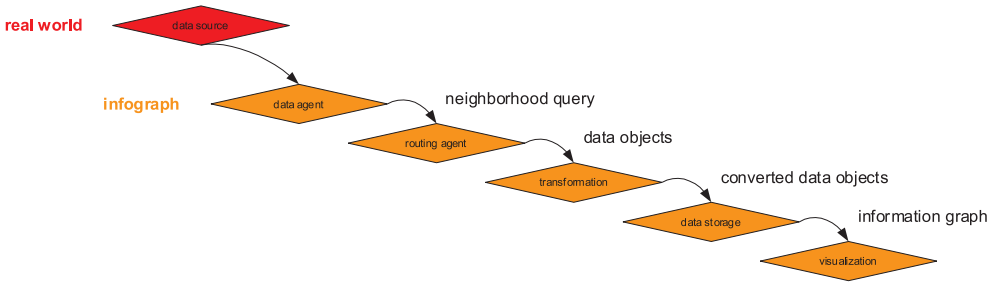


Figure 5.1: Data Flow in the Information Visualization Framework

The underlying packages of the framework look like this:

packages

	description
infograph.*	all framework classes
infograph.agents.*	collection of implemented data agents
infograph.visualizations.*	all implemented visualizations
infograph.transformations.*	all implemented transformations
infograph.distortions.*	all available and implemented distortions
infograph.j3d.*	support libraries (Java3D)

5.2 Application

We implemented a system called 'InfoGraph' in Java and several plug-ins to the framework during this work.

5.2.1 User Interface

The user interface of our application consists of three components. First there is the canvas for the visualizations, then the panel for the buttons and third the panel with the query interface and the selection of animations. (see 5.2 and 5.3)

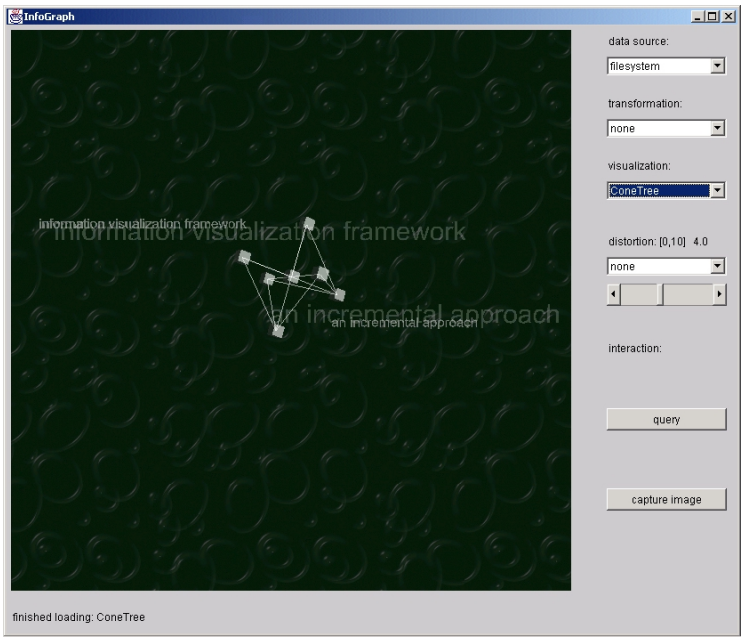


Figure 5.2: Screenshot: User Interface

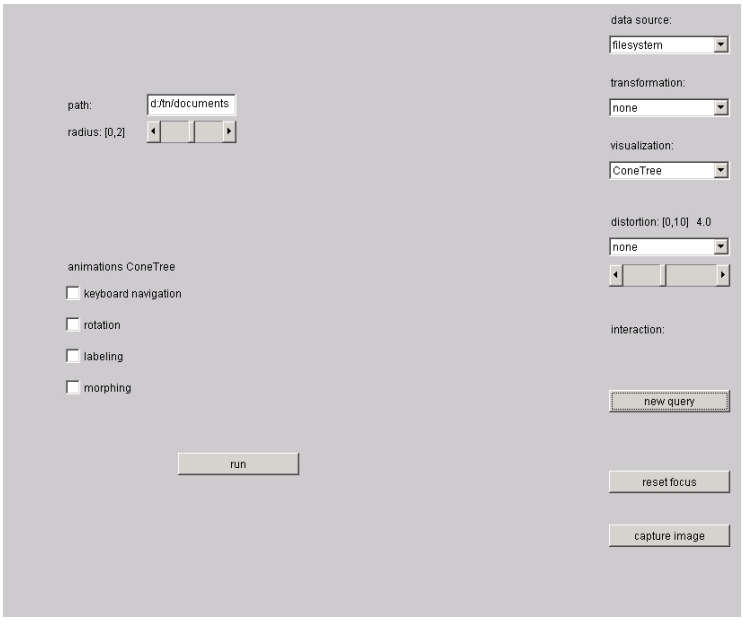


Figure 5.3: Screenshot: User Interface with Querying Options of a Filesystem

5.2.2 Properties

During the start of the application we read an XML-file with all paths used in the system. (e.g background image, capturing output path, offline-version of BiblioNet) Also the names of available plug-ins are defined with this file. Thus its very easy to add and change visualizations, data sources, transformations and distortions in the system. The file should look like this to be recognized:

```
<InfoGraphProperties>
  <backgroundImage>
    D:/cvshome/All4U/InfoGraph/Background0.jpg
  </backgroundImage>
  <BiblionetzBaseURL>
    file:///D:/Bibliothek/
  </BiblionetzBaseURL>
  <ImageDBBaseURL>
    http://dalmatiner.ethz.ch/isearch/bin/ImageSimilarity.tcl
  </ImageDBBaseURL>
  <CapturingOutputPath>
    c:/temp/
  </CapturingOutputPath>
  <Visualizations>
    <Visualization>
      ConeTree
    </Visualization>
    <Visualization>
```

```
        SpringForce
      </Visualization>
    </Visualizations>
    <DataAgents>
      DataAgent>
        FilesystemAgent
      </DataAgent>
      <DataAgent>
        ImageDBAgent
      </DataAgent>
    </DataAgents>
    <Distortions>
      <Distortion>
        RadialDistortion
      </Distortion>
      <Distortion>
        OrthogonalDistortion
     </Distortion>
    </Distortions>
  </InfoGraphProperties>
```

5.2.3 Installation

For the installation of the framework and the application InfoGraph, certain Java libraries have to be installed. The Java3D extension of the virtual machine has to installed with the setup available at SUN [G]. If Java3D should be executed on Linux systems, the implementation by BlackDown [B] can be taken. We need two other libraries: JAI (Java Advanced Imaging) for the manipulation of images and JDOM for the reading and parsing of XML-files.

libraries

name	version	description	reference
Java2 SDK	1.3.0_02	Java Enterprise Edition Platform	[G]
Java3D	1.2.1_03	Java 3D API for 3D graphics	[H]
JAI	1.1.1	Java Advanced Imaging API	[F]
JDOM	Beta 7	Java Simple API for XML (SAX) and the Document Object Model (DOM)	[D]

The command line to run InfoGraph must have the following structure:

```
java -Xmx256mb -classpath $CLASSPATH infograph.Infograph
./properties.xml
```

Ensure that the classpath contains all libraries stated above and that the correct path of the properties file is given. The parameter Xmx defines the amount of memory

allocated for the virtual machine. On a linux system we additionally have to specify the location of the Java3D library with the option:

```
-Djava.library.path=/usr/java/j2re1.3.0/jre/lib/i386
```

Otherwise the Java3D extension cannot be found and activated.

5.3 Information Graph

5.3.1 Data Objects

An information graph consists of data objects as vertices and each vertex has a vector with references to its neighbors. Each data object has a unique id and the subject-URL pointing to the entity in the data source. Further each object has a text for the brushing.

For each data agent it should be possible to add signs to data objects. If none is specified the visualization will add a standard sign. The methods '*setRepresentation*' should be used to add signs to data objects. This should not be mixed up with the variable '*sceneGraphObject*', which is only used by the visualization to hold the current visible sign. This is also true for the variables 'label' and 'position'. A visualization algorithm often has to store additional information for each data object. Therefore we can assign an arbitrary Java-object to each data object.

setRepresentation

sceneGraphObject

There are two types of links to other data objects to distinguish. The vector '*associations*' contains all associations to the neighbors of an object with a measure for the similarity. The second vector is called '*outgoings*' which is a subset of 'associations'. In this vector we only have the associations which we have taken during the routing. How this works is written in 5.6.

associations

outgoings

5.3.2 Different Data Objects

The information visualization framework knows three types of data objects. The data units, data cubes and data queries are the extension of a general data object (see 5.4 for the class diagram). For two reasons we distinguish different types of data objects. It gives the possibility to present them with different signs in the visualization and to add special functions. So the visualization can plot signs based on the type.

For the data cubes we provide the functions fold and unfold, meaning the hiding of all data objects contained in the data cube. These function calls are propagated recursively to all objects contained in the data cube and in each step of the recursion there is a call of the fold/unfold method in the visualization. Thus the drawing can

be adapted. Additionally the data units are weighted by a value in the interval of]0,1] to show the importance of the unit with respect to all others. In the context of a filesystem this can be for example the filesize or the access frequency.

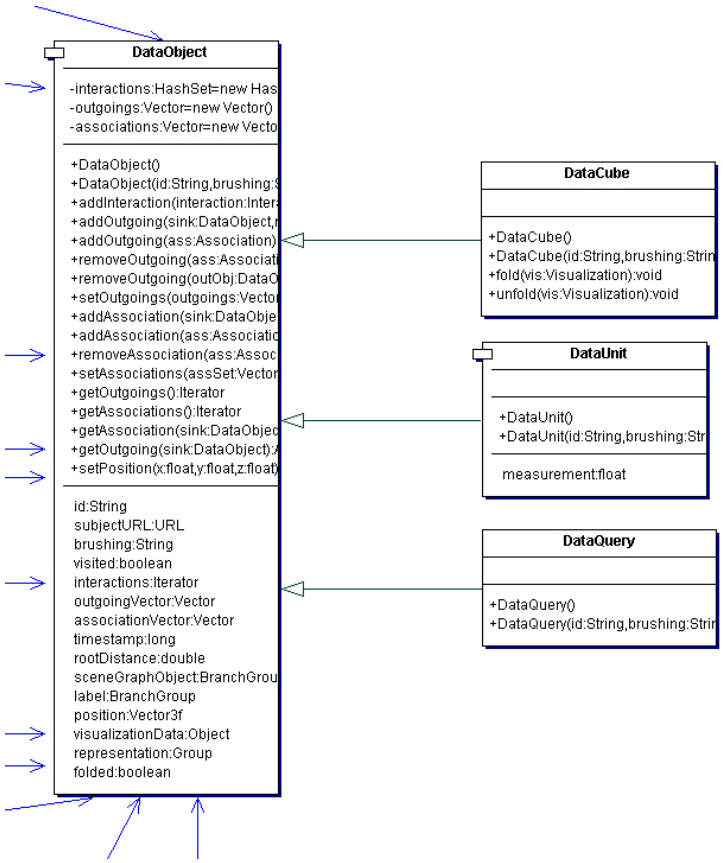


Figure 5.4: Part of the Class Diagram showing the Different Data Objects

5.3.3 Implementation of an Information Graph

With the data object above we can implement an information graph simply as set of data objects, because each object knows about its neighborhood. We need functions for the calculation of distances in the graph between two data objects and methods to set/get the center object of the information graph. This data object should be placed by a visualization at (0,0,0) and can be used to traverse the whole graph fast.

5.4 Data Sources

For each data source, we have to implement the abstract class called infograph.DataAgent. The implementation of data agents are packaged into infograph.dataagents.*. To show how this can be done, we describe the data agents implemented in our application.

The main functionality for a data agent is the evaluation of a query. The result is an information graph with the query object as center object and the query results linked by an outgoing association. For each new routing cycle it starts with a query, where the observation point is not set. Afterwards the routing agent recursively fetches the neighborhood of the query by sending the same query with an observation point set. The data agent then has to deliver the vertices similar to this special point. You can find an example of this in the examples in section 5.4.1.

```
public abstract Graph evaluateQuery(Query q)
```

Because of the internal structure, we must have a mechanism to specify individual querying options. Therefore each implementation of a data agent has to extend the classes infograph.Query and infograph.QueryInterface. The query interface is a panel displayed for the input specification by the user. So we have to add the GUI elements for this definition. (e.g. scrollbars, fields, options) When the user has finished with the preparation of the query, the query interface generates with the input of the user a query, which typically is an extension of the class infograph.Query. This extension is necessary because of the individual querying options for a data source. Further the data agent method to execute an interaction has to be implemented:

```
public abstract void executeInteraction(Interaction interaction,
    RoutingAgent rA)
```

How to implement a data agent?

- define the querying by implementing the class infograph.Query and its evaluation in the data agent.
- design a query interface as extension of infograph.QueryInterface and set the GUI elements on the panel from this class.
- add application specific transformations to the data agent
- to each data object: add interactions and implement their execution
- add data agent to properties.xml

5.4.1 Filesystem

The neighborhood queries in the filesystem are very simple to describe and to implement. Corresponding to the definition of an information graph, we take the hierarchy of folders as the tree of data cubes. Thus if the query points to a folder, the neighborhood is defined by the child folders (data cubes), the files in the folder (data units) and the parent folder (data cube). Because there is no similarity measurement available in the data source we define the values to be 0.5 and 1.0 as shown in figure 5.5. These values are chosen for the following reason. The default value is 0.5 for our association and the value 1.0 for the parent folder shows that the path upwards in the filesystem is of lower interest.

First we have to extend the class infograph.Query to hold the path requested by the user as starting point of our search. Therefore we have to add the methods 'setFilename' and 'getFilename'. When the data agent receives a query object, it can read the requested path. In the context of the filesystem, routing should be limited by a radius which is computed as the sum of association measurements on the shortest path from the root object. Thus the query interface is not complicated as shown in figure 5.9. There is a text field for the starting path of the desired view and the radius as limitation of the data fetching. Finally, we have to look at the signs and interactions for the data objects. The sign, which is the representation of the object in the virtual environment, can be the default defined by the visualization except for data units (files) standing for an image. This image is wrapped as texture on the surface of a sphere. So it's very easy to see images among the other files. For each data object we additionally add an interaction to begin a new query with the current object as starting point.

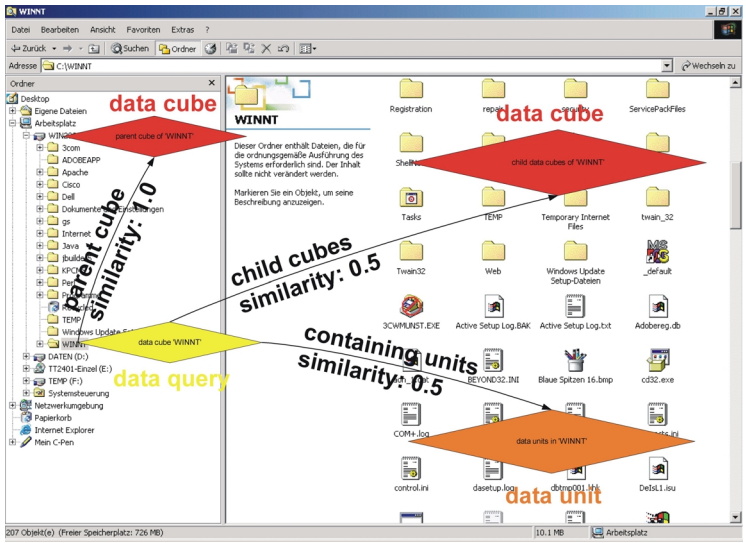


Figure 5.5: Example of a Data Source: Filesystem

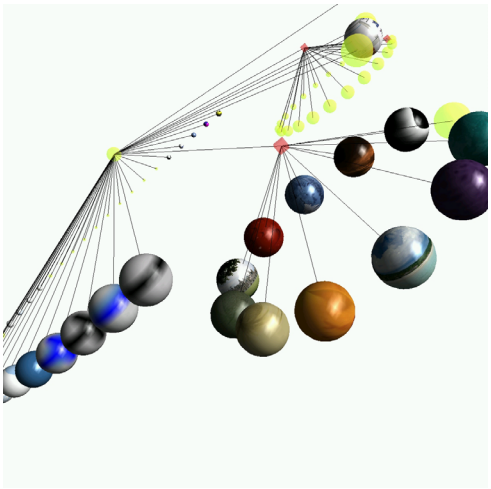


Figure 5.6: Filesystem in ConeTree with Textured Spheres for Images

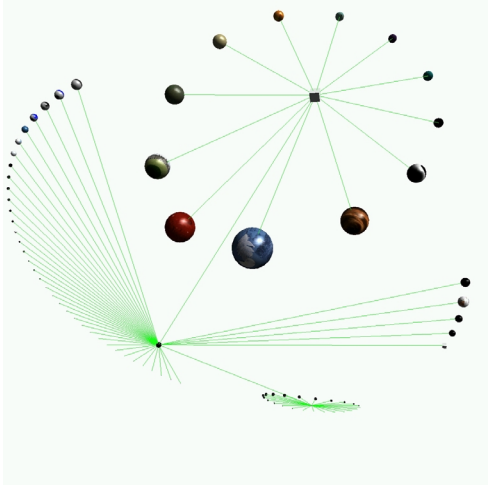


Figure 5.7: Filesystem in Balloon2D with Radial Distortion (same data set as in 5.6)

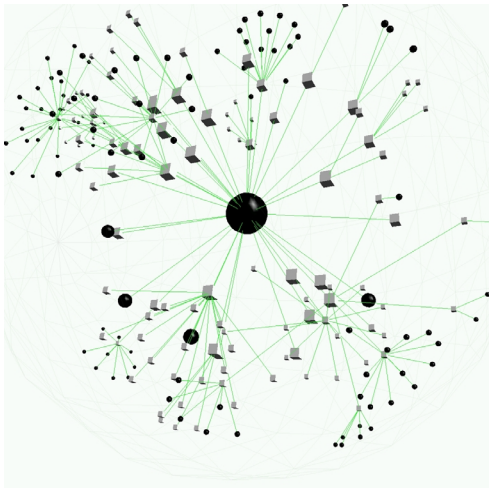


Figure 5.8: Filesystem in Radial3D with Hyperbolic Distortion

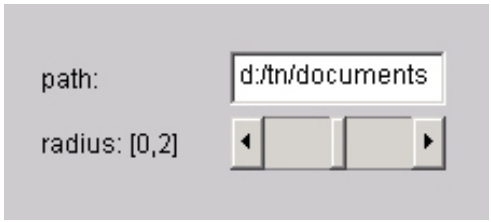


Figure 5.9: Query Interface for the Filesystem Agent

5.4.2 BiblioNet

BiblioNet is a collection of books, persons, notes and thoughts developed by Beat Doebeli [*A*]. It contains summaries and explanations about the topics: books, persons, subjects, terms, questions and statements. The entries within the BiblioNet are linked by their inter-relationship. The system is currently a set of static html-pages generated by an MS Access database for the administration.

For our information visualization system we implemented a data agent looking at the persons, books and terms in the BiblioNet. For some of the entries, there is an image of the person or the cover of the book associated. We combine these results with queries on the Amazon data collection.

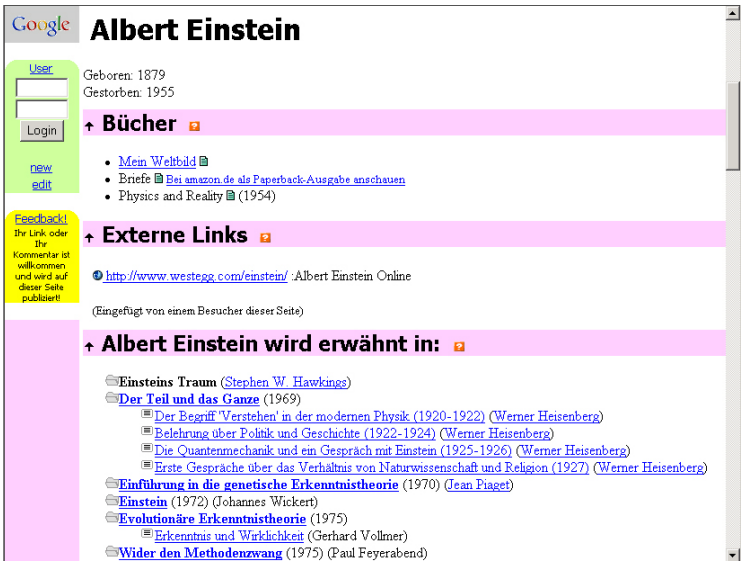


Figure 5.10: Example of a Data Source: BiblioNet

Our data agent starts with a full-text search over books, persons and terms and generates the answer for the first query with these search results. In the next iteration of routing we get the pages of one entry in the BiblioNet and follow the links. All linked entries are defined to be the neighborhood of an entry. With this example we additionally want to explain the application specific transformations. Our data agent has a transformation called 'Only Books and Persons' acting as a filter to add only entries of this types to the data storage and because of that to the visualization.

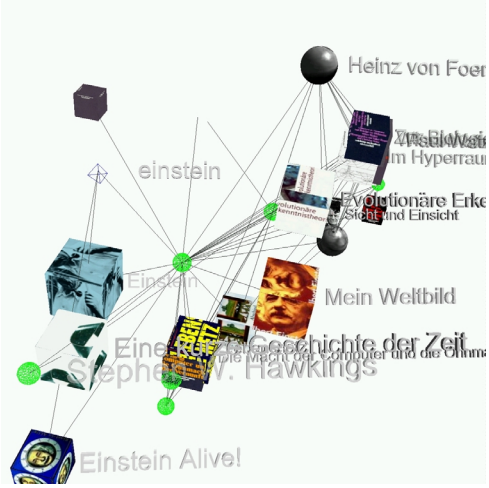


Figure 5.11: BiblioNet in Spring-Force Visualization

Figure 5.13: BiblioNet on the Query 'Mind Map' in SOG Visualization

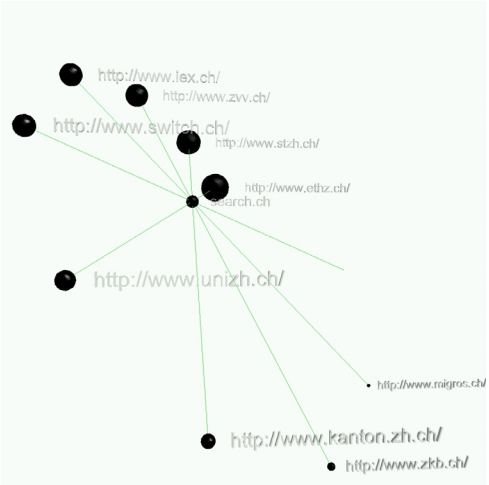


Figure 5.12: Search Results of Query 'zurich' visualized with Radial2D

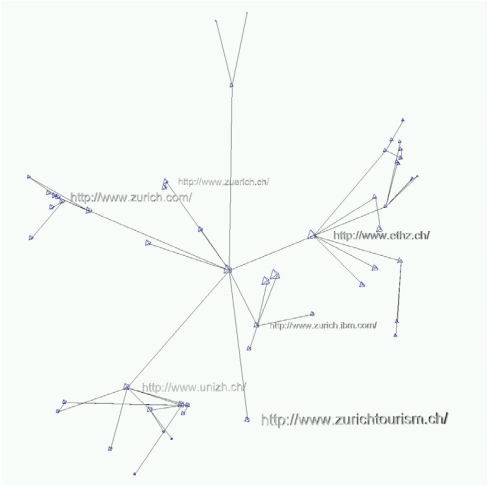


Figure 5.14: Google Search extended with Reverse Linkage (Query: 'zurich')

5.4.3 Search Results

A very simple data source are the search engines all over the internet. Our first implementation of a data agent is an interface to the Swiss search engine 'search.ch' [*E*]. The extension of the class infograph.Query has to hold the keywords and define the number of results to be fetched from the search engine. To see the order of the search results we weight the outgoing associations from the query with a linear descending value.

To make it more useful we added more information to the search results. Our second implementation is based on Google (www.google.com) where we used the possibility to query for pages linking another. So we first request for the search results to some keywords and parse the resulting URL's. In the following rounds of routing we ask for each URL the pages with a link to it. Thus the resulting information graph can be of the type 'network'. The advantage of such a visualization is to enable the user to recognize the relationships of search results based on their linkage.

5.4.4 XML-Files

Another kind of data source we look at is an XML-file. These semi-structured files are per definition built in a tree-like construction. The benefit resulting from the visualization of an XML-file is to have an overview in a three-dimensional environment which allows to recognize similar groups of nodes and the overall structure.

We implemented the data agent using JDOM for the reading and parsing of XML-files. Each data object is determined by the filename and an XPath-like expression. The difficulty to implement this agent was that in an XML-file it is allowed to have more than one node with the same name in one level of the document. Because of that we numbered the children of a node. Now we are able to exactly locate a node in the XML-tree structure. As defined before we have to assign a data cube object to each node of a tree in the data source. In the example of XML-files each node is a data cube and the attributes are data units.

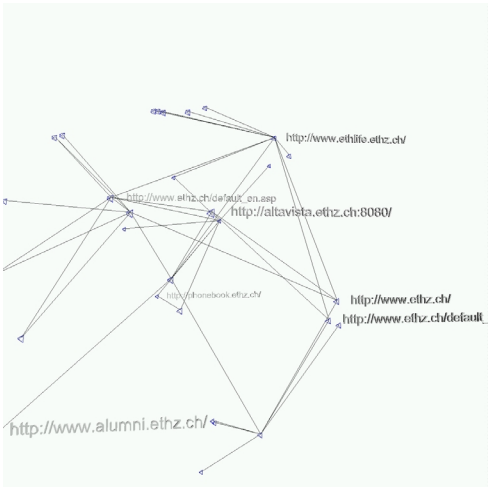


Figure 5.15: Linkage started from 'www.ethz.ch' (visualization: SOG)

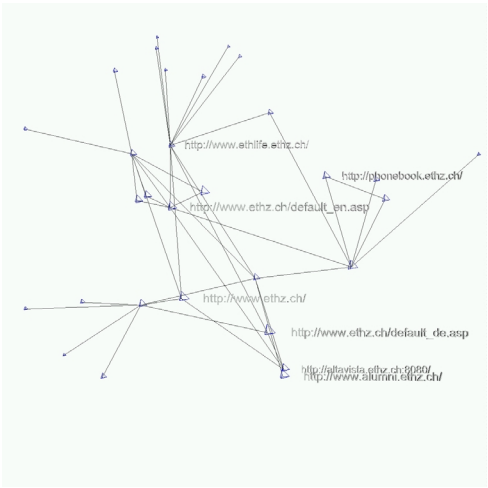


Figure 5.17: Linkage started from 'www.ethz.ch' (visualization: Spring-Force)

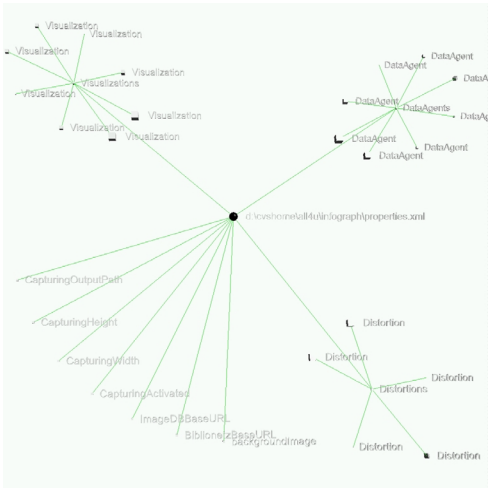


Figure 5.16: Properties-File (XML) of our System (visualization: Balloon2D)

5.4.5 Web Linkage

Very similar to the idea of showing the reverse linkage in the case of search engines, it is interesting to present the structure of the world wide web based on the outgoing links within html pages. The information graph resulting from our data agent can be a large network with each URL as a single node. Therefore we only take the first n links from each page. The parameter n can be defined in the query interface to the linkage data agent. We present one example in Figure 5.15. It is the reachable region of the internet starting from 'www.ethz.ch' rendered by the self-organizing graph algorithm.

5.4.6 Image Database

The Database Research Group has developed an image database which provides similarity searching using different features. A feature, for instance, color moments, summarizes the contents of an image. More information about this project can be found at [C]. For this data source we can calculate a similarity value for each pair of images and thus we receive a complete network (proximity matrix). For most of the visualizations this is too much information because of the complexity. But in this case the algorithm 'FastMap' (3.2.4, 5.8.9) can be used. For a good representation in three-dimensional visualizations we project the images as textures on the surface of boxes. Thus the image is visible from every direction.

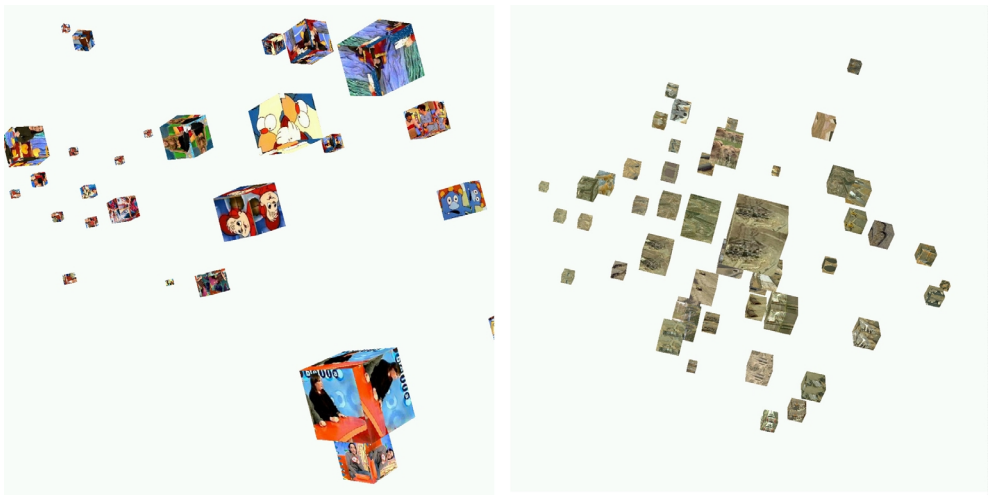


Figure 5.18: Sample Pictures of Image Database 'Chariot' drawn with FastMap

5.5 Data Storage

The data storage is an extension of our information graph with additional functions for the 3D visualization. The instantiation of a new data storage forces the creation of a new 3D canvas. The function '*setVisualization*' is then used to bind a visualization to this canvas. This is finished by the call to '*initVisualization*', where the navigation behavior and the initial scene graph of the visualization is set to the canvas. The functions 'addVertex', 'setCenterObject', 'remove', 'startRouting', 'endRouting' and 'clear' are called by the routing agent through a transformation. This immediately effects the graph and by the event model the visualization. We implemented this event model through method calls in the visualization and transformation. To avoid a big overhead with unnecessary method calls, the event subscription is made with boolean flags.

setVisualization
initVisualization

Example: If the routing agent finds a new vertex, it sends this data object through a transformation and then to the data storage. Here the data object is added to the information graph and if the visualization has registered the add event, the corresponding event is thrown.

5.6 Routing Agent

The routing agent implements a second thread, because the fetching of data should be interruptable by the user.
We describe once more the concept of routing and here with more details. The term 'routing' means the data fetching of all linked data objects in a well defined region. This region is defined by a starting point and a radius, whereas the radius can be chosen as the sum of association weights from this point or a number of objects. Now we present the pseudo-code for the main loop in the routing agent.

Algorithm *Routing*
(∗ precondition: data storage, data agent and transformation assigned ∗)
1. transformation.startRouting()
2. priority queue P
3. **if** query.getObservationPoint() $\neq$ null
4. **then** put observation point to P
5. **else** put dummy object to P
6. **while** $P \neq \emptyset$ **and not** *stopped by user*
7. **do** let q be the first element of P
8. query.setObservationPoint(q)
9. graph G = dataAgent.evaluateQuery(query)
10. let c be the center object of G
11. **for** $i \leftarrow 1$ **to** $|c.getOutgoings()|$
12. **do** $a \leftarrow$association[i]
13. **if** $a.sink() \in DataStorage$
14. **then** convert a to a regular association

```
15.                    (* because association is not followed, but should be vis-
                       ible in graph *)
16.              else  if radius or count not greater than threshold
17.                    then add a.sink() to P
18.                    else  remove a from c
19.        transformation.addVertex(c)
20.  transformation.endRouting()
```

5.7 Transformations

The transformations are the filter between the routing agent and the data storage. Each object which should be added to the data storage has to pass a transformation. For the easiest case, no transformation to apply, we extended the abstract class infograph.Transformation to '*zeroTransformation*'. Here the events addVertex, setCenterObject, startRouting and endRouting, which have to be implemented for a transformation, are implemented to pass data objects immediately to the data storage.

zeroTransformation

The natural transformations can be applied to some graph types to convert them into another type without changing the structure. One example is the visualization of a tree as general network. In contrast to the 'zeroTransformation' the output type of the transformation is set. The data objects are still passed through without any changes. We also implemented a general transformation which takes a general network and removes all associations from the data objects except the outgoing ones. Thus we can see the paths taken in the routing to fetch the whole information graph. We called this transformation '*spanning tree*'. The resulting information graph is a tree and hides all the non-tree links as introduced by Munzner in the Hyperbolic Tree visualization. [36]

spanning tree

For some applications it could be necessary to temporarily store all objects in the current logical view before the first vertex can be added to the data storage. Then we can add an object cache to the transformation and keep all objects. In the 'endRouting' event all objects can be added to the data storage.

5.8 Visualizations

At long last we present the description of the visualizations. In the abstract class infograph.Visualization we can find the abstract methods to be implemented for all the events described in 4.3. Further there is a boolean flag for each event a visualization can set to activate the method calls. Some of the functions used by each implementation are in this class for the construction of the screen representation.

The construction of a scene graph is the main task for the visualization. The first step of the construction is made by the data storage initialization with the instantiation of the 3D canvas and the navigation behavior. The result of the

initialization is a new scene graph, which can be completed by the visualization. The second part of the construction is to build an initial screen (lighting, rotation of the environment, background, etc.) for the visualization. Since the Java3D scene graph is a tree, we have to keep references to several points in it. One of these references is the '*rootTransformation*' . That's the transformation group of the first transformation in the scene graph and this is also the first node in the scene graph constructed by the visualization. Typically we use it for the rotation of the environment. To this node we can add any Java3D objects.

root Transformation

After the building of the initial screen we must have a node in the scene graph to add the signs of our data objects. This should be the leaf of the scene graph up till now. Therefore our initialization has to keep a reference to this '*initialLeaf*'. Now we are ready to add signs for each data object. This is normally done in the implementation of the method 'addVertex'. As we know from the data agents some of the data objects can have a predefined sign. To all others we must add a default object. For each data object we once have to call the function 'createPosition' of the abstract class visualization. There we create a new BranchGroup and add to this a TransformationGroup. The BranchGroup is used to add the sign to the 'initialLeaf' and the TransformationGroup keeps the translation in the spatial arrangement for the position. Finally the sign is added to the TransformationGroup.

initialLeaf

If the visualization has to change the position of a data object we should call the method 'setPosition', because the this method handle everything concerning the distortion which will be described later. The construction of the spatial arrangement is done by the implementation of the methods corresponding to the events.

How to implement a new visualization?

- implement infograph.Visualization:
- construct a scene graph for the initial screen (lighting, rotation of the environment, background, etc.)
- implement methods for each event
- implement methods for the interactions (mouseOver, mouseOut, runAction, mouseClicked)
- implement methods for folding and unfolding
- implement the function 'changeDistortion' (e.g. for the repositioning of visible edges)
- add the visualization to properties.xml

5.8.1 Brushing

level of detail

Another task for the visualization class is the management of labels for the brushing. We decide to implement a *level of detail* brushing, meaning that objects near the eye position are labeled and the others are not. This is part of our focus+context approach and is supported by the Java3D LevelOfDetail-Behavior. This behavior consists of a switch group with an array of distances. Depending on the distance from the eye position to the data object this behavior select the suitable representation from the switch group. There is one function called '*labelingFunction*' doing this. A visualization can call this function with an integer value to select the action.

labelingFunction

labeling

1	create a new label
2	update the position of the label
3	remove the label
4	attach the label to the scene graph → set visible
5	detach the label → hide
6	set non transparent
7	set transparent
8	show permanent
9	show corresponding to level of detail

For the labels we use a transparent three-dimensional text combined with an OrientedShape3D object. The transparency is used so that nothing in the drawing is completely covered by the labels and the oriented shape is always aligned to the viewing plate and therefore always readable.

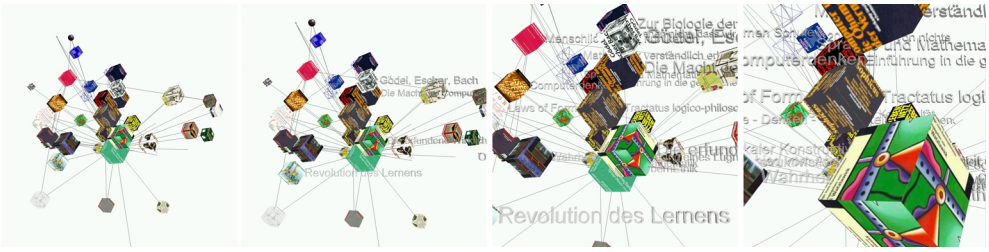


Figure 5.19: Brushing with Level of Detail Behavior during Zooming

5.8.2 Animations

Our framework can keep a list of animations for each visualization. When the user defines a new query, he can also activate these animations. A visualization plug-in can check if the user has activated an animation and change the drawing adequately.

morphing
keyboard navigation
brushing

For each visualization we want to have three typical animations. Thus *morphing*, *keyboard navigation* and *brushing* are added by default to all visualizations. The brushing is implemented in the 'labelingFunction' described in the previous section and the keyboard navigation is supported by Java3D and can easily be added during the initialization of the visualization. The morphing functionality is special to our incremental approach, meaning the animated change from one to another logical frame. The visualization has to be able to present at each time the available data and integrate new objects dynamically in the past arrangement. So a visualization has to provide two running modes. One for the situation that morphing is activated and another without. The second situation is much easier because the calculation of positions can be done for all objects at the same time, whereas the incremental building has to run often.

5.8.3 ConeTree

The ConeTree places the center of the information graph at the root of the virtual environment and, recursively, all the children at equal distances along the base of a cone. In our implementation we don't take the equal distances but we multiply this distance with the weight of the association. So we can show the similarity of objects by the distance to the parent node.

For the default sign of the data objects we took transparent objects so that even if they are covered, they are visible. We added an animation to our ConeTree visualization taking the callback function 'mouseOver' for the rotation of the children. Thus it's easy to explore the graph by rotating the part of nodes the user is interested in. Figure 5.21 shows the rotation of a subgraph and the changes in the focus+context sense.

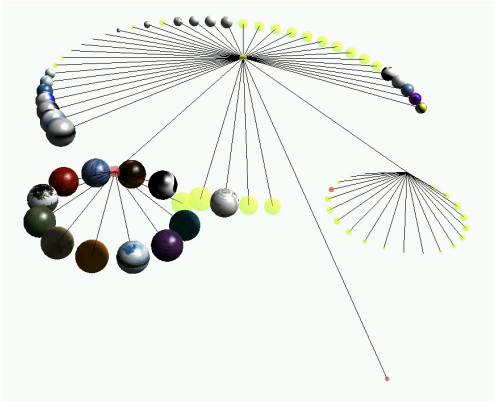


Figure 5.20: ConeTree applied to Filesystem

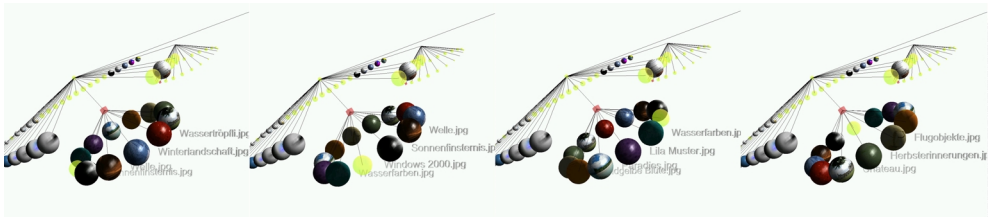


Figure 5.21: ConeTree - Rotation of Objects

5.8.4 Spring-Force Model

We also implemented a modified version of the force-directed algorithm GEM by Frick et al. [15] The spring forces are adapted by taking all objects from the data storage. For each data object we have to do the calculation of the attraction forces for each associated object, and to all others, the repulsive forces. The cumulative impulse is taken for the repositioning of the data object. This calculation is done in the method 'endRouting' if the morphing is not activated, otherwise in regular intervals after the insertion of new objects.

In the spring-force algorithm we add an animation called 'lines' to activate the drawing of lines along the edges. This is useful for simplificaton if the information graph is very dense. Figure 5.23 shows two examples of the BiblioNet once with lines and once without them. Other examples of the algorithm are the Figures 5.14, 5.17.

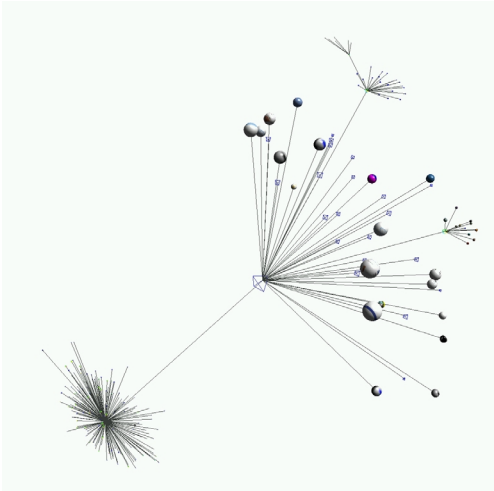


Figure 5.22: Spring-Force Algorithm applied to the Filesystem

Figure 5.23: Spring-Force Algorithm with the animation 'lines' (data source: BiblioNet)

5.8.5 Radial Layout 2D

The radial visualizations are quite simple to implement, but provide a nice looking tree drawing. It's necessary to have two iterations over all objects to calculate this representation. The first run is for the recursive calculation of the children in each node. Thus we are able to estimate the space used for all children in traversing the tree. In the second step we calculate the positions for the data objects by doing a breadth first search and keeping the angle which is used up to now. The position can be calculated by this angle and the radius which corresponds to the distance from the center object in the information graph.

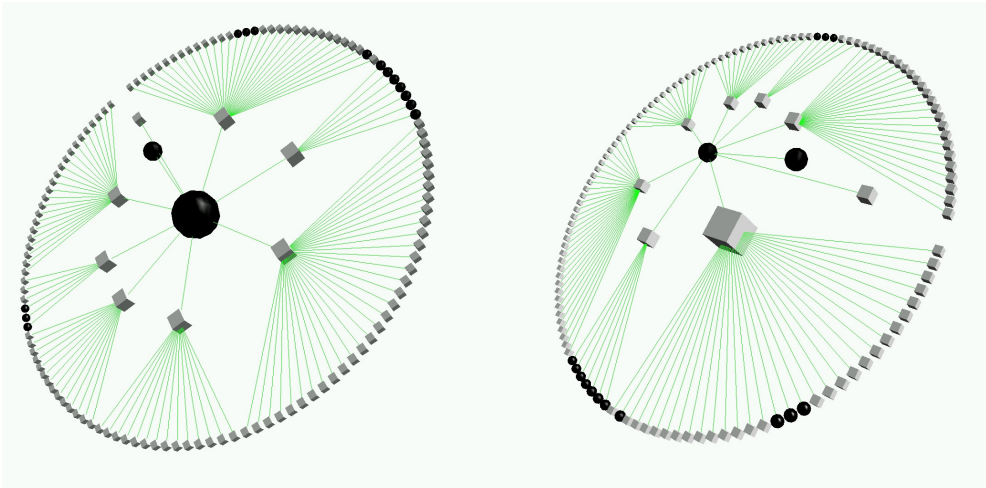


Figure 5.24: Radial2D distortion-free and with Focus set in RadialDistortion

5.8.6 Radial Layout 3D

This algorithm is very similar to the Radial2D visualization because it is an extension of it to three dimensions. We also need the two iterations over all data objects and estimate the space used to place all items in a sphere. For the calculation of the positions we looked at the implementation of the HyperbolicTree [36], because the radial three-dimensional arrangement is the base of the other algorithm. In the HyperbolicTree this arrangement is calculated in the hyperbolic space and projected to the Euclidean.

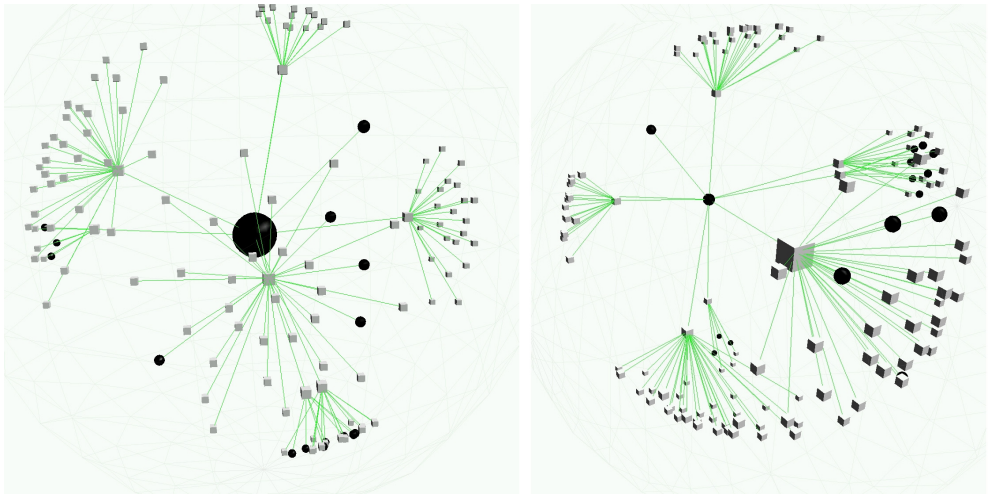


Figure 5.25: Radial3D distortion-free and with Focus set in RadialDistortion

5.8.7 Balloon Layout 2D

The last visualization of this type and methodology is the Balloon2D. The procedure for the calculation is similar again to the Radial2D, but the placement of children is not in a straight line from the center. They are arranged in a circle around the parent node, which is similar to the ConeTree implementation. Thus the difference to a ConeTree is the estimation process for required space. This kind of calculation is described by Carrière and Kazman in detail [7].

5.8.8 Self-Organizing Graphs

The drawings produced by the Self-Organizing Graph (SOG) algorithm are quite similar to those produces by the force-directed method, but the calculation is completely different. The SOG is a clustering algorithm based on self-organizing maps (SOM).

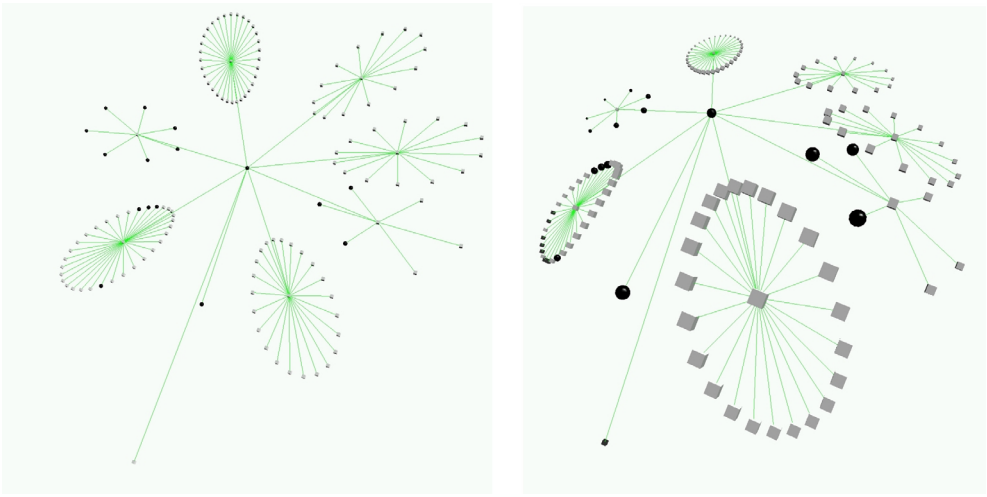


Figure 5.26: Balloon2D distortion-free and with Focus set in RadialDistortion

"Self-organizing graphs are a novel approach to graph layout based on a competitive learning algorithm. This method is an extension of self-organization strategies known from unsupervised neural networks, namely from Kohonen's self-organizing map." [34]

Each iteration of the calculation process takes a random point in the three-dimensional space and looks for the nearest data object. With a breadth first search we calculate for each item, in the neighborhood of this data object, an attractive translation to this random point with decreasing effect for the more distant objects.

5.8.9 FastMap

The FastMap algorithm was developed for the clustering of data objects based on a similarity matrix for complete information graphs. In our framework we also allow not so dense graphs, and the description of the information graph is not available by a proximity matrix (distance matrix). Therefore, the first step in the FastMap algorithm is to construct such a proximity matrix, which is very simple for a data source providing a complete graph, but more difficult for example for a tree structure. In this case we fill the matrix with the association weights available in the information graph and all other distances with a large random value. This little randomization is necessary, because otherwise the FastMap algorithm often maps two data objects to the same point. The FastMap algorithm can run for two or three dimensional coordinates and we implemented an animation to activate the two-dimensional variant. The implementation of the core clustering algorithm in Java was done by Kai Jauslin.

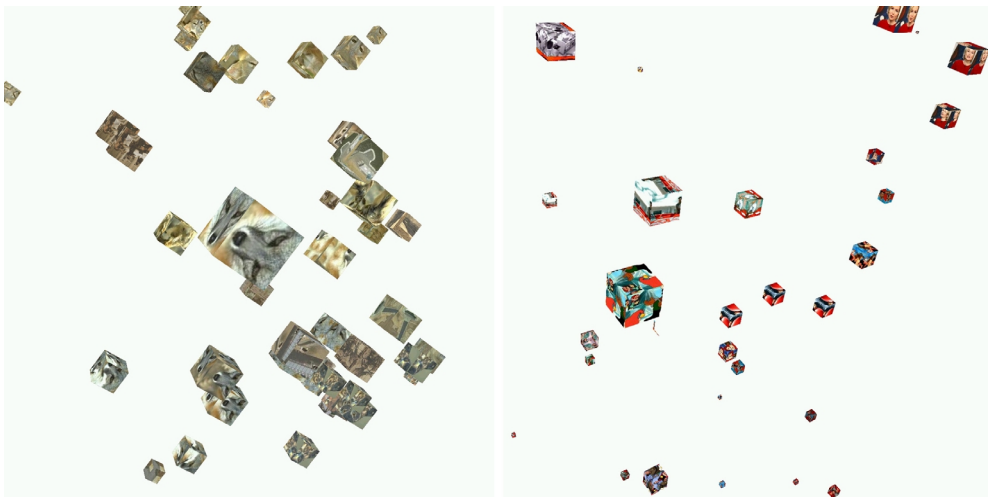


Figure 5.27: FastMap Algorithm applied to Image Database

5.9 Distortions

The distortions, as described in the chapter 4, are rearrangement functions independent from any visualization to support the focus+context approach. We implemented a set of distortions. A distortion consists of a change in the position and size of the visible signs. These two operations are automatically performed by the main visualization and the distortion classes. A system developer only has to care about the drawing of all additional elements like lines or boxes to place them onto the distorted position. The visualization can get the distorted and non-distorted position for each data object by calling 'getDistortedPosition(obj)' and 'getPosition(obj)'.

Each implementation of a distortion function has to respect two parameters: the distortion value, which can be changed by the user for the amount of distortion, and the focus point to plot this item in the root of the environment.

5.9.1 Radial / Fish-eye

The radial distortion, also called Fish-eye distortion, is described in detail by [27]. It scales the distance from the focus point while keeping the direction. For each position we first have to calculate a translated point corresponding to the translation from the original position of the focus point to the root.

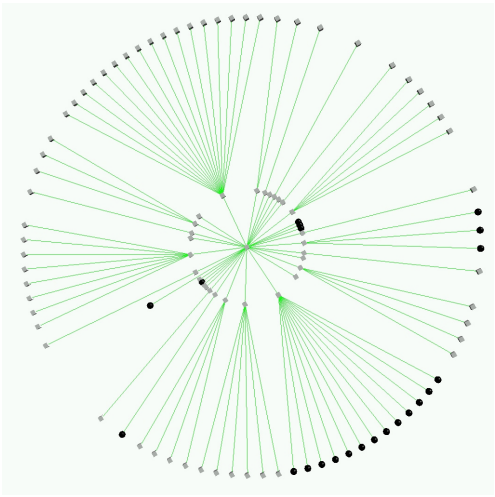


Figure 5.28: Example without Distortion

Distortion function for all directions:

$$f(x) = \frac{(distortionFactor + 1) * x}{distortionFactor * x + 1} \tag{5.1}$$

Scaling function:

$$s(x) = 8 * (1 - f(\frac{position.length()}{distortionRadius})) + 1 \tag{5.2}$$

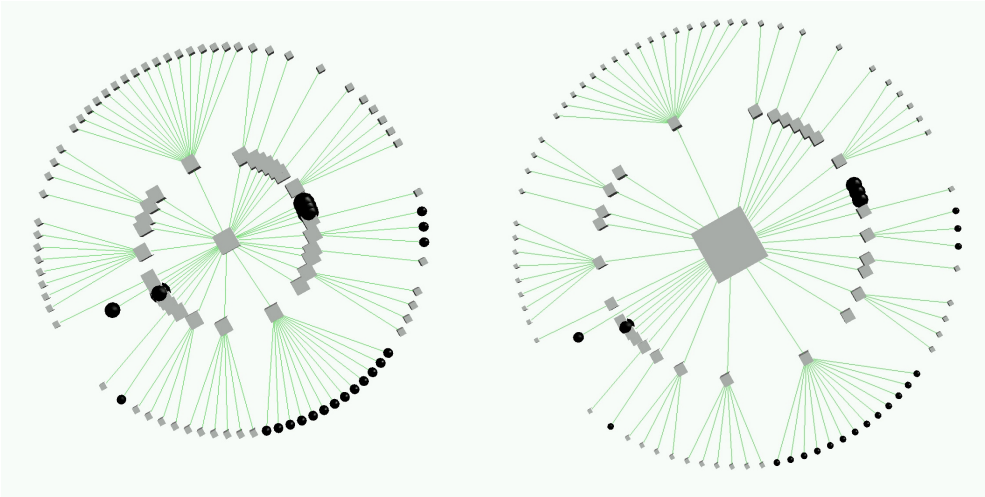


Figure 5.29: Radial Distortion with Factors 3 and 6

5.9.2 Orthogonal

The orthogonal distortion [27] takes a cube instead of the sphere projection we can find in the radial distortion. Each dimension is taken separately and the distortion function applied to it. The formulas are equivalent to the ones in the radial distortion.

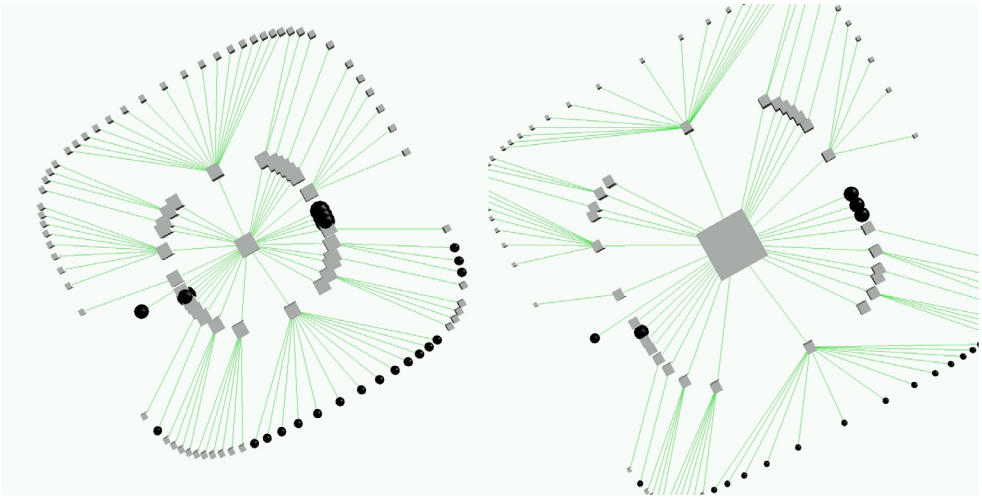


Figure 5.30: Orthogonal Distortion with Factors 3 and 6

5.9.3 TanH Distortion

Alternatively to the distortion function in the radial distortion we can choose the tanh function. The distortion function then looks like this:

$$f(x) = \tanh(\frac{distortionFactor}{2.0d} * x) \tag{5.3}$$

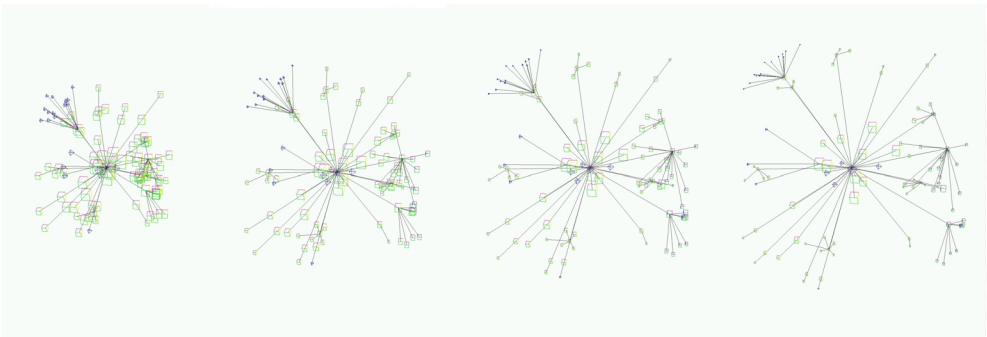


Figure 5.31: Series of Drawings with TanHDistortion for different Distortion Values

5.9.4 Plane Distortion

If we ignore one of the three dimensions, we get a simple projection to a plane. Exactly this is done by the plane distortion. The advantage in taking this projection as distortion is that not every visualization has to implement it and the user can switch between a three- and two-dimensional drawing.

5.9.5 Hyperbolic Distortion

The hyperbolic distortion is used by many visualization systems because it provides a natural focus+context representation. The geometry of a hyperbolic space is well described in [33] and also the projection from it to an Euclidean space.

Our distortion takes the original positions of the data objects as they were in a hyperbolic space and projects these coordinates to Euclidean. This is done by taking the point in 3D, extending it to 4D with 1 as fourth coordinate and then projecting it back to three dimensions.

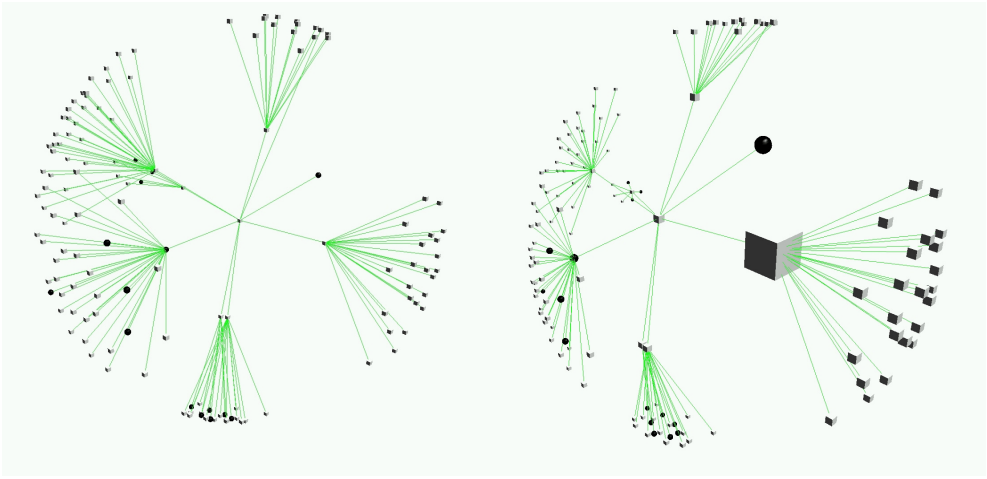


Figure 5.32: Hyperbolic Distortion and Radial3D Visualization with Focus Point

5.10 Interactions

We can add interactions to each data object by the data agent. By default the framework adds to all data object the interaction ‘open’. If the open interaction is triggered by the user, the system starts the browser and calls the subject-URL of the data object. Additionally, to all data cubes the interactions ‘fold’ and ‘unfold’ are added and executed by the system itself.

An interaction contains a name and a query. The name is used for the differentiation. A typical interaction is 'new query' which should be added to each data object for the data fetching started at this vertex in the information graph. The execution of the interaction has to be done by the data agent, because it may require knowledge of the source application. The support of interactions is included in the navigation behavior. It detects the clicks on signs and propagates this event to the system for the presentation of the list of interactions.

5.11 Java3D

The current version of Java3D, provided by Sun Microsystems [*H*] in Version 1.2, has many features which are necessary to implement a visualization system. This API is an application programming interface used for writing three-dimensional graphic applications and applets. The creation and manipulation of 3D geometry and rendering for very large worlds is the goal of this Java library. Java 3D has implementations in DirectX and in OpenGL, and was designed for high performance and a rich set of features. To run Java3D properly, install Java3D as extension to an existing Java Runtime Environment. If the application has to run as applet in the browser, the extension also has to be made to the Java Virtual Machine used by the browser.

typical characteristics:
- scene graph-based API
- high-resolution coordinates
- write once, run anywhere
- object-oriented programming
- internet capabilities
- browser plug-in

There are other visualization systems using Java3D: Walrus [4], Tron3D [49], Web-based 3D interface for product data management [24].

5.11.1 Rendering in 3D

immediate mode

retained mode

Compiled-retained mode

The Java3D rendering model provides three major modes: immediate mode, retained mode, and compiled-retained mode. The *immediate mode* allows maximum flexibility. The programmer directly influences the drawing without constructing a scene graph. Java3D can perform only localized optimizations. The *retained mode* provides a substantial increase in rendering speed, while all objects defined in the scene graph are accessible and manipulable. The changes in the scene graph are instantly visible. *Compiled-retained mode* allows Java3D to perform an arbitrarily complex series of optimizations and ensures high graphics rendering speed. The programmer can request that Java3D compile an object or a scene graph. Once it is compiled, the programmer has minimal access to the internal structure.

For our incremental approach it is necessary that changes in the scene graph are rendered immediately, the retained mode suits best. In the implementation we must have a '*sleeping*' function for the interruption of the visualization thread due to the cooperative multithreading in Java. In the mean time the Java3D threads can render the representation.

sleeping

5.11.2 Java3D Library

The Java3D Library is used for the construction of Java3D objects for data agents and visualizations. The signs we provide are spheres, boxes, cubes and tetrahedrons and several appearances (wired, solid lit, lighting, transparent, textured and combinations of that). For the use of textures we have to modify the image to be quadratic and the size to be a power of 2. Therefore we use Java Advanced Imaging to crop a part of the image meeting these demands. Very often a visualization needs lines for the drawing of associations. We integrated a function doing this by specifying the distorted position of two vertices. The labeling function has to create text objects with the brushing string of a data object. The generation function we integrated can also be used for the creation of text signs. Almost every visualization has to add lighting effects to the scene graph. To do it quickly there is a standard lighting available in the Java3D library. And also the keyboard navigation is suitable for most visualizations. It uses arrow keys and 'PageUp'/'PageDown' for zooming and moving the representation in the virtual environment.

5.11.3 JAI Image Reader

The Image Reader is used by the algorithm which calculates the modified images for textures. It can take most of the well known image formats and generate a BufferedImage. This image can be used for doing the manipulation.

5.11.4 Capturing

For the illustration of this work we had to take a lot of screenshots. There are two possibilities documented doing this task. The rendered image in the 3D canvas can be read into a raster of image point and stored as a JPEG-File. The resolution then is limited by the size of the canvas, which usually isn't good enough. The *offscreen rendering* is published in the Java3D specification for the re-rendering of a scene graph to an offscreen canvas. In the demos coming along with Java3D we found an example how to use this functionality and integrated this offscreen rendering. But there is a catch to it. The size of the picture is limited by the real screen resolution. It does not make sense to render the image to a bigger canvas because the part outside of the supported region will appear blank.

offscreen rendering

5.12 Summary

Our work focuses on the design of a framework for 3D visualizations for large information spaces. The limitation to a small portion of a huge information graph gives the possibility to use complex and well studied visualization algorithms. On the other hand, the framework has to be extendable and very general for the integration of any visualization. During the implementation we had to think about the modeling of an information graph, a general mechanism to fetch data for different data sources and to prepare it for a visualization algorithm.

The use of Java3D as three-dimensional library was a good decision because the implementation was straight forward and all of our ideas are realized. Additionally it's now very easy to add new data sources or visualizations as plug-ins and to do experiments with them.

The resulting graph drawings are aesthetical and assist the viewer in understanding the big picture and relationships among data objects. The focus+context method helps comprehension. Here we want to mention the level of detail brushing, the transparency used and the distortion functions. Combining these elements with the navigation functionality to rotate and zoom gives a very natural way for the exploration of an information graph.

Chapter 6

Conclusion

The information visualization framework realized in our work can be used for the exploration of data sources and visualizations. The combination of a three-dimensional visualization to several data sources shows the usability of the layouting algorithm in a data domain. On the other hand, we are able to compare different representations of the same data set.

6.1 Discussion

Our goal was to create a framework for the three-dimensional visualization of information spaces with open interfaces for the flexible extension. During this work we implemented several data agents and visualizations which can be loaded dynamically into our application for doing experiments and ergonomic studies. The features of our implementation are the use of neighborhood queries for data fetching, the incremental building of screen representations and the morphing between two logical frames. Apart from these realized functions, we offer a flexible and open information visualization framework for three-dimensional drawings. This generality cannot be found in any other work.

An interactive visualization aids comprehension by using the visual capabilities of people. The advantage of such a system is the gained speed for the collection of data and information. The user expects a preparation of data and relations combined with a nice looking representation. But the knowledge behind the prepared structures has to be generated by the user himself. Thus such a system only can be the interface to huge data collections.

The system lacks a little bit of performance during the fetching of data, because the querying is cost intensive. The main delay is data delivery and calculation in the data source (e.g. networking time, database or second storage access, searching, cpu time). This bottleneck in our system could be reduced with a caching mechanism such that the same query never has to be performed twice by a data agent.

Another bottleneck is the the memory consumption of our Java application. The construction and rendering of a large scene graph on the one hand, and the generation of data objects on the other hand are responsible for it. From time to time it likewise is the reason for crashes of the Java virtual machine. We tried to reduce the memory usage by the definition of class variables used as temporary object during the calculation in visualization components. The improvement in speed is remarkable.

Nevertheless we were surprised by the speed gained by our implementation in spite of the usage of transparency, textured shapes, animations and complex layouting functions. This is possible because of the hardware support in rendering of the drawings. Java3D uses DirectX or OpenGL and therefore it is quite fast on all operating system we tested. Especially for two reasons, the framework cannot be used in a productive environment till now: The delay switching from one logical frame to another takes too long and the the user has to know the data sources. It should be possible to implement a system which combines all available data sources by the definition of similarities among the entities. Thus we can hide the origin of an entity.

6.2 Future Work

This work is a good foundation for usability and user studies showing the acceptance of our incremental approach and the navigation issues. We can imagine more visualizations (e.g. TreeMap, Multi-Level algorithms, clustering algorithms) and data agents (e.g. email repository, discussion group, document management systems, music database) to be implemented. Likewise the integration of other transformations (minimal spanning tree, pathfinder network). A second line of thought is the integration of all data sources into one huge data collection with similarities among all entities. Then we can exchange the cost intensive neighborhood queries with better alternatives and the generality of this framework can be reduced by the elimination of transformations and the different types of information graphs.

The interaction with the application can be improved by smooth animations between two logical frames or higher parallelization in data fetching and the visualizations. This can reduce the delay of a frame switch. For most of the visualizations it is possible that more than one data object is placed at the same position in the drawing. These overlaps can be reduced by a displacement of some data objects. This function can be implemented independently from the visualizations as the distortions are.

Appendix A

Bibliography

A.1 Related Work

[1] J. Hollan B. Bederson. Pad++: A zooming graphical interface for exploring alternative interface physics. In *ACM UIST Proceedings*, pages 17–26. ACM Press, 1994.

[2] Franz-Josef Brandenburg, Michael Himsholt, and Christoph Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In *Graph Drawing*, pages 76–87, 1995.

[3] D. Brodbeck. Die Welt der Infoskope. In *GDI-IMPULS*, pages 52–59, 2001.

[4] CAIDA. Walrus - graph visualization tool. http://www.caida.org/tools/visualization/walrus/index.xml.

[5] Stuart K. Card, George G. Robertson, and William York. The webbook and the web forager: An information workspace for the world-wide web. In *CHI*, page 111, 1996.

[6] M.S.T. Carpendale, M. Tigges, and D.J. Cowperthwaite. Bringing the advantages of 3d distortion viewing into focus. http://citeseer.nj.nec.com/246949.html.

[7] Jeromy Carriere. Interacting with huge hierarchies: Beyond cone trees. http://citeseer.nj.nec.com/article/ere95interacting.html.

[8] John Cugini, Sharon Laskowski, and Marc Sebrechts. Design of 3-d visualization of search results: Evolution and evaluation. citeseer.nj.nec.com/323698.html.

[9] Ron Davidson and David Harel. Drawing graphics nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.

[10] P. Eades and M. Huang. Navigating clustered graphs using force-directed methods. In *Journal of Graph Algorithms and Applications*, volume 4, pages 157–181, 2000.

[11] Peter Eades. A heuristic for graph drawing. In D. S. Meek and G. H. J. van Rees, editors, *Proc. 13th Manitoba Conf. Numerical Mathematics and Computing*, Winnipeg, Canada, 29 –1 1983. Utilitas Mathematica Publishing.

[12] Peter Eades and Qing-Wen Feng. Drawing clustered graphs on an orthogonal grid. In *Proceedings of the 5th International Symposium on Graph Drawing, GD '97*, pages 146–157, 1997.

[13] Peter Eades and Roberto Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical Report CS-89-09, 1988.

[14] Christos Faloutsos and King-Ip Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, 22-25 1995.

[15] Arne K. Frick, H. Mehldau, and A. Ludwig. A fast adaptive layout algorithm for undirected graphs. In Roberto Tamassia and Ioannis G. Tollis, editors, *Proc. DIMACS Int. Work. Graph Drawing, GD*, pages 388–403, Berlin, Germany, 10-12 1994. Springer-Verlag.

[16] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[17] R. Tamassia G. Di Battista, P. Eades and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[18] P. Gajer, M. Goodrich, and S. Kobourov. A fast multi-dimensional algorithm for drawing large graphs.
http://citeseer.nj.nec.com/gajer00fast.html, 2000. Submitted to GD '2000.

[19] Stuart K. Card George G. Robertson and Jock D. Mackinlay. Information visualization using 3d interactive animation. In *Communications of the ACM*, pages 57–71, 1993.

[20] J.D. Mackinlay G.G. Robertson and S. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems*, pages 189–194, 1994.

[21] David Harel. On visual formalisms. In Janice Glasgow, N. Hari Narayanan, and B. Chandrasekaran, editors, *Diagrammatic Reasoning*, pages 235–271. The MIT Press, Cambridge, Massachusetts, 1995.

[22] Ivan Herman, Guy Melancon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[23] M.L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In *Proceedings of the 6th Symposium on Graph Drawing GD '98*, pages 374–383. Springer Verlag, 1998.

[24] Chunrong Yuan Jin Sun. Wed-based 3d interface for product data management. http://www.informatik.fernuni-hagen.de/import/pi5/veroeffentlichung/VIIP/paper.pdf.

[25] Ralph E. Johnson. Frameworks.
http://st-www.cs.uiuc.edu/users/johnson/frameworks.html.

[26] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[27] T. Keahey and E. Robertson. Techniques for non-linear magnification transformations. In *Proceedings to Information Visualization Symposium. IEEE '96*, 10 1996.

[28] S. Koruna. lecture notes in knowledge management. Swiss Federal Institute of Technologie, Zurich, 1999.

[29] A. Kumar and R. Fowler. A spring modeling algorithm to position nodes of an undirected graph in three dimensions. Technical report, Department of Computer Science, University of Texas - Pan American, 1997.

[30] Wei Lai and Peter Eades. A graph model which supports flexible layout function. Technical Report 96-15, Department of Mathematics and Computing, University of Southern Queensland, Australia and Department of Computer Science, University Newcastle, Australia, Callaghan 2308, Australia, 1996.
http://citeseer.nj.nec.com/189844.html.

[31] Ivan A. Lisitsyn and Victor N. Kasyanov. Higres - visualization system for clustered graphs and graph algorithms. In *Proceedings of the 7th International Symposium on Graph Drawing, GD '99*, pages 82–89, 1999.

[32] J. Mackinlay, G. Robertson, and S. Card. The perspective wall: Detail and context smoothly integrated. In *In Proceedings of CHI '91, New Orleans, LA*, pages 173–179, 1991.

[33] Charlie Gunn Mark Phillips. Visualizing hyperbolic space: Unusual uses of 4 x 4 matrices. Technical report, The National Science and Technology Research Center for Computation and Visualization of Geometric Structures (The Geometry Center), 1991.

[34] Bernd Meyer. Self-organizing graphs - a neural network perspective of graph layout. In *Proceedings of the 6th Symposium on Graph Drawing GD '98*, pages 246–262. Springer Verlag, 1998.

[35] P. Eades M.L. Huang and R.F. Cohen. Web OFDAV - navigating and visualizing the web on-line with animated context swapping. In *Proceedings of the 7th World Wide Web Conference*, pages 636–638. Elsevier Science, 1998.

[36] Tamara Munzner. Interactive visualization of large graphs and networks. Master's thesis, 2000.

[37] L. Nigay and F. Vernier. Design method of interaction techniques for large information spaces. In *Proceedings of AVI'98 (Aquila, May 1998)*. ACM Press, 1998.

[38] Diethelm Ironi Ostry. Some three-dimensional graph drawing algorithms. Master's thesis, 1996.
http://citeseer.nj.nec.com/ostry96some.html.

[39] Greg Parker, Glenn Franck, and Colin Ware. Visualization of large nested graphs in 3d: Navigation and interaction. *Journal of Visual Languages and Computing*, 9(3):299–317, 1998.

[40] N. Quinn and M. Breur. A force directed component placement procedure for printed circuit boards. *IEEE Trans. Circuits and Systems*, CAS-26(6):377–388, 1979.

[41] Tom Roxborough and Arunabha Sen. Graph clustering using multiway ratio cut. In *Proceedings of the 5th International Symposium on Graph Drawing, GD '97*, pages 291–296, 1997.

[42] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In Penny Bauersfeld, John Bennett, and Gene Lynch, editors, *Human Factors in Computing Systems, CHI'92 Conference Proceedings: Striking A Balance*, pages 83–91. ACM Press, Mai 1992.

[43] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Trans. Systems, Man and Cybernetics*, 21(4):876–892, 1991.

[44] Kenneth J. Supowit and Edward M. Reingold. The complexity of drawing trees nicely, 1983.

[45] Daniel Tunkelang. A Practical Approach to Drawing Undirected Graphs. Technical Report CMU-CS-94-161, Pittsburgh, PA, June 1994.

[46] Y. Zhang W. Lai, M. Huang and M. Toleman. Web graph displays by defining visible and invisible subsets. In *Proceedings of AusWeb99 - the Fifth Australian Web Conference*, pages 207–218, April 1999.

[47] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti Hearst. Animated exploration of graphs with radial layout.
http://citeseer.nj.nec.com/yee01animated.html.

[48] P. Young. Three dimensional information visualisation. Technical Report 12/96, 1996.

[49] Markus Zehnder. Tron3d - a game implemented in java3d.
http://www.markuszehnder.ch/projects/tron3d/tron3d.html.

A.2 Resources

A BiblioNet
http://beat.doebe.li/bibliothek

B BlackDown. Java3D for Linux
http://www.blackdown.org/java-linux/jdk1.2-status/java-3d-status.html

C Chariot - the Image Database of the Database Research Group
http://simluant.ethz.ch/Chariot/

D JDOM
http://www.jdom.org/

E search.ch - search engine
http://www.search.ch

F Sun Microsystems Inc. JAI
http://java.sun.com/products/java-media/jai/iio.html

G Sun Microsystems Inc. Java2 SDK 1.3
http://java.sun.com/j2ee/sdk1.3/index.html

H Sun Microsystems Inc. Java3D
http://java.sun.com/products/java-media/3D/index.html

Appendix B

Class Diagram

Appendix C

Task Description

Diplomarbeit: Framework zur Visualisierung von Informationsgraphen

---

Professor	Prof. Dr. H.-J. Schek
Student	Patrick Bichler
Assistent	Kai Jauslin
Beginn	29. Oktober 2001
Abgabe	28. Februar 2002

Motivation, Hintergrund

Die Menge elektronisch verarbeiteter Informationen nimmt stetig zu, die grundlegenden Methoden zu deren Verwaltung aber haben sich in den letzten Jahren nicht gross geändert. Einzelne Programme dienen der Bearbeitung und Verwaltung spezifischer Informationstypen und -mengen. Der Austausch und die Verknüpfung von Informationen ist dadurch stark erschwert.

Das Ziel des *All4U – Dynamic Collaborative Information Spaces* Projektes, welches im Rahmen von ETH World durchgeführt wird, ist die Entwicklung einer integrierenden Software zur Visualisierung, Organisation, Navigation und Kommunikation von Informationen. Dabei bleibt die eigentliche Bearbeitung den bestehenden Programmen überlassen. Anstatt auf fix vorgegebene Informationsstrukturen kann der Benutzer eine Informations–Basismenge dynamisch visualisieren. Diese dynamische Visualisierung berücksichtigt z.B. Aspekte wie die Ähnlichkeit von Informationen.

Aufgabenstellung

Die Beziehungen zwischen Informationen können mittels eines Graphen modelliert werden. Die in dieser Diplomarbeit zu bearbeitenden Fragen gliedern sich in die

untenstehend aufgeführten Problembereiche. Die aus der Theorie gewonnenen Erkenntnisse sollen in ein Framework umgesetzt werden, wobei der Schwerpunkt der Diplomarbeit auf der praktischen Implementierung der Visualisierungsmethoden gesetzt ist.

- Modellierung von Informationsgraphen
  Wie lassen sich Einheiten von Information, deren Struktur und Beziehungen untereinander (*Meta–Information*) allgemein beschreiben, so dass sie später auf einfache Art visualisiert werden kann? Wie kann diese Beschreibung sinnvoll an eine Frontend–Software übermittelt werden? Welche Probleme entstehen dabei für sehr grossen Graphen (z.B. Linkstruktur im Web, Citation Mapping)?

- Visualisierung
  Für die Visualisierung *hierarchischer* Graphen (Bäume) existieren in der Literatur eine Reihe von interessanten Methoden, wie z.B. Hyperbolic Trees 2D/3D, Cone Trees. Wie geeignet sind diese für die Darstellung von Informationsgraphen? Wo liegen die Probleme (auch in Bezug auf Performance)? Was für Methoden existieren für allgemeine Graphen?

- Interaktion und Navigation
  Die Visualisierung dient dem Benutzer als Navigations–Frontend. Wie kann das System auch bei grossen Graphen effizient auf Benutzeranfragen reagieren (z.B. differentielle Übertragung)?

Aus diesen Problemstellungen sollen u.a. folgende Resultate erarbeitet werden:

- Protokoll zur dynamischen Kommunikation von Informationsgraphen
  Schnittstelle zum Visualisierungs–Frontend. Wie und was wird von der Visualisierung benötigt, in welcher Form könnte die Übertragung stattfinden (z.B. Client/Server, Streaming). Im Speziellen soll der Fall betrachtet werden, wo Thumbnails als Meta–Daten (Grösse abhängig vom Zoomfaktor der Visualisierung) über das Netz transportiert werden müssen.

- Framework zur Visualisierung von Informationsgraphen
  Die theoretischen Erkenntnisse sollen in einem prototypischen Framework umgesetzt werden. Dieses soll möglichst offen und flexibel für neue Visualisierungsarten sein. Performance ist auf dieser Stufe erwünscht, aber nicht Schwerpunkt.

- Implementierung verschiedener Visualisierungen
  Als Beispiel für das erarbeitete Konzept sollen im Framework zwei bis drei Visualisierungsmethoden implementiert werden.

- Einfache Navigation (Zoom & Switch)
  Der Benutzer soll die Möglichkeit haben, in die gegebene Visualisierung hineinzuzoomen und im Graph zu navigieren (Kontextwechsel).

Ziel und Schwerpunkt der Arbeit ist die exemplarische Verwirklichung der Konzepte zur Visualisierung an einem Prototyp–Framework. Die Aspekte Performance und Navigation sind in dieser Arbeit nicht zentral. Die Verfügbarkeit von Informationsgraphen und benötigten (d.h. in dieser Arbeit definierten) Meta–Informationen soll vorausgesetzt werden.