U N I V E R S I T Y   O F   T A R T U

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Institute of Computer Science

Computer Science speciality

Dmitri Danilov

3D Graph Exploration

Master Thesis (20 cp)

Supervisor: Ulrich Norbisrath, PhD

Author: .... “.....” May 2010

Supervisor: .... “.....” May 2010

Allowed to defence

Professor: : .... “.....” May 2010

TARTU 2010

Contents

Acknowledgments

I would like to express my sincere gratitude to the people who helped me in the process of writing this thesis.

At first I would like to thank my supervisor Ulrich Norbisrath. His guidance and novelty changed a lot in my way of thinking. He is one of the few, who supported me on this difficult way.

I would like to thank my family, my mom Natalia and dad Nikolaj. They always believed in me and supported me in the hard times. Without their support this thesis would not see the light.

I would like to thank my friends, who encouraged me, when the complexity of the problems made me sad.

And many many thanks I want to say to my special person - Julia, who always helped and inspired me. Without her support this thesis would not be completed.

Many thanks to all of you!

Introduction

People are explorers by nature. According to Maslow's hierarchy of needs information seeking is a fundamental human activity. However, the information overload and its duplication become a significant problem in society. Efficient search tools become more and more valuable. This thesis introduces a set of ideas and implementation solutions for data visualization in relation to the exploratory search strategy.

The predominant retrieval paradigm of search systems today is "query and response", where queries are issued by the user and a set of potentially relevant items are offered in response. This approach is usually insufficient for complex cases like vacation planning or research of some area. The inability of search engines to give an adequate response to a complex problem motivates researchers to develop new information seeking techniques. One of the most promising is exploratory search [29].

The exploratory search technique is one of the most suitable searching strategies if a user is unsure about the ways to achieve his goals or even unsure about his goals in the first place. The goals of exploratory search extend beyond simply locating information toward activities associated with learning and understanding [29]. Thus, the exploratory search technique may be used, when a user is not familiar with the domain of his goals. A set of ideas provided by the exploratory search technique allows to reduce search time, introduce learning and investigation into the search process and make information finding more efficient.

Some of the key elements of exploratory search systems are interface and information visualization. The information in common usually represents a complex network of objects and their relations. These are tables and a set of keys in relational databases, for example research papers and references or websites and hyperlinks. The classic way to visualize relations between objects is drawing graph. A graph view may also be used for a data exploration process. There are three main classes of elements in the graph: nodes, edges and their attributes. This simple way of data visualization and strong graph theory base can provide all necessary tools for a complex and structurized network representation.

Exploratory search information visualization focuses on the visual representation of large data collections to help people understand and analyze the data. In graph theory this property can be successfully implemented by graph clustering (see chapters 1 and 2). Graphs can also be used to reduce data duplication. Data may be divided into the pieces and each piece of information would be unique within the graph. Different types of layouts structurize the graph and give additional information about data hierarchy. According to the list of exploratory search system features (see chapter 4 in [29]) the software must offer visualizations to support insight and decision making: Systems must present customizable visual representations of the collection being explored to support hypothesis generation and trend spotting. Drawing graphs is an efficient technique to explore the complex object relations and allows to generate hypotheses based on the explored relations.

There is a set of effective methods to visualize and structurize graph in two-dimensional space (2D). However, a human has a native ability to understand representation of objects in three-dimensional space (3D). The 3D graph visualization is relatively new field of knowledge and offers many benefits. Utilizing one extra dimension gives a possibility to go from plane to the space, from schematic representation to the visualization of complex systems, where the major roles are played by navigation techniques, graph structure, and interface solutions. The 3D representation allows to reduce the space of the graph at the expense of a user's imagination ability and makes navigation more dynamic. Thanks to modern 3D engines, the visualization of a graph can be photorealistic and provide desired dynamism. The user interface can include models of known physical objects that will make it intuitively understandable. The navigation in 3D in common is more flexible and allows a user to change the viewpoint and focus on a desired group of data. Unfortunately, representation of 3D space today is limited to the display technology.

Until 3D displays or holography technologies are not present in every particular home, people still need to transform 2D representation to 3D in the mind. The 3D engines generate 2D pictures (frames) for each moment of the scene transformation in a process called rendering and this is the closest way to emulate 3D space on a 2D display. Actually, computers generate the stream of 2D visual data that is by-turn transformed in the user's brain by the native human ability to the 3D scene. Depth perception is the visual ability to perceive the world in three dimensions. In this case a user does not receive the information about all object distances (scene depth), because there is only one viewpoint for both eyes. When this type of adaptation is used (one viewpoint), users are limited in their native cognition abilities.

Likely, there are techniques to partially overcome this problem. A user needs to have more visual data that is coming from efficient dynamic navigation and knowledge of graph structure (see layout descriptions in chapter 1). The dynamic interface that allows easy and frequent changes of the camera viewpoint will give more information about the object transformation in space that will by-turn give information about object distances (see section 2.8). The graph visualization software can provide a clear and understandable layout structure to help the user to orientate in space (see section 2.5). If the 3D graph layout is intuitively understandable and gives sufficient information about the graph's hierarchical structure the user can navigate it more effectively. The emphasis on the clear rigid structure simplifies cognition and requires less processing in the user's mind. The environment of the graph exploration can also help the user to orientate (see section 2.4).

This thesis will introduce several solutions for the described 3D graph visualization problems in context of exploratory search. The first chapter makes a small overview of the most noticeable graph visualization software at the time of writing and analyze software graph exploration properties (1). It also review the problems related to the 3D graph exploration in detail. The second chapter describes the implementation of the 3D graph explorer and problem solutions (2). The third chapter is a small guide for using and extending the developed 3D graph explorer (3). Finally, the fourth chapter introduces a new features and ideas that can be implemented to improve the developed 3D graph explorer and fulfill some of the exploratory search strategy aspects (4).

Chapter 1

3D Graph Exploration

Data exploration with a 3D graph offers many benefits. First of all, the additional dimension provides an extra space for the graph layout. This is an advantage for representation of a larger amount of structured data. At the same time it allows to achieve the exploratory search visualization goal in representing larger data collections. The graph representation in 3D can introduce understandable and also more complex visualization of the structured data with a less space on the 2D display. The layout can show the hierarchy of the data relations and target the valuable data centers. These essential elements allow the user to search for important data nodes and generate hypotheses. The data structured with hierarchical layout helps the user orientate and navigate the graph. The graph could be partitioned to the clusters and give necessary level of details on demand.

A graph-based 3D visualization of complex data does not require additional cognition skills. However, the 2D display does not provide enough information about the scene depth. A user gains this information from the scene transformations during the navigation process and cannot estimate the scene depth if the viewpoint is static. The solution of this problem is always a challenge. The developer must keep a balance between layout complexity and navigation solutions to help the user orientate in the space. It is easier for a user to orientate in a less complex layout. However, the layout must give more information about the data hierarchy to support hypotheses generation. It is easier for a user to understand the graph depth from continuous graph transformations (like in Interactorium in section 1.1.4 with force-directed layout in section 1.2.1), but in this case the user loses the impression of structurized data and cannot orientate.

The human ability to hold the structure in mind varies a lot from user to user. Thus, the graph layout complexity must be customizable. The graph can be partitioned into clusters and provide the necessary level of details. Moreover, graph partitioning must be dynamic and change cluster size on user's need globally or locally. In analogy with exploring the file system in modern operation systems it must be possible to explode and implode the clusters to choose the level of details on demand (See section 2.5). It can be achieved by graph partitioning techniques.

The most common problem for graph visualization is its layout development. In case of graph exploration the layout must provide the data hierarchical structure and in the same time it must have less edge crossings. The structure of the graph must provide the base for the user's orientation.

There is not many 3D graph visualization software today. One of the reasons is the fact that computer graphics technology is one of the most dynamic fields in computer science and the possibilities of 3D visualization today differ dramatically from the possibilities a few years ago. The increasing availability of powerful graphics hardware assists the development of 3D graphics technology. The new approaches in 3D engine architecture significantly speed up the development process. Today, it is possible to create a 3D application using a high level programming language that would not lose much in performance compared to the corresponding low level application.

This chapter will review several 3D graph visualization software and describes the software advantages and disadvantages in context of graph data exploration. This chapter also includes the graph structure (or layout) solution descriptions of the reviewed software.

1.1 The 3D Graph Visualization Software

In the World Wide Web a several 3D graph visualization applications can be found. However, they are designed with a focus on the visualization of the graph structure. The data exploration in most of the software is problematic or very limited. This section gives a brief overview of the most noticeable 3D graph visualization tools at the moment of writing and review the software functionality in context of exploratory search strategy. The aim of this section is to prove the need of development of new 3D graph visualization software that will fulfill the exploratory search visualization strategy goals.

1.1.1 CAIDA Walrus

CAIDA (The Cooperative Association for Internet Data Analysis) Walrus is an open-source tool for interactively visualizing large directed graphs in three-dimensional space [3] (Figure 1.1). It was developed by Young Hyun at CAIDA and it is based on Java3D technology. As opposed to other software, Walrus has an advantage in handling very large graphs. The main idea is to use the hyperbolic space, in other words, to display graphs under a fisheye-like distortion. User can imagine a magnifying glass moving around the graph. Graphs are rendered inside a sphere that contains the Euclidean projection of 3D hyperbolic space (See 1.2.2 on page 17 for details). Unfortunately, the layout is effective only for moderate sized graphs or graphs that are nearly trees. Otherwise, the degree of node's connectivity increases and user loses the impression of structurized data.

The rendering session of the graph with CAIDA Walrus must be started manually by the user. Before rendering session is started the *.graph file with saved graph data must be opened. Unfortunately, only LibSea graph files (documented CAIDA-developed input format) are supported. The user can access all the tool features from the main menu on the top of the screen. There are navigation features like “Show Root Node”, “Show Parent”, but mostly the navigation is done by mouse right clicks on the graph nodes. The graph is positioned inside the unit ball of Klein hyperbolic model (See section 1.2.2 on page 17) and could be rotated around selected node (that would be always in the center of the unit ball) by holding the left mouse button and moving the mouse. This approach would suit for people, who have problems with orientation in space. The analogy for this graph navigation technique would be the rotation of some physical object in front of the observer. As edges and nodes Walrus uses simple geometry - lines and small squares as nodes. One noticeable disadvantage of the Walrus interface is absence of the zoom with mouse scrolling. Overall, the interface is user-friendly, however the import of custom graphs would require additional work with CAIDA-specific file format. The navigation is limited to zooming and rotating the graph around selected nodes.

One of the main disadvantages of this tool is the way it handles labels (attributes). User can only access it by pressing the middle button on the node. There is no opportunity to display them for each node (or a group of nodes) automatically during the exploration process. Thus, this tool is designed with a focus on the graph (tree) structure exploration and it is less suitable for data exploration. The tool uses specific *.graph input file format. The latest version is 0.6.3, released on Mar 30, 2005.

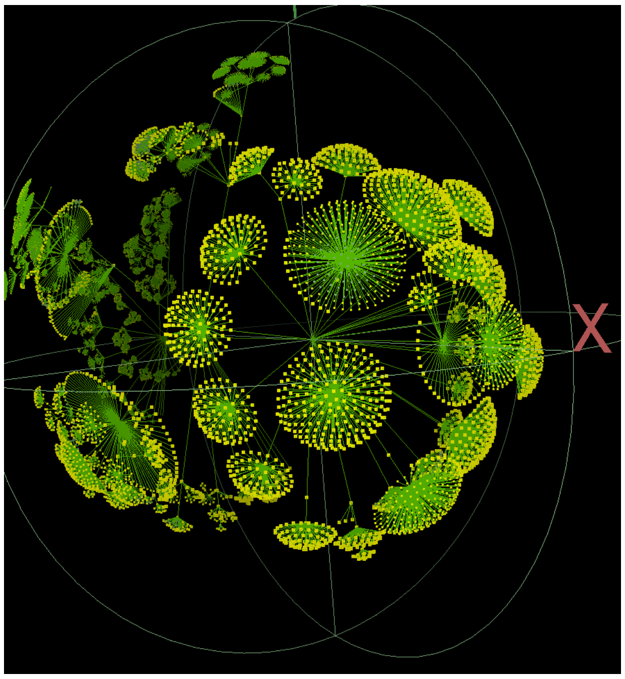


Figure 1.1: CAIDA Walrus 3D graph visualization tool

HypViewer HypViewer is an open-source simple tool developed by Tamara Munzner[21] for exploring 3D graph (Figure 1.2). This tool is based on the H3Viewer library [8, 9] created by the same author. According to the CAIDA Walrus website [3], Walrus tool is based on the Tamara Munzner's research and the idea of 3D graph representation in HypViewer is the same as in Walrus tool.

The HypViewer tool has a minimalistic user interface allowing to observe the graph with the mouse clicks. The rendering starts immediately after loading HypViewer specific file (*.lvhist [9]) with graph data. As opposed to Walrus the additional advantage of the HypViewer is possibility to smoothly change the center of the unit ball over the graph. Another advantage is possibility to control the amount of labels being visible near the center. The disadvantages in the context of graph exploring are poor interface and inability to explore the data in the nodes or edges. Overall, this tool is more a prototype than a visualization package of a full value.

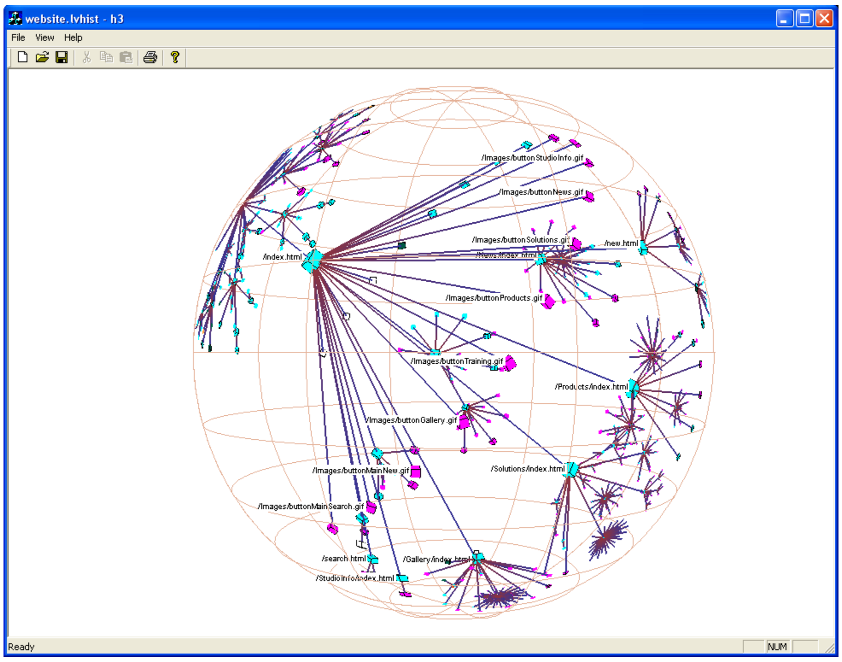


Figure 1.2: HypViewer by Tamara Munzner[21]

1.1.2 Nodes3D

Nodes3D is an open-source 3D graph visualization program written by Issac Trotts in the labs of Edward G. Jones [2] (Figure 1.3). This tool provides more native view of the graph in the space. The layout of the nodes is automatic and similar to a star. User can zoom and rotate the graph around the common center of the graph. There is also possibility to travel from node to node. This is a nice tool for viewing small graphs, but it is not effective with larger graphs (Figure 1.4). There is also no possibility to set the random rotation center for the graph. Only initial center and node positions could be set as graph rotation point. The navigation in the graph is limited to zooming, graph rotation (around its imaginary center) and automatic camera position translation to the selected node. The program uses *.lua[18] script file as input.

The graph rendering in Nodes3D starts immediately when the *.lua script file is loaded to the program. The list of allowed commands is documented in the Nodes3D package. The interface of the tool is simple window with loaded graph. Some of representation settings could be accessed from the mouse context menu (right click), but most of them are assigned to the keyboard specific buttons. A list of this buttons could be found in documentation. The automatic camera translation to the selected node in Nodes3D redefines the rotation center to the selected node. It is not possible to redefine the graph rotation center to the random place. The windows version of the tool is not stable with all example graph files provided with the package, but interface is user-friendly with minimalistic approach. The navigation is not a strong side of this tool. The graph exploration is possible, but it is very limited.

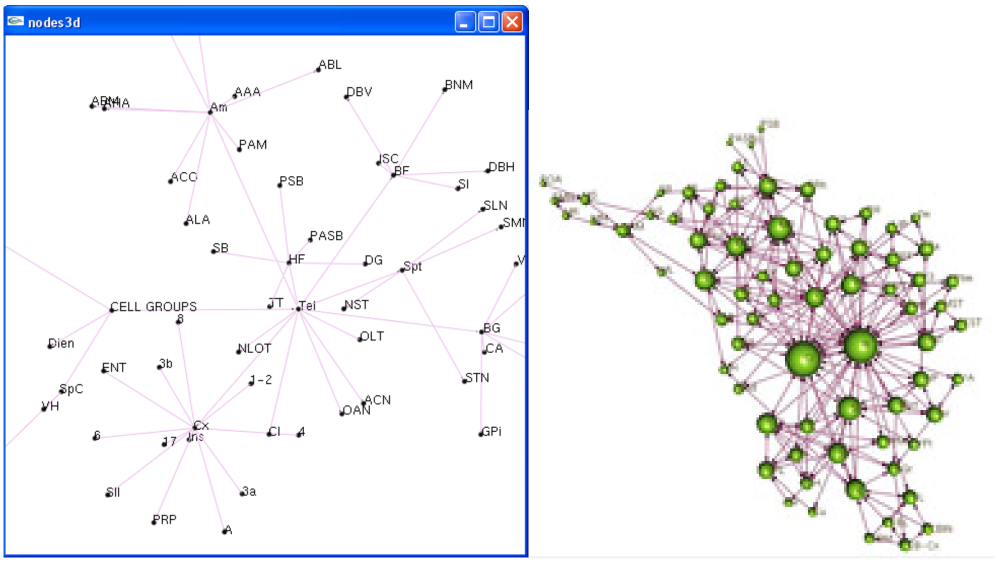


Figure 1.3: Nodes3D by Issac Trotts

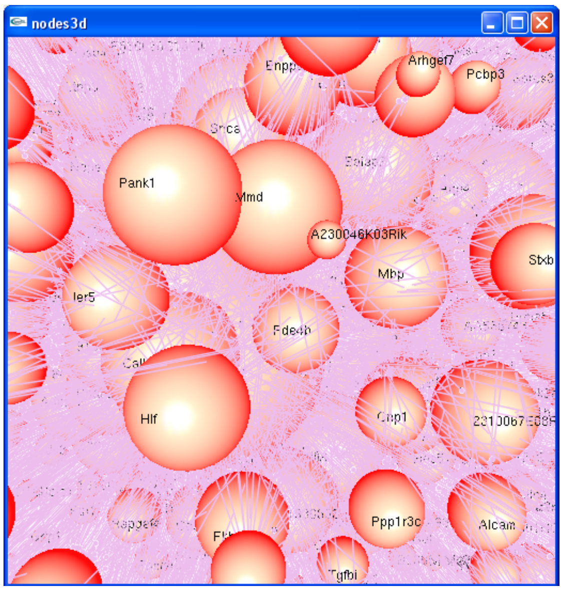


Figure 1.4: Nodes3D with large graph

1.1.3 WilmaScope

WilmaScope is a Java3D open-source application which creates real time 3D animations of dynamic graph structures[23, 22] (Figure 1.5). WilmaScope has an automatic installer for Windows and integrates into the OS. Wilma was originally created by Tim Dwyer, with valuable contributions from Peter Eckersley and James Cowling[23].

Wilmascope graph visualization tool is one of the most rich solutions for the graph representation, analysis and modification. The interface offer many menus and settings. It is possible to adjust forces in force-directed layout (see 1.2.1), modify the graph deleting or adding new nodes, attributes, clusters. The most important feature of this application is user-friendly interface for graph modifications. Wilmascope allows 3D graph modification on the fly. The tool provides a few graph analysis features: "Degree centrality" and "Biconnected components" with options of an action that must be performed for the graph (according to the analysis results) like adding labels, changing node size, changing node repulsion and others.

The application uses force-directed layout (See section 1.2.1 on page 16). Wilmascope also provides additional functionality that simplifies its usage. Advantages of the application are a lot of features like graph auto generator and a panel, where the user can adjust the forces of the layout. Disadvantages of this application are inability to change the graph's rotation center and light instability. Sometimes the application crashes. The disadvantages of the navigation are missing possibility to redefine the graph rotation center and no automatic camera positioning to the selected node/edge. This application is comfortable for graph modifications and observing small or medium size graphs (also a set of not connected graphs). However, it is not comfortable for graph exploration process. It is not considered to access to the more complex data inside the nodes.

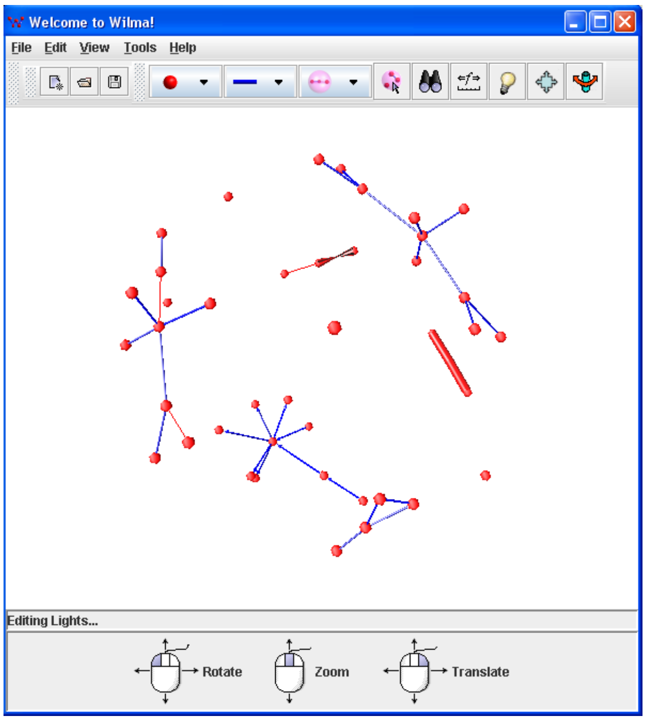


Figure 1.5: Wilmascope

GEOMI The GEOMI (GEOmetry for Maximum Insight) is a visual analysis tool for the visualization of different network types like web-graphs, biological networks and social networks. GEOMI is being developed by VALACON (Visualization and Analysis of Large and Complex Networks) project team members in the National ICT Australia (NICTA) IMAGEN program[25].

The GEOMI interface differs from old style Java based interface of WilmaScope. However, the navigation and functionality are mostly the same. The GEOMI provides many additional features in graph analysis like "Closeness Vitality", "Bonachis Standard Centrality", "Counting Spanning Trees" and additional graph layouts like "Rod Tree", "Clustered Circular", "Planar Cluster Force", "Spectral Layout", "Hierarchical Layout" and others.

The GEOMI is based on the WilmaScope (See section 1.1.3 on the previous page) graph visualization library, but provides many additional features for network analysis, graph layout as well as interaction methods [25]. As an advantage of this implementation a large amount of new layouts can be named. However, this implementation have the same problems with stability as the WilmaScope has and do not introduce new techniques for graph navigation.

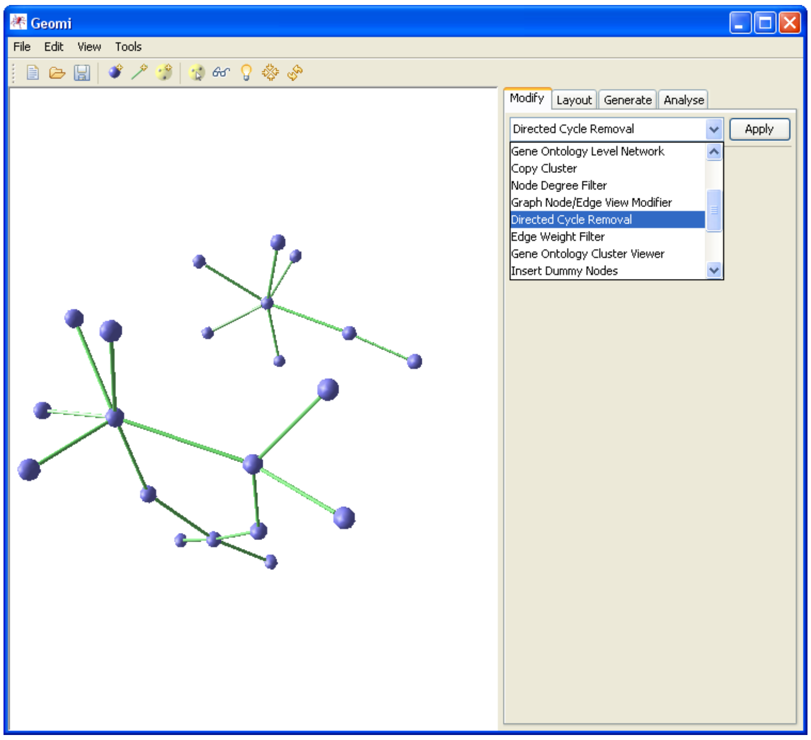


Figure 1.6: GEOMI

1.1.4 SkyRails/Interactorium

The Interactorium is a platform built to visualize very large interactome datasets [10]. Developed in collaboration with the School of Computer Science and Engineering at the University of New South Wales, it was adapted from the SkyRails visualization engine, which was originally developed by Yose Widjaja. Interactorium application functions as an atlas of known protein-protein interactions [10] (Figure 1.7). The application has featured interface and provides intuitively understandable navigation techniques. Application also gives the possibility to navigate by selecting nodes and edges. The Interactorium visualizes the cell from 3 different levels: from the cell, to protein complexes and interactions, and into protein structure [24].

The Interactorium is one of the most interesting visualization tools today. Many shaders and visual interface solutions make it interactive and visually rich. As opposed to other software in this area, the main fundamental difference of Interactorium is a “first person” navigation. User can imagine the flight in the space from node to node exploring the data and its relations. Most of the navigation is performed by the mouse clicks. When user clicks on a node with a left mouse button the camera fly to the node and the navigation mode changes. If some node is selected then camera starts to rotate around the node showing its surroundings. In order to return to the previous navigation mode user must select “free view” from mouse context menu (Figure 1.7).

Interactorium also provides an additional navigation technique by node attribute filtering. User can dynamically type the name of the target node and Interactorium would suggest existing node names during the typing. Once the node is defined, user would immideatealy fly to the appropriate node. This feature could be considered as one of the exploratory search stratagy elements. Systems must help users formulate queries and adjust queries and views on search results in real time (see chapter 4 in [29]).

One of the important advantages of Interactorium is a console and support for scripting. In the console user can run commands using Skeilein/Roenskripp. Interactorium can integrates a 3D model of protein structure inside the node model circle (Figure 1.8). This demonstrates the power of the method of exploring the graph in 3D, where the data of the node could be represented as any 3D model giving more analogy with known physical objects.

The interface of the Interactorium is dynamic and intelligent. The forces of the layout are customizable. The interface is user-friendly, but the documentation is insufficient. Interactorium provides one of the best navigation experiences. As opposed to Interactorium, the SkyRails [30] could be used for other graph thematics. Unfortunately, it has poor documentation and regular user can't get advantage of this system.

The both systems use force-directed layouts that evolve dynamically during the exploration session. It can be considered as the main disadvantage of this application because continuous change of the data structure makes impossible to orientate in the graph and the probability of revisiting of already explored elements grows significantly.

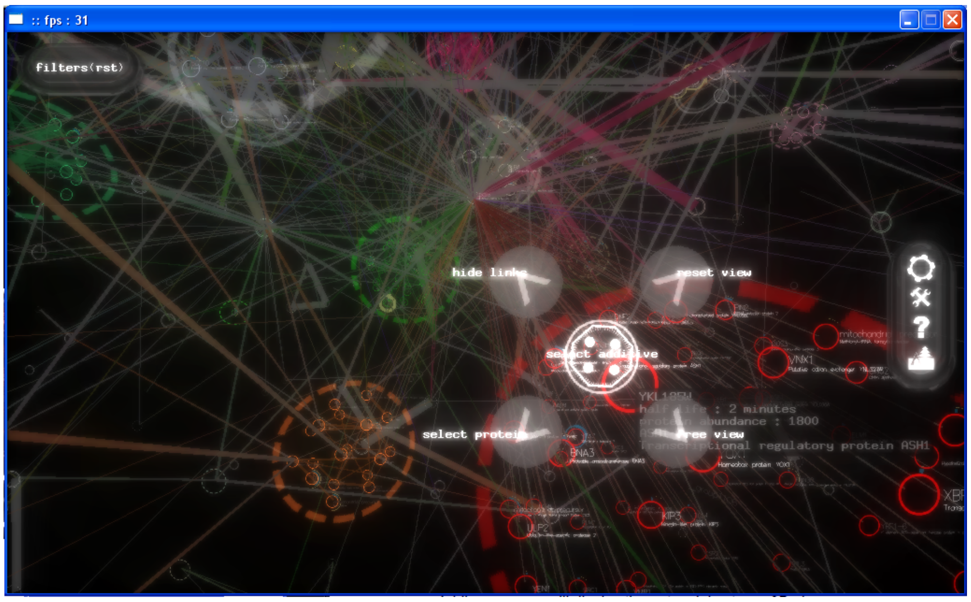


Figure 1.7: Interactorium. The navigation.

1.1.5 Redfish Solutions

The force-directed graph layout project demo can be found on RedfishGroup website[19]. The project demo is the simulation of a closed physical system that self-organizes in a way similar to the formation of crystals[20] (Figure 1.9). The project idea of using the rules of different type of crystal formation could give new approaches to the 3D graph

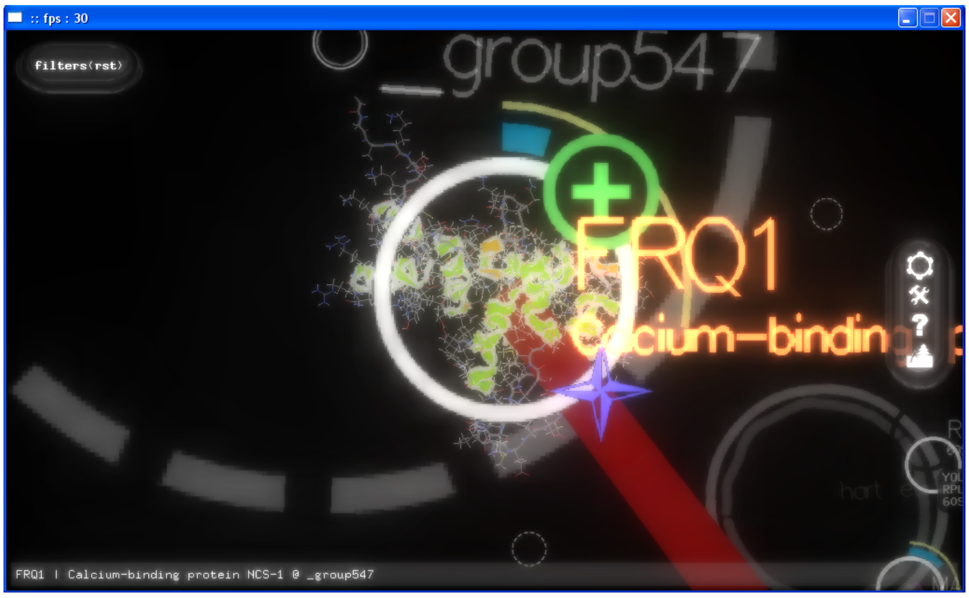


Figure 1.8: Interactorium. The 3D object inside the node.

layouts. The online demo provides a possibility to load the graph from the file using the URL. The file format is documented. The tool is rather a prototype for the layout idea representation. Navigation is limited to zooming and graph rotation around imaginary graph center point. Unfortunately, there is not much information about this project.

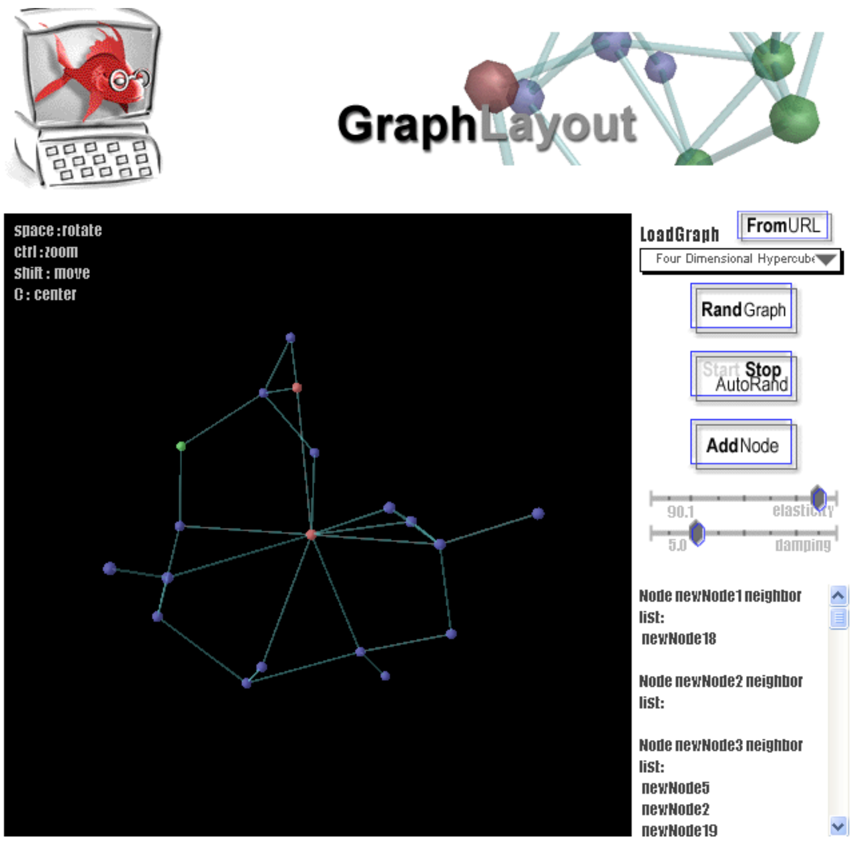


Figure 1.9: Redfish 3D layout demo

1.1.6 Data exploration property

The most suitable visualization tool for the graph data exploration is an Interactorium platform. The navigation interface is intuitive and effective. However, the slowly evolving force-directed layout makes the orientation problematic. Poor documentation of an application limits usable features. All other tools are considered more for observing the graph structure and do not focus on the navigation among the graph data. The most interesting for the handling the larger graphs is the CAIDA Walrus. The hyperbolic approach allows to visualize the structure of the graph in an efficient way with local details and global context simultaneously. Nevertheless, it is confusing for an average user and do not give much help for data exploration tasks. The WilmaScope, GEOMI and Nodes3D are suitable for observation of smaller graphs, but the navigation part is poor and can't be used for graph data exploration.

None of the software can handle the graph data exploration tasks. On the other hand, some interface ideas can be useful for data exploration systems. The idea to use different navigation modes (like in Interactorium) can make the data exploration process more felxible and efficient. The different graph theory algorithms that modify the selected properties of nodes or edges (size, color, etc) can improve the layout and help to find the desired data (implemented in WilmaScope and GEOMI).

All the software uses a low level programming languages. In order to extend the software developer needs additional skills in programming computer graphics. On the other hand there are 3D engines that offer support of high level programming languages. These 3D engines can provide all necessary tools for developing the dynamic interface and 3D graph drawings with less time investment.

1.2 3D Graph Layouts

The term "layout" in graph visualization represents the method of the arrangement of the graph nodes in the space (or on the plane for 2D). The most important features of layout are clear structure and efficient utilization of space. It is always a challenge to keep the balance between intuitively understandable structure and efficient utilization of third dimension in 3D graph layouts. This section will give a short overview of different layouts used by 3D graph visualization tools in previous section.

1.2.1 Force-directed layout

Also known as force-based or force-directed algorithms for drawing graphs. The aim of the algorithms is to provide a graph layout, where graph would have approximately equal edge lengths and a small amount of edge crossings. These aims are achieved by assigning forces among the set of edges and the set of nodes. There are a lot of implementations with known laws of physics like using Hooke's law of elasticity for edges and Coulomb's (electrostatic interaction) law for nodes. The forces applied to the graph elements pull them closer together or pushing them further apart (Figure 1.10). The algorithm iteratively calculates the forces and changes the layout each step until equilibrium state. At the moment of equilibrium the graph is drawn.

One of the main advantages of this approach is its flexibility. Force-directed algorithms can be easily adapted and extended with additional forces or coefficients to reach the criterias. Usually these algorithms are intuitive since it is easy to find physical analogies in the outside world. Force-directed algorithms can provide an interactivity allowing user to see the layout evolving in time.

The main disadvantage of this layout is high running time. Force-directed algorithms usually have at least $O(V^3)$ time complexity (per step), where V is the number of the nodes in the graph. Another problem that can be named here is poor local minima problem. In short, the force-directed algorithms calculate the graph with minimal energy. The local equilibrium (found local minimum) state achieved by the algorithm is not always the global minimum. This makes it less reliable.

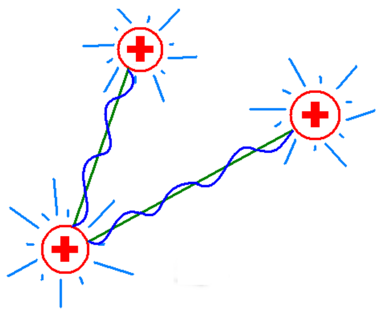


Figure 1.10: Force-directed layout approach

1.2.2 Hyperbolic space approach

The aim of hyperbolic space approach is to provide the graph display with local details and the global context. User can imagine the magnifying glass that zoom in and show details in the center and zoom out near the boundaries. There are several models of surfaces in which the parallel postulate fails. Two of them are used by 3D graph visualization programs discussed in the first chapter.

The Klein model of the hyperbolic space is used in CAIDA Walrus graph visualization tool. This model of n-dimensional hyperbolic geometry corresponds to the n-dimensional unit ball (disk in case of two dimensions). The "points" in that model are points in the interior of the unit ball. The "lines" in the model are represented by the chords (straight line segment with endpoints on the boundary sphere). The "distance" between two points A(x,y) and B(u,v) (Euclidean coordinates) in the simple Klein model of hyperbolic plane is $arccosh(\frac{(1-xu-yv)}{\sqrt{(1-x^2-y^2)(1-u^2-v^2)}})$ (Figure 1.11 of regular pentagon in Klein's model of hyperbolic plane).

According to the notes of Young Hyun the HypViewer developed by Tamara Munzner uses the Poincare model. However, in Tamara's work [26] she describe the layout as "second-generation 3D hyperbolic cone tree" and is stating that application uses the Klein model in her work. To understand the difference between Klein and Poincare models, some details about Poincare model are provided below. The Poincare model corresponds to an n-dimensional unit ball like the Klein model do. But, as opposed to Klein model, the "lines" in Poincare model are represented as arc of a circle, whose ends are perpendicular to the disk's boundary, where diameters are also permitted (Figure 1.11 of regular pentagon in Poincare's model of hyperbolic plane).

Thanks to the described peculiarities, hyperbolic models in Euclidean space provide an important property of geometry magnification in the center of the model and reduction on its boundaries. This property allows to see the details in the center of the models and the global context on the unit ball boundaries (Figure 1.12).

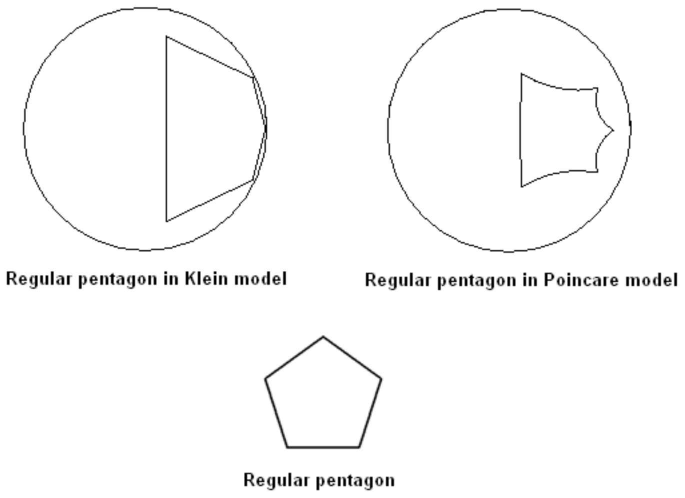

Regular pentagon in Klein model
Regular pentagon in Poincare model
Regular pentagon

Figure 1.11: Regular pentagon plane in hyperbolic models

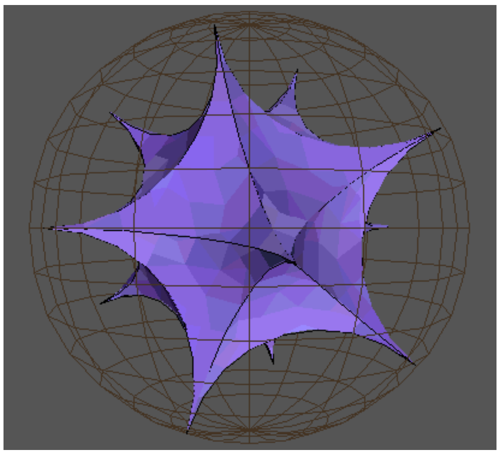


Figure 1.12: Poincare hyperbolic model in 3D[13]

1.2.3 The 2.5D layout

In computer graphics the 2.5D (also known as pseudo-3D and "two-and-a-half-dimensional") is a set of techniques, that use a series of 2D scenes or images to emulate the 3D environment. The 2.5D approach for graph drawings (graph layouts) represents a series of planar subgraph layouts organized in 3D space (Figure 1.13). The subgraphs on the planes are usually organized according to the hierarchy of relations. This approach could also be used to represent the graph evolution in time, where each plane is one of the graph states and the connections between nodes on the different planes represent equal nodes. One of the main advantages of this layout is clear hierarchical structure. This is one of the best approaches, when the hierarchy of the graph is important.

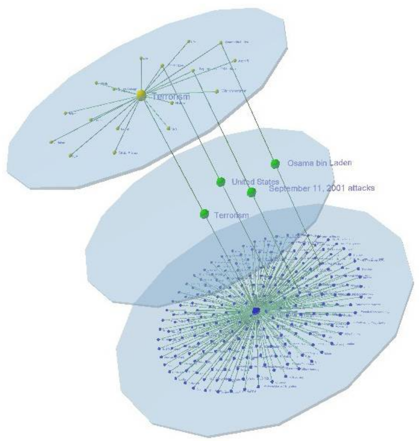


Figure 1.13: 2.5D graph layout[27]

Chapter 2

Implementation

The implementation represents a 3D application that visualizes the directed graph with a 2.5D layout and provides the possibility to explore the graph data. The main motivator for this application development is the idea to use the graph for complex data representation and apply the graph theory and 3D engine features to fulfill the goals of exploratory search visualization strategy. The reasonable and hierarchical layout for data relations in 3D space can represent more structured data on a 2D display and support hypotheses generation based on the explored relations. The 3D engine can provide with the tools to realistically visualize the graph and make navigation more dynamic and intuitive. This chapter describes an important parts of the application implementation using the Panda3D engine (see section 2.1).

Data of the graph is parsed from a properly formated CSV file (see section 3.3). This is a test extension of the developed 3D graph visualization logic (see section 2.2 and 3.2 for structure and extension description). There is also one practical extension made for the Email Information Concentrator (see section 2.9). Email Information Concentrator aims to deliver emails via IMAP and parse the relevant parts of messages to other formats, such as an Email Graph or Prolog statements.

This chapter is divided into nine sections. Each of them represents an important part or aspect of the developed application. The first section describes the 3D engine used by the application. The second section makes an overview of the developed application structure and describes application main classes. The third section gives an overview of an important engine feature called "scene graph" and describes its implementation in the developed application. The fourth section describes the techniques of the environment emulation. The fifth section gives an explanation of the graph layout solution. The sixth section describes the techniques associated with a camera object. The seventh section explains the engine feature called "Task manager" and describe its implementation in the application. The penultimate section describes the user interface, navigation solutions and different features. Finally, the last section describes the practical extension of the application for another project.

2.1 The application engine

The base of the application is the 3D game engine "Panda3D". Panda3D is a game engine, a framework for 3D rendering and game development for Python and C++ programs [17]. It was selected because it is an open source project and free for use and supports a high level programming language – Python. The high level programming language allows to speed up the development process. This was an important requirements for this research project, as this work is not only about studying 3D programming but also the getting a prototype for exploratory search realized. Another important aspect is the possibility to use either DirectX[12] or cross-platform OpenGL[15]. As a result the final application runs on several platforms. The engine itself is written in C++ programming language and uses an automatic wrapper-generator to expose the complete functionality of the engine in a Python interface. Thereby the engine allows rapid development and keeps the performance of a compiled language in the engine core. It is also possible to directly access the engine using the C++ code. The Python runtime is included in the Panda3D distribution and an end-user does not need to configure the installation.

Originally the engine was developed by the DisneyVR studio as a proprietary project. Since 2002 the engine was released as an open source project to be able to work with universities on research projects. The transition to open source allowed Carnegie Mellon's Entertainment Technology Center[4] to join in the development of the engine. The Carnegie Mellon's Entertainment Technology Center team contributed to the project polishing the engine for public consumption, writing documentation, and adding certain high-end features such as shaders (programs to calculate rendering effects on graphics hardware). The community of the Carnegie Mellon's Entertainment Technology Center is quite active in its forum and helps developers to come up with a solutions.

2.2 The application structure

The application is divided into eight classes. The classes "graph3D", "world" and "simpleGraph" have the main part of the logic. The class "world" mainly handles environment, navigation and interface code. The class graph3D deals with a graph drawing (layout), animation and graph structure in memory. The class simpleGraph is a source of the graph data. The structure of the application was developed with the aim to make it as independent from a graph data source as possible. There are only six functions that define application interaction with the graph data source (see section 3.2 for details). In case of exploratory search the data source can be any object network and an easy extensibility was one of the main priorities.

In many aspects the current implementation with a simpleGraph as the graph data source is an extension of the developed graph visualization logic. The six functions that interact with a graph data source are located in the graph3D class. They are enclosed in a box with red borders (Figure 2.1). The realization of this set of functions is located in the simpleGraph class. This structure solution was developed after the request to extend the application with another graph source - Email Information Concentrator. During this work a set of six main functions was separated from the other logic to provide developers with a "minimalistic bridge" for a graph data source definition (see chapter 2.9).

The "node" class holds the information about a graph node. The main reason of the node class development is a need to define a cluster as a special purpose node and integrate clusters to the layout. In future it would have more complex structure (see chapter 4) .

The four other classes ("mouseMenu", "popUp", "contentScroll", "DirectWindow") are GUI (Graphical User Interface) objects. The mouseMenu is a mouse context menu object (Figure 2.8 and section 2.8). The popUp is an info label object that represents a text that appear on the screen if a mouse cursor is positioned on the data object (Figure 2.2 near the mouse cursor). The contentScroll class represents a scrollable, resizable and draggable transparent window that has a text as its content data (Figure 2.8 and section 2.8). The contentScroll object uses the DirectWindow object that has all logic for window representation. The DirectWindow class code was found in the Carnegie Mellon's Entertainment Technology Center forum. The author of the DirectWindow code and design is Reto Spoerri from Zürich, Switzerland.

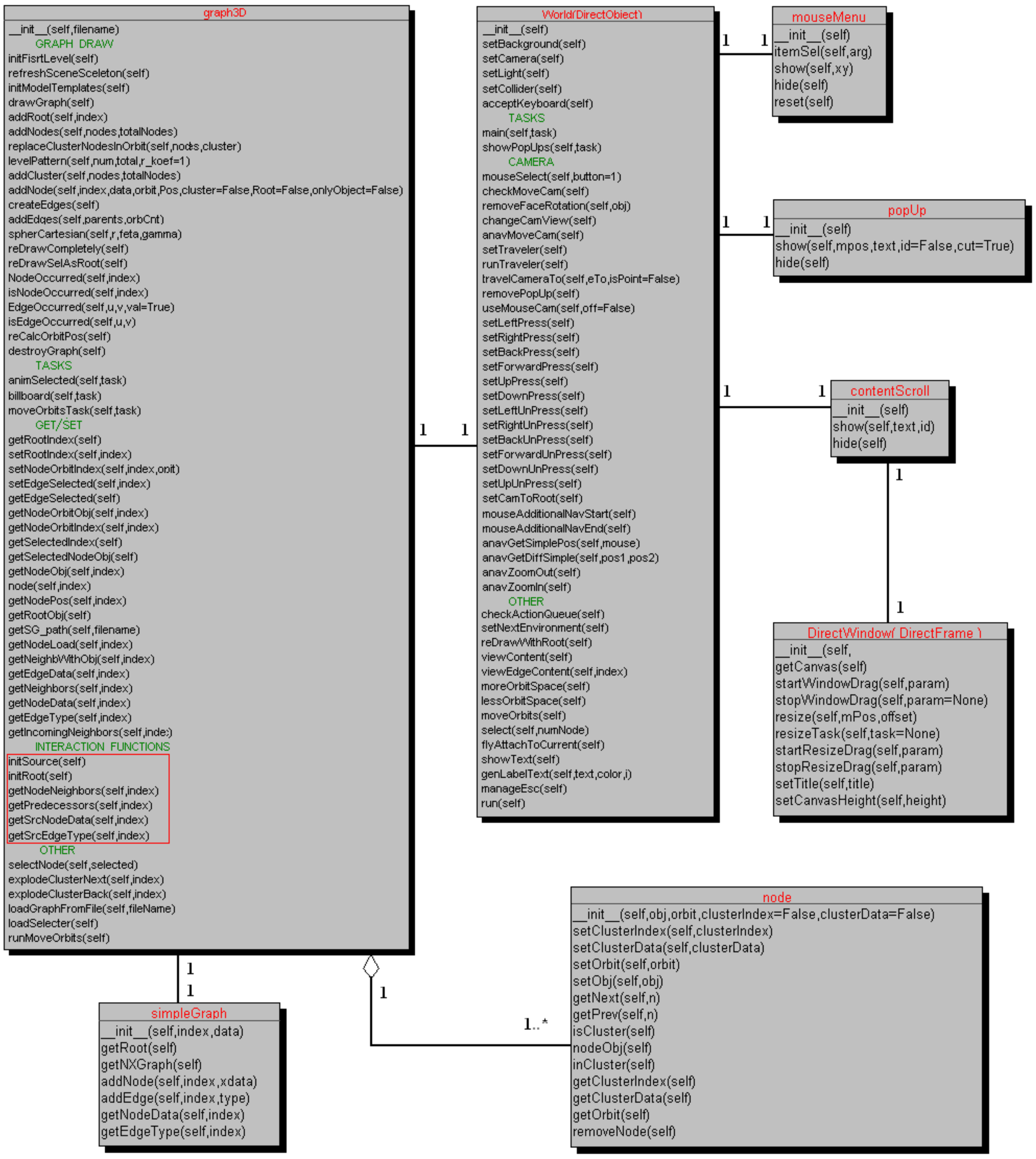


Figure 2.1: Simplified UML class diagram[13]

2.3 The scene graph

The Panda3D engine virtual world is initially empty. Usually, simple 3D engines use a list of objects for rendering. Panda3D handles it a little bit different. Instead of maintaining a list of objects to render, it maintains a tree of objects called the scene graph. The developer adds models to the virtual world by attaching them to the scene graph. The root node of the scene graph is the node called "render". All models or geometry attached to the root node would be rendered by the engine. The scene graph is a powerful tool because all transformations and position changes (translations) are relative to the parent node. Thus, if the developer attached the model as node A to the root node, changed its x coordinate to *10* and attached the other model as node B to the created node A then node B would also have x coordinate equal to *10* relatively to the root node. Actually, the node B has the x coordinate equal to *0*, but its coordinate space has a zero point at the parent node position. For example a model of man and his hat. If the hat model node is a child node of the man model node then in case of the man model movements the hat will be left on his head. However, if the hat model node is a child node of some man model node parents then the hat model will not move with a man model automatically. The model transformations (for example size or shape change) are also inherited from the parent node. This feature can help with exploratory search process. In the futrue applications a user will be able to modify a desired node and all subgraph connected with that node will inherit the modifications. This is a nice tool for selection of connected components based on the layout hierarchy.

This is very powerful method for complex scenes. It is easier to define the behavior of the object in the parent node coordinate system it must interact with. It is also very useful in reverse order. As an example we can take the rotation of the earth planet (object A) around the sun (point $C(x,\ y,\ z)$) with radius r on x, y plane. The classic solution of this problem is: $A(t) = (rsin(t) + x,\ rcos(t) + y,\ z)$. Nevertheless, the scene graph properties can give developers another way to solve it. The dummy invisible node N can be created at the point $C(x,\ y,\ z)$ (with position of sun) and the earth (object A) can be attached to it with desired radius r as x or y coordinate. Then the dummy node N could be rotated around the z axis. As a result the earth (object A) will rotate around the point sun (with point $C(x,\ y,\ z)$) because the zero point of the parent node (dummy node) coordinate system is rotated. To estimate the benefits of this method, developer can consider more complex scene like full emulation of the solar system, where objects (like sun, planets, satellites or asteroids) rotate around many points. In this case, instead of complex equations developer can create the proper scene graph structure using the given technique.

Most of the models are created in the free open source 3D content creation suite called "Blender" [1]. The Panda3D accepts the *.egg 3D model files. One of the ways to convert the model created in Blender to the egg format model is to export it in *.x format (DirectX native) and then convert it with a program called "x2egg" provided with Panda3D engine. There are also some plugins for Blender to make an egg file export automatically. The plugins are "Chicken", "EggX2" and "Panda3DExporter".

The term "texture" in computer graphics mean a 2D image that is used to cover 3D models. During the development of the application, I created a number of simple objects and textures. The application contains geometry models (Figure 2.2 for details):

- Sphere model. This model is used by the graph nodes with a texture defined in a code. The same model is also used by the transparent "selecter" (green sphere indicate the selected object) object, but the selecter object uses another texture and objects with the same shape look different.

- Pyramid models. These models are used for edge start and end arrows. The end arrows are almost the same as designed in Blender, but the start arrows (appear near the selecter object) have a modified shape by the application logic. The pyramid models were designed with a shift from its origin point to avoid the math calculations for positioning outside the node/cluster model. When the pyramid model is placed to the same position as the node object has, it appears outside the node model because the arrow model has a shift that is approximately equal to the node sphere model radius. The only operation the application performs is adjusting the direction.
- Cube model. This model is used for a cluster object and environment map (see section 2.4). These objects use different cube models. As opposed to the environment box, the cluster object is designed to use only one picture for all its faces.
- Circle mesh. This flat model is used for the cluster control buttons. The same model is used for two buttons and each button has its own texture.

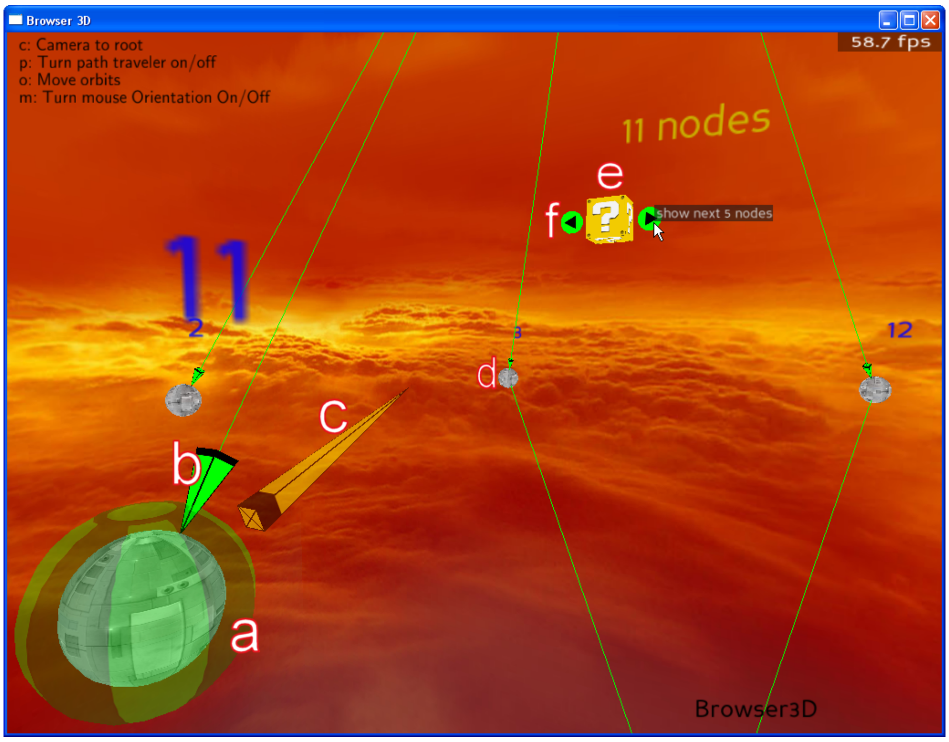


Figure 2.2: Scene graph models: a and d - selecter/node sphere models, b and c - arrow pyramid models, e - cluster cube model, f - circle mesh model

All the objects in the graph are created from the models in the list above. The engine provides a possibility to use a custom texture for a model.

In order to improve the performance, the "Instancing" technique is used. Animating of each separate model involves a lot of per-vertex matrix calculations. In case of the graph visualization application, it is the node object. Instead of creating the copy of the same model each time the developer can use only one model. The application creates only one node object and during the rendering places it to all node positions. In order to create this type of nodes, the developer must create a model and a target dummy node where to put the instance of the model. The developer needs to use a built in function $createdModel.instanceTo(dummyNode)$ to put the model instance to the dummy node. This method allows to use only one model for all nodes of the graph. When Panda3D renders the scene graph a reference to a model for each such node lead to the one model instance.

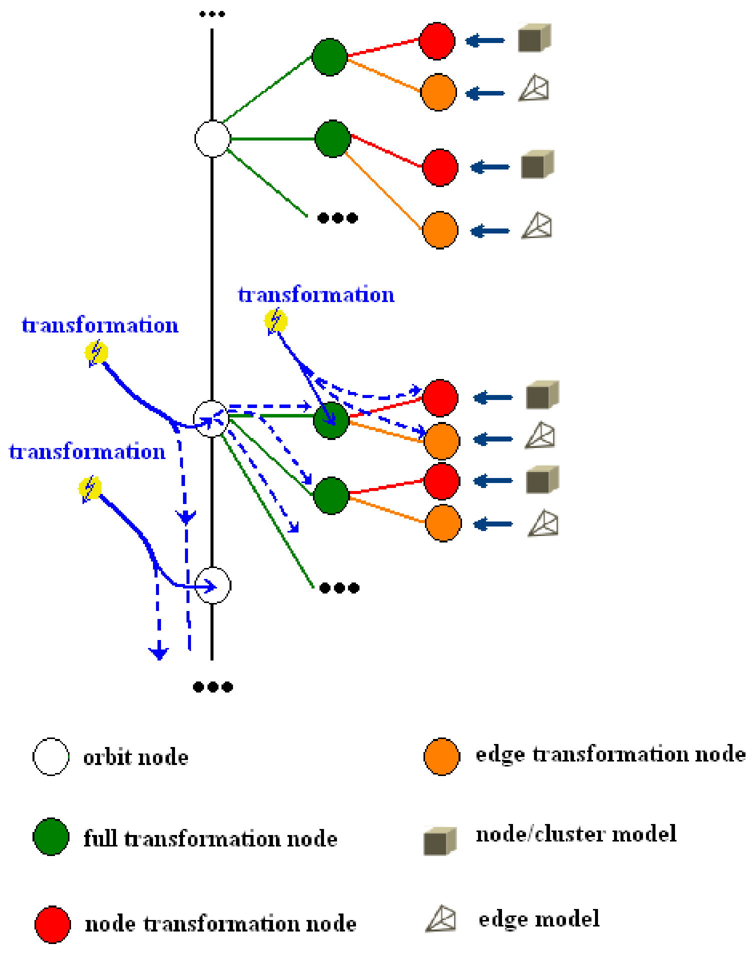


Figure 2.3: The structure of a scene graph

The scene graph of the application has a more complex structure (Figure 2.3) than the visible part of it. First of all, there is a set of “orbits”. The orbits are invisible dummy objects for position definitions of a 2D subgraph circular layouts or “levels”. All nodes of each level have the same distance (number of edges) from the root. A set of orbit objects is the skeleton of the scene graph. All transformations applied to the particular orbit are inherited by all the nodes attached to the orbits starting from the current (see blue lines on Figure 2.3).

Each node has a “full transformation” parent object (green circle on the Figure below). The transformations applied to that node would be inherited by a node and edge objects. The reason of such structure is an ability to apply transformations either to each object separately or to a specific group of objects. Currently this structure allow to transform:

- all graph (if apply transformation to the first orbit)
- part of the graph starting from defined orbit (if apply transformation to the defined orbit)
- node and its edge (if apply transformation to the "full transformation" node, green circle on the Figure below)
- specific model (if transformation will be applied directly to the model node)

In future work, the scene graph will have a more complex structure to allow the transformations of any group.

2.4 Environment

The environment of the application is created with the skybox. The skybox is a method to emulate the surrounding world (sky, nature, etc) for a 3D scene. When a skybox is used, the scene is enclosed in a cube with the desired environment projected onto the cube's faces. Each six faces of the cube are covered with a texture image. The images used for the cube faces are created using a technique called cube mapping. The texture created with this technique represents the appearance of six faces of a perfectly mirrored cube as seen from vantage points in each of the six cardinal directions (Figure 2.4). The viewpoint is located in the center of the cube.

The skybox model is a child object of the camera. The idea is to keep the camera in the center of the skybox model. Since the child node inherits the transformations from the parent node it would translate its position with the camera. However, the skybox would also inherit the parent node rotation and it will rotate together with the camera. To compensate this side effect the "CompassEffect" is applied to the skybox. The "CompassEffect" causes a node to inherit its rotation from some other reference node in the graph. In our case it is the root node ("render"). Thus, when the camera changes its position the skybox model also changes position together with the camera, but when the camera rotates the skybox does not inherit its rotation.

The realistic and static environment is important for user orientation. During the navigation it is easier and native to rely on the environment object position like sun or rock. A good user orientation in space makes the data exploration more efficient since a user can remember the explored node positions using the environment objects. Visually pleasant environment helps to enjoy the data exploration process.

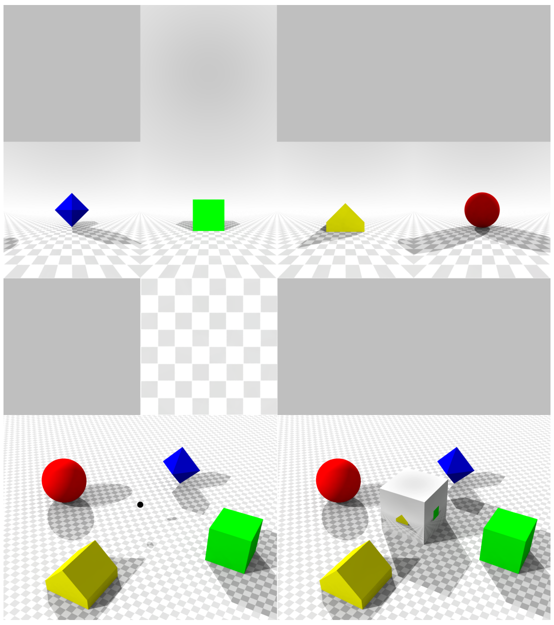


Figure 2.4: The lower left image shows a scene with a viewpoint marked with a black dot. The upper image shows the net of the cube mapping as seen from that viewpoint, and the lower right image shows the cube superimposed on the original scene.[7]

2.5 Graph Layout

The graph layout represents a set of circular 2D subgraph layouts on the planes (Figures 2.5 and 2.6). The ideas are showing the hierarchy of the relations, make the minimum number of edge crossings without additional logic and use simple and rigid layout to support an easy user orientation. These ideas cannot be realized with a force-directed layout (see 1.2.1) like in Interactorium (see 1.1.4).

Instead of using complicated algorithms that will make complex decisions for the layout structure the application uses a set of circular 2D layouts, where each circle of nodes is drawn on the different plane. The group of nodes on the plane represents a “level” or the distance to the root node. All the nodes drawn on each such plane represent a group of nodes connected to the previous level. The drawing of graph is started from the predefined root node with position x, y, z coordinates equal to zero. All the root neighbor nodes are drawn on the plane below (Figure 2.6). After that, all the neighbors of the drawn group are drawn on the second level from the root and so on. This layout idea shows the hierarchy of the relations in respect to the root node and allows the generation of hypotheses by the user.

This layout decision allows to reduce the space of the classic circular layout (Figure 2.5). The problem of the classic circular layout is the fact that each next circle of nodes must have the radius greater than the previous circle radius. In the developed application the radius of each circular subgraph depends only on the number of nodes in that group. This solution allows to reduce the space by utilizing the 3-rd dimension and keep the structure simple.

The application layout structure decision guarantees that relations for a particular node in the graph will appear only with previous level nodes, next level nodes, or with nodes in the same group. Thus, the neighbor nodes of the node a user is interested in are always near the object in question and the direction (next or previous group) represents the distance with respect to the root node. This allows to avoid long edges and provide a user with information in respect to the root node. The application allows to redraw the graph layout with the selected node as a root node and allows to apply this feature to the particular node on the fly.

Another important possibility that provides 3D graph exploration is more flexibility in changing the viewpoint. This is important when a user wants to focus on a group of nodes, a group of relations or both. Thanks to the dynamically growing entertainment industry the 3D engines are perfect tools for data visualization in space and provide tools to easily implement such a navigation. We consider this an advantage as it is a synthesis of layout and navigation.

The application uses a spherical coordinate system to calculate the positions of nodes in the layout. The formulas for the layout pattern is: $r = \frac{10(N-1)}{\pi}$, $\gamma = \frac{2n\pi}{N}$ and $\theta = const = \frac{\pi}{2}$, where N is the amount of nodes in the group and n is the order number of each node. After obtaining the results the spherical coordinates are converted to the Panda3D native Cartesian coordinates with the formulas: $x = rsin(\theta)cos(\gamma)$, $xy = rsin(\theta)sin(\gamma)$ and $z = rcos(\theta)$. This solution instead of Panda3D technique (see section 2.3) is selected to give more control for the layout pattern in future (See layout ideas in section 4).

The layout pattern formulas given above define the circular layout of only one group on the plane (because $\theta = const$). To distribute the groups between levels the algorithm uses the scene graph technique. In the beginning, the application creates the skeleton using dummy nodes called "orbits" (the graph drawing in space is in many ways similar to solar system) along the z axis with customizable distance. Each group of nodes has its own orbit. During the layout drawing, each group is attached to the different orbit node of the scene graph skeleton and inherits its orbit position as the origin of the coordinate system. This approach simplifies the math part of the algorithm and allows to simplify developing of more complex logic in future. The other advantage of this approach is the access point for transformations of each particular group or a number of groups.

The graph visualized in the application is a directed graph. The layout algorithm traverses the graph starting from the predefined root and fills the levels according to the distance to the root node. However, in case of the directed graph not all nodes can be visited. The application has a setting to either show the incoming edges or not. If this setting is turned on the graph will include all connected nodes like for not directed graphs. The simplified version of the layout algorithm is provided below:

```
while nodes > 0 and orbitcnt <= Limit:     #while there are traversable
                                           #nodes and we didn't reach the
                                               limit
  sendStack = []                           #empty the stack we send for
      drawing
  totalOrbitNodes = 0                      #reset counter of nodes in the
      group
  for parent in nodes:                     #for each node in the stack
    neighbors = getNeighbors(parent)       #obtain its neighbors
    for node in nghbrs:                    #for each neighbor
      if not isNodeOccurred(node):         #check it is not drawn (if cycle
          )
        NodeOccurred(node)                 #set node as visited
        sendStack.append(node)             #add node to the stack that we
            will
                                           #send for drawing
        totalOrbitNodes++                    #increase the amount of nodes in
                                             #the group
  if totalOrbitNodes > maxNodesInOrbit:     #if the number reached the
     maximum
                                             #amount of nodes in the group
    ddCluster(sendStack, totalOrbitNodes)#then add them as cluster
  else:                                      #otherwise
    addNodes(sendStack, totalOrbitNodes) #draw the group of nodes
  nodes = sendStack                          #update the traversable nodes
```

The current layout solution is not effective without hiding the nodes to the cluster in case of huge graphs. The object marked with letter “e” on the Figure 2.2 represents the cluster. The application automatically creates the cluster object instead of drawing a group of nodes if the amount of nodes in the group reaches some customizable limit k. Clusters have two buttons one on the left – “previous” and one on the right – “next”. The user can explode the cluster showing the k next nodes or k previous nodes using the buttons near the cluster. Buttons represent the circle mesh objects always facing the camera. Moreover, when the camera moves along the graph the buttons rotate around the cluster object and keep their positions on the left and on the right. The cluster drawing improves performance and simplifies the layout. It allows a user to focus on the desired node group.

If a user explodes k nodes from the cluster they also can have relations with another hidden nodes left in the clusters. In this case the user will see the blue relations between nodes and the cluster (Figure 2.8). The blue relations say that the cluster has at least one node that has a relation with the visible graph node. The blue relations can be switched off to simplify the scene. The cluster is positioned a little bit above the level and relations between nodes on that level will not intercept the cluster model.

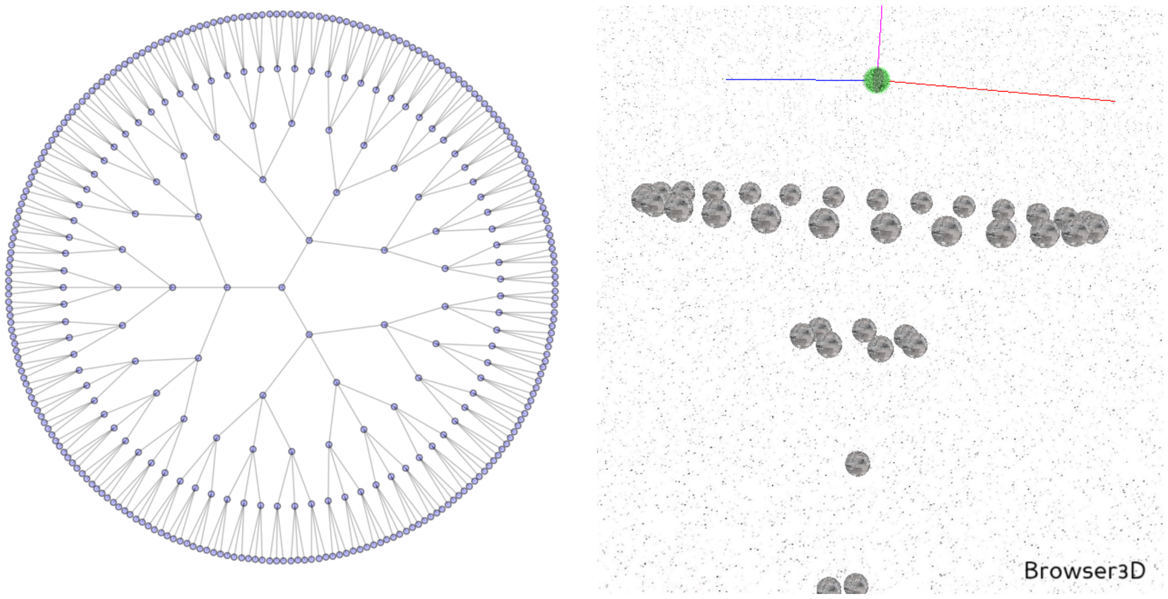


Figure 2.5: Circular layout (taken from [5]) and 2.5D layout of the application prototype.

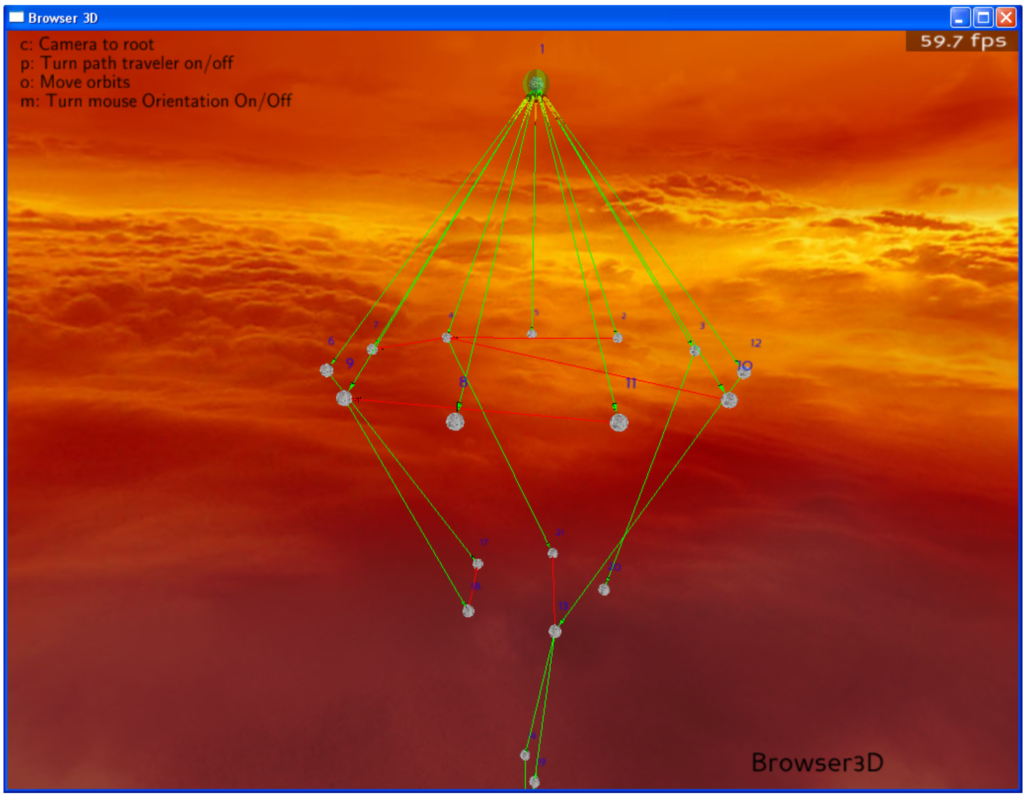

Browser 3D
c: Camera to root
p: Turn path traveler on/off
o: Move orbits
m: Turn mouse Orientation On/Off
59.7 fps
Browser3D

Figure 2.6: The graph visualization in the application

2.6 Camera

The camera is considered as a child node of the "render" object and the navigation could be simplified by the same techniques as described in the section 2.3. By default the camera node is created automatically and has the "perspective lens" object that behaves the same way as the physical lens in a photocamera works or the same way the lenses in our eyes work. The lens object of the camera has many customizable parameters like the field of view (FOV), aspect ratio, or film size. The FOV is modified in the application and has the greater angle value.

The user starts the navigation process near the selected root node looking at it. The position and point to look at are customizable and located in the "world" class "__init__" function. The Panda3D has some render effects considered for the camera that could be applied to the scene models. One of them is the "Billboard Effect". The billboard effect causes a node to rotate automatically to face the camera (regardless of the direction from which the camera is looking). The billboard effect is used in the application for node labels. For every camera viewpoint the labels are always facing the camera and user can read them from any position (if close enough).

Object selection and collision detect

The selection of the 3D object by the mouse is not as trivial as for 2D graphics applications. To select the object developer must "shoot" from the mouse position to the scene with a ray. Then detect the collisions of the scene objects and the ray and, finally, sort the objects in order to get the closest to the camera. Collision detection allows to realize that two objects are touching each other. The collision detection in Panda3D is handled by the “CollisionTraverser” object.

In order to show the popup labels an application uses the same technique. During the navigation an application is automatically “shooting” to the scene graph from a mouse cursor with a customizable period of time. If some data object is detected, then an algorithm calls for unhiding the popup label and sets its text and position. The simplified algorithm example is provided below.

```
if mouseMenuhidden:                                      #if popUp is not disabled
  frameCnt = tmpCnt + 1                                  #calculate frame delay
  if frameCnt == delay:
    frameCnt = 0                                         #reset delay
  if mouseWatcherNode.hasMouse() and frameCnt == 0:      #if mouse is inside the window
                                                         #and delay is cleared
    mpos=mouseWatcherNode.getMouse()                     #get the mouse position
    pickerRay.setFromLens(camNode,                       #set ray position
                          mpos.getX(),                   #according to the camera
                          mpos.getY())
    collisionTraverser.traverse(graph.orbits[firstOrbit]) #traverse the scene graph
                                                         #starting from the first level
    amount = collisionHandler.getNumEntries()            #count a number of collisions
    cnt = 0
    found = False
    if amount > 0:                                       #if there are collisions
      collisionHandler.sortEntries()                     #set the right order
      while cnt < amount:                                #for all found collisions
        pickedObj=collisionHandler.getEntry(cnt).getIntoNodePath() #get object
        pickedObj=pickedObj.findNetTag('type')           #check if object has a "type"
        if not pickedObj.isEmpty():                      #if object has a "type" tag
          cnt = amount                                   #stop the loop next time
          id = pickedObj.getTag('id')                    #get object id
          type = pickedObj.getTag('type')                #get object type
          if type == 'node':                             #if object is a graph node
            if popup.hiding or id != popup.id:           #if popup is not active
                                                         #and this is a new node
              popup.show(mpos,graph.getNodeData(id), id) #show popup
          if type == 'edge':                             #if object is an edge
            id2 = pickedObj.getTag('id2')                #get target node id
            idtmp = id + id2
            if popup.hiding or idtmp != popup.id:        #if popup is not active
                                                         #and this is a new edge
              popup.show(mpos,                           #show popup
                    graph.getEdgeType((id,id2)),         #with edge data
                    idtmp)
          if type == 'cluster':                          #if object is cluster
            self.popup.show(mpos,                        #show popup
                      graph.getNodeData(id),             #with cluster content
                      id,
                      cluster = True)
          if type == 'cluster_control':                  #if object is cluster button
            popup.show(mpos,                             #show popup with action desc.
                  'show '+getAction()+' '+getNodesInOrbit()+' nodes',
                  cut = False)
        found = True                                     #set found flag
      cnt = cnt + 1
  if not found:                                          #if nothing is found
    if not self.popup.hiding:                            #and popup is active
      popup.hide()
else:                                                    #hide if disabled, but active
  if not self.popup.hiding:
    popup.hide()
```

2.7 Task manager

The Panda3D task manager (“taskMgr”) is a global object that handles all tasks. The task in Panda3D is a subroutine that is called by engine every frame. The task allows developer to update the world between rendering steps. Some important application features are made by the tasks in navigation and user interface. There is also one problem with object orientation that was solved with the development of an additional task. The billboard effect described previously (see section 2.6) is a render effect and considered only for camera. Actually, the model is not rotating in the global sense and collision mesh is left on the same place. This is a problem for object selections by mouse click (see subsection 2.6). It took a while to find the reason of that behavior. Thanks to the active community of the Carnegie Mellon's Entertainment Technology Center, this problem was solved replacing the billboard effect with a task that rotates the scene subgraph with cluster control buttons (Figure 2.7).

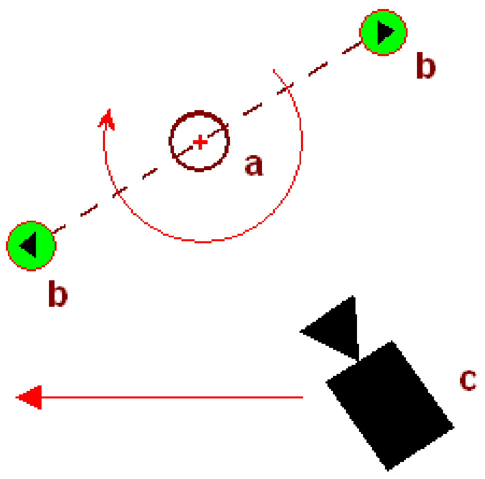


Figure 2.7: Cluster control buttons. a - dummy node; b - circle mesh models that act as buttons; c - camera

Intervals and Sequences

The interval in Pand3D is an object that acts like a task, where user can define a property change over some period of time. In case of camera it can be two position points $A(x_0, y_0, z_0)$ and $B(x_1, y_1, z_1)$. The interval can be used to animate the camera flight (position change) from position A to position B over some time t. The camera navigation by mouse clicks in the application is made by the intervals and sequences. The sequence in Panda3D represents a complex task that can include multiple intervals executed one by one. However, the developed application uses more comfortable Panda3D class called "Parallel". The Parallel class is similar to the Sequence class except it executes all intervals at the same time and run them in parallel. When user clicks on the desired node C the application calculates an end point and direction of view of the camera near the node C. After the calculation of the points and direction of view the application creates intervals for position and view direction change and executes them in parallel using the Panda3D Parallel object. As a result a user can see the camera flight to the selected node from the previous position. In future the application will have more such automatic "flights" (see chapter 4).

2.8 User interface and navigation

The interface was developed with aims to support the navigation only with a mouse or with a composition of mouse and keyboard. In future there is a plan to develop a navigation by face recognition using a web camera (See chapter 4). By default Panda3D provides a navigation technique for moving a camera with a mouse, but it is not comfortable and flexible. To implement another camera controls the default navigation must be disabled first with a function *base.disableMouse*().

The camera position change is handled by the two sets of functions. The first set of functions accepts the keyboard button events and modifies the flags: "right", "left", "back", "forward", "up", "down". The application has a main task (see section 2.7), where the second set of functions is executed. The second set of functions moves or rotates the camera according to the flags and navigation modes every frame. This decision was made to smooth the camera movements.

There are two camera navigation modes. The first mode is a "free look", where a user can move among the graph and "selected" mode, where the camera is attached to the node and the same camera control buttons (see manual in chapter 3 for description of controls) will rotate the camera around the selected node. The navigation modes make interface more flexible and allow a users to choose the navigation style he likes (according to the goals).

During the graph data exploration process a user mostly clicks on the different nodes to check the data and relations. The application has some logic to accelerate these actions. When user clicks on the desired node with a left mouse button the application immediately moves the camera to the selected node. The application calculates the position $Pos_1(x_1,\ y_1,\ z_1)$ near the node using the initial position $Pos_0(x_0,\ y_0,\ z_0)$, target node position $Pos_n(x_n,\ y_n,\ z_n)$ and a distance to target node $d = const$ by the formulas:

$$\overrightarrow{v(x_v,\ y_v,\ z_v)} = (x_n - x_0,\ y_n - y_0,\ z_n - z_0)$$

$$k = \frac{d}{\sqrt{x_v^2 + y_v^2 + z_v^2}}$$

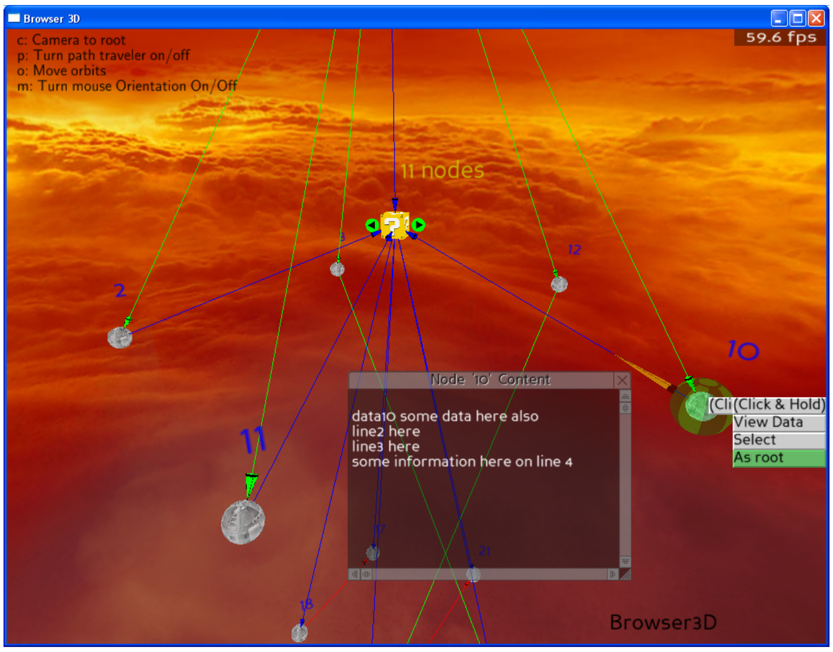
$$Pos_1(x_1,\ y_1,\ z_1) = (x_n - kx_v,\ y_n - ky_v,\ z_n - kz_v)$$

To access the graph node data a user needs either to put a mouse cursor over the node and observe the popup textual data (limited to thirty characters) or to double click (left mouse button) on the desired node in order to open a window with a full content. The window interface object is not a built-in Panda3D functionality (see section 2.2). User can use already classic "drag-and-drop" window position movements. The content inside the window is scrollable and a window is resizable. One important property of the window movements is automatic window positioning near the corner if window is moved off-screen (automatically makes it visible).

After a selection of the desired node, the selecter object (a transparent green sphere with stripes) indicates that the node inside the selecter is currently active. After the selection of a node the outgoung relations are marked by the additional transparent start edge arrows. The start arrows also can have the data and popup text and window objects are also accessible for that graph elements. In exploratory search process user must be able to easily access the data behind the visible structurized objects.

It is possible to redefine the root node and redraw the graph during the exploration process. This feature allows to restructurize the graph in respect to the desired node and explore the resulting relations. This mean that user can define the point of interest and apply the power of the layout to observe the surrounding data. This feature is accessible from the context menu if user clicks on the node with a right mouse button. The context menu is a realization of Panda3D "DirectOptionMenu" GUI class. In a DirectOptionMenu developer must define the menu elements and a function that will be automatically called by Panda3D on element selection event. One disadvantage of this class is the need to hold the left mouse button in order to choose the menu elements. This is not intuitively understandable and convenient for users, who are used to the usual mouse context menu in modern operation systems. In future development it would be replaced with a custom menu made in the same way as the window object.

Overall, the interface supports the graph data exploration, but it is not yet featured and rich in navigation solutions. It is a completed prototype for the exploratory search task tests, but the full power of the 3D engine interactivity will be implemented in careful and thorough future work (see chapter 4).



data10 some data here also
line2 here
line3 here
some information here on line 4

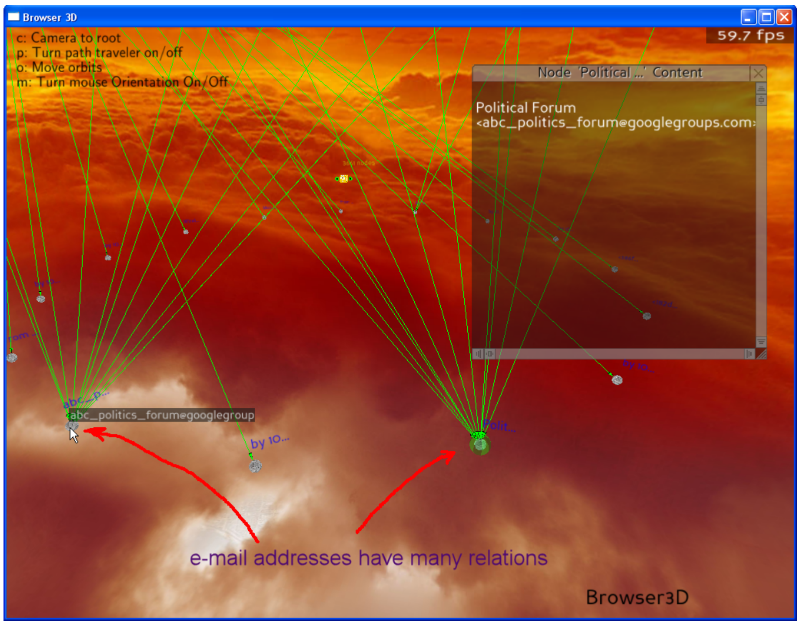
Figure 2.8: The GUI elements

2.9 Practice and results

The application is already used in another project called "Email Information Concentrator". The aim of this knowledge management area project is a research for the solution of the growing e-mail message overload problem. This project uses the graph to make a complex system from the data parts of the e-mail messages. The idea is to keep only unique mail message data parts and create relations for the duplicates. This technique allows to reduce the data duplication. The graph visualization application is used for the resulting e-mail graph examinations and research of the data relations.

The graph data source of the e-mail graph is integrated to the application using the six functions described in the section 3.2. The graph data is exported (serialized and stored as file) with a “pickle” python module. The serialized object has a logic to support the six actions defined in the functions. The graph data is then loaded by the graph visualization application and the user can explore the resulting graph.

The e-mail graph has a huge amount of nodes per each level. This problem is successfully solved by the clusters created instead of the nodes (see section 2.5 for description about clusters in the graph layout). The user review the nodes exploded from the cluster in smaller groups. The e-mail data is accessible by the popup text or by the node/edge content window (Figure 2.9).


Browser 3D
c: Camera to root
p: Turn path traveler on/off
o: Move orbits
m: Turn mouse Orientation On/Off
59.7 fps
Node 'Political ...' Content
abc_politics_forum@googlegroup
by 10...
e-mail addresses have many relations
Browser3D

Political Forum
<abc_politics_forum@googlegroups.com>

Figure 2.9: Graph of e-mail data

Chapter 3

Manual

In order to run an application the free and open source 3D engine Panda3D must be installed[17] (see section 2.1 about the application engine). After the installation the user can start the application by running the *main.py* script with a python provided by Panda3D (typing *python main.py* if no other active python installation is already available). After the execution a window of the application will occur (Figure 2.6 as an example). The root node will be selected on application load. After the window loading a user can immediately start the navigation and data exploration process.

In order to move the camera, the user can use arrow buttons for the camera movements in the vertical plane and use the mouse scroll to move the camera forward or backward (like zoom-in and zoom-out). Another possibility is to use buttons: “w”, “s”, “a”, “d” to move the camera in horizontal plane and use a “Space” and “Ctrl” to move camera up and down. It is more convenient because of full control in all six directions by only the left hand. The mouse scroll in this case can be used as forward and backward movement acceleration.

The change of the camera view direction is handled by the right mouse button. The user can press and hold the right mouse button and move the mouse pointer. The camera will change the direction of view according to the mouse pointer movements. One important feature is added to that way of camera view direction change. If camera is moving (one of the movement flags is active) the effect of the camera direction view change is stronger.

There is also another way to change the camera direction of view. If the user presses the “m” button another mode will be activated. In this mode user does not need to hold the right mouse button. The camera direction view is continuously changing according to the mouse pointer position. Near the center of the application window the camera direction view change is minimal, but when the mouse pointer is near the window border the camera view direction is changing faster.

In order to focus on the desired node a user can click on it with a left mouse button and the camera will immediately move to the point near the desired node. The desired node will be in center of the application window. The end point distance to the desired node is customizable (see next section for the customizable settings).

During the camera movements and camera view direction changing a user observes the graph structure and the data behind the objects in order to generate hypotheses. The data can be reviewed in short using a popup text or in full size using a window (Figure 2.8 for window object and Figure 2.2 for popup text examples). The user can see the popup text with a maximum length of thirty characters, when the mouse pointer is hovering over the data object. The window can be opened in many ways. The simple way is to double click on the data object. There is an additional possibility for edges to press only the right mouse button over the edge object because the left button will notify the application to move the camera near the target node. For a node object the window with a content data is also accessible from the mouse context menu.

The mouse context menu opened for a node has the following elements:

- View Data. If this menu element is selected an application opens the window with an object data.
- Select. If this menu element is used an application changes the camera navigation mode. Camera will be rotated around the node.
- As Root. If this menu element is used the application redraws the graph with a selected node as root.

When the user finds some interesting node, the user can use the "select" mouse context menu element and change the navigation mode. During this mode a camera is rotating around the selected node using the same control buttons: either the arrow buttons or "w", "s", "a", "d" buttons. When the camera is rotating around the node, the user can better observe node relations. If the node needs to be observed in context of a greater amount of surrounding nodes, the user can use the "As Root" mouse context menu element and redraw the graph in respect to the desired node (with a desired node as root). After the use of this feature, a camera will fly to the root node.

The cluster objects act the same way as the nodes, but the data of the cluster would be a list of node indexes hidden into it. The "As Root" mouse context menu element is not accessible for a cluster because of its nature. However, the user can select the cluster with the left mouse button and the camera will move to the cluster object the same way as to the node object. User can access the data inside the root the same way as for the node, but the data of a cluster is a list of nodes hidden inside. The popup text and window objects are also working for the cluster object.

As opposed to the node object, the cluster has additional features and geometry. The cluster object can be exploded by a user. The user can click on the buttons near the cluster object and draw the n next or previous nodes hidden in the cluster on the cluster level (Figure 2.2). The right cluster button will draw the next n nodes. The left cluster button will draw the previous n nodes. The number n is a global level node number limit and can be redefined in the application code (see section 3.1 for details). If the number of nodes on any particular level is greater than n, the nodes of this level will be replaced with a cluster object.

The environment of the application 3D scene can be changed during the exploration process by pressing the "e" key. After pressing the "e" key the application loads the next skybox model from the next environment folder located in the project folder in the *./environment* directory (see section 3.4 for more details).

The application has two possibilities for a graph data import. The first one is using the CSV (Comma Separated Values) file (see section 3.3 for the file format). The second opportunity is a development of an application extension (see section 3.2).

3.1 Customizable settings

The application has a number of constants for graph drawing, the navigation, and interface. This section will give a small overview of their purpose and location. In future work this will be integrated in the GUI.

The settings are located in init functions of graph3D and World classes. The detailed list of settings is provided below:

- World.py
  - *anavSpeed* - is a setting that defines a sensitivity of the camera view direction change, when user is holding the right mouse button.
  - *flyToNodeTime* - is a time (in seconds) of the automatic camera flight to the selected node. Lower values mean faster speed of the flight.
  - *zoomFactor* - is a coefficient that increases/decreases the zooming step. Bigger values mean greater zooming step.
  - *fov* - or field of view in degrees (see section 2.6).
  - *doubleClickInterval* - the interval in ms for the left mouse button clicks.
  - *initPos* - initial position of the camera.
  - *worldScale* - the coefficient for the environment (skybox, see section 2.4) size.
- Graph3D.py
  - *orbAmount* - the maximum number of levels to draw.
  - *maxNodesInOrbit* - the maximum number of nodes on one level.
  - *spacing* - the distance between levels
  - *hideEdgesWithCluster* - if set to *True* then application will not draw the relations with a cluster object (blue color relations on Figure 2.8).
  - *showNotTraversableButConnectedComponents* - if set to *True* the application will draw all connected nodes even if they are not reachable from the root node.

3.2 Extending the application

In order to extend the application, the developer must redefine 5 basic functions in the Graph3D class:

- *initSource*() - this function must initialize the source logic and load the data.
- *initRoot*() - this function must define the initial root node. Developer must put the index f the initial root node to the global variable *rootIndex*.
- *getNodeNeigbors*(*index*) - this function must return a list of indexes of given node neighbors.

- *getPredecessors*(*index*) - this function must return a list of indexes of given node predecessors.
- *getSrcNodeData*(*index*) - this function must return the data for a given node in string format.
- *getSrcEdgeData*(*index*) - this function must return the data for a given edge in string format.

These six functions provide all the needed logic for a graph drawing in the developed application. The default realization uses a CSV file as a data source and the “networkx” [16] python package for the graph data storing and manipulations. The realization uses the same six functions provided above.

3.3 Input CSV file format

The CSV (Comma Separated Values) is a textual file format used to store the table like structures. In the CSV file format the data pieces are usually enclosed by the quotes and separated by commas. In order to minimize the risk of errors, the application uses a CSV file format, where the values are separated by the semicolon. The CSV file contains two different types of data: nodes with their data and edges with their data. In order to use only one input file and separate the sections with different data, the special section names are used. The section name “%SECTION_OF_NODES%” is used to notify the application that it deals with the nodes and their data. The section name “%SECTION_OF_EDGES%” notifies the application that it deals with the edges and their data.

In the section of nodes the index of a node is defined on the first position. The second position defines the textual data of a node. In the section of edges the first and the second positions are the out of node index and destination node index respectively. The third position defines the textual data of an edge.

An example of correct input CSV file is provided below for a reference:

```
"%SECTION_OF_NODES%";
"1";"From"
"2";"To"
"3";"test@mail.ulno.net, 763534rjdsbnf"
"4";"data4"
"10";"data10 some data here
also line2 here
line3 here
some information here on line 4"
"11";"data11" "100";"data100"

"%SECTION_OF_EDGES%";
"1";"2";"data 1 -- 2"
"1";"3";"data 1 -- 3"
"1";"4";"data 1 -- 4"
"1";"10"; "24,100";
```

3.4 Installing a custom skybox

The skybox or environment of the 3D scene (see section 2.4 for details about the skybox) can also be easily customized. The application project folder has a directory *./environment* where different environment graphics is stored. The only requirement for an extension is the existence of the separate folder and a *skybox.egg* model. In other words, the environment customization must be stored in separate folder and have a *skybox.egg* model file. The textures must be defined inside the *skybox.egg* model file. The easiest way is to copy the *skybox.egg* model from another environment folder and modify/replace the textures for the faces of the cube. The application automatically lists the folder names in the *./environment* folder on initialization. When a user presses the "e" button during the graph data exploration process, the application tries to load the *skybox.egg* model from the next folder in the list. An example of a customized environment can be seen on Figure 3.1.

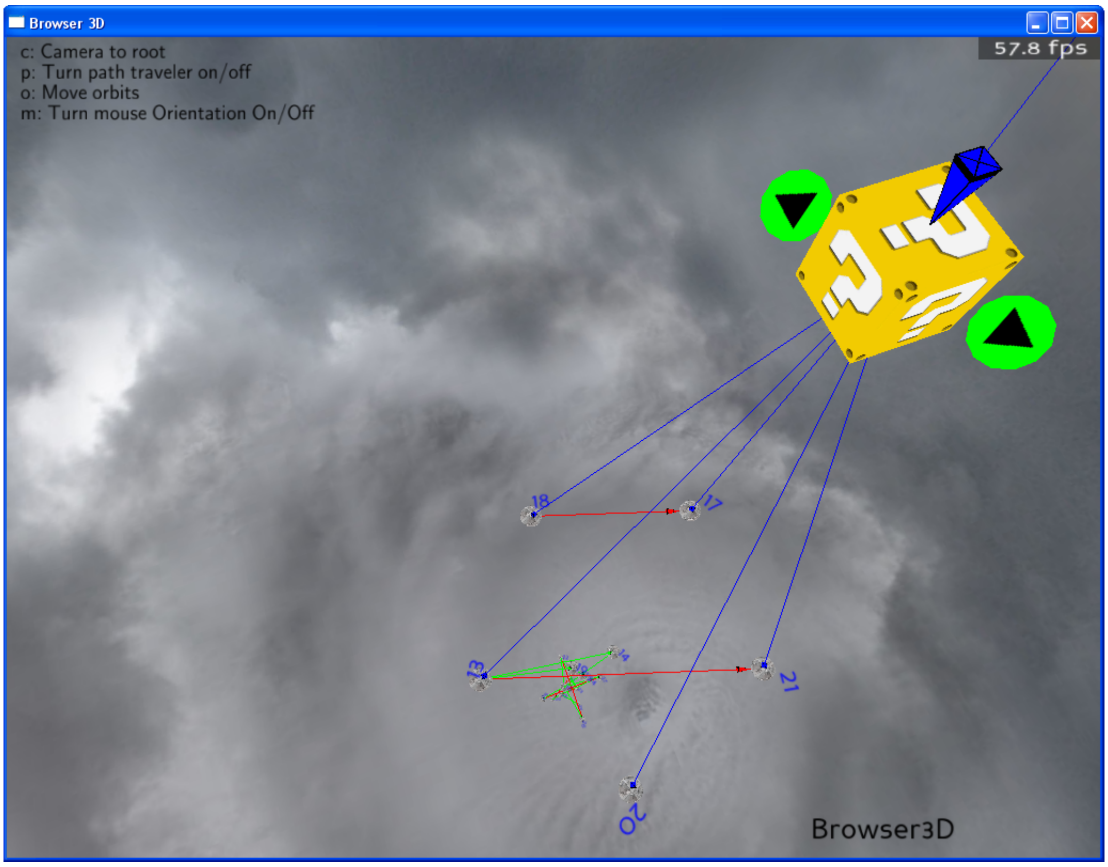


Figure 3.1: Customized environment example

Chapter 4

Future Work

The current application is a first version of a 3D graph explorer. The aim of this work is the research of 3D exploring techniques and related technologies as well as developing the application. Future implementations will also support a number of interface and navigation solutions. One of the most ambitious goals is an interface design that will help people involved in a common task achieve their goals in an exploratory way. It will support a collaborative work and navigation using the latest human-computer interaction (HCI) solutions. A possibility of collaborative work is another important aspect of the exploratory search strategy.

Collaborative work can be achieved by using a multitouch interface. The multitouch interface (hardware part) could for example be supported with Nintendo Wii[6] technology. There are a lot of projects[14] based on the Managed Library for Nintendo's Wiimote[11]. This is a low-cost tracking support for a lot of interface solutions with outstanding visual experience. One of the most exiting projects is tracking the head position that allow to make an illusion of watching to the scene like through a usual window. This solution can help to solve the problem related to the flat nature of displays (see thesis introduction and beginning of the chapter 1 for the problem description).

The less ambitious goals are mostly related to the more intelligent interface and navigation and are listed below:

- The camera movements will be partially replaced with navigation patterns by mouse clicks. The challenge of this task is the finding of cases, where it can be performed. A simple example of such a pattern is to zoom out and move outside the circular subgraph of the selected node and show the whole subgraph on the screen. This is like the opposite effect of the node selection. This action can be bound to the mouse buttons pressed simultaneously and for backspace keyboard button in analogy to the "Back" action in web browsers. The more analogies the interface will have, the easier and faster a user will learn and interact with the application.

- Pointing device gesture support that allows to accept drawn signs as action events (like navigation mode switches). It is especially effective with a multitouch interface support. Users can draw the signs to create new elements like an additional node, make complex selections and combine groups. Users can also activate/-draw additional control elements to make complex rotations and graph layout modifications. An example of the simple gesture of back/escape/abandon action can be found in Figure 4.1.

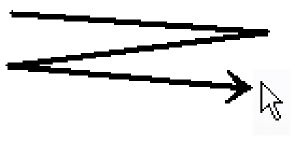


Figure 4.1: Example mouse gesture

- The logic for the node models will be improved to support a possibility to define a type of the node. There will be a special tool to make the matching pairs of node type and corresponding 3D model. This feature will allow to make the graph more understandable and will help a user to orientate. This feature also requires more design work in 3D modeling.
- The graph data source extension management will be more intelligent. There is a plan to develop a mechanism to manage the different graph source extensions using the GUI and rigid instructions for developers. This will allow to switch between graph data sources or even combine them.
- The navigation will have more different modes and it will be customizable in GUI. The customizations for camera navigation will be also extendable in the same manner as graph data sources will be. User will be able to choose the navigation mode dynamically during the exploration process.
- The layout logic for the graph will be improved to minimize the edge crossings. The nodes in the level group will be sorted in appropriate order.
- The cluster will have more features and a user will be able to define the amount of nodes to show. The system will also suggest more intelligent divisions into the groups to allow a user to see all relations for a given group located inside the cluster.
- There would also be more layout patterns. One of the planned layouts will put subgraphs (level groups) to the surface of sphere instead of the surface of plane. The subgraph in that case will not have the circular layout and will be more similar to a grid or star to utilize the space more efficiently. This layout requires additional logic for node placement to minimize the edge crossings (triangulations[28]).
- Intelligent selection algorithms will rely on distance to the object and will activate or deactivate some control elements if the distance to the object is long or short. As an example, if the number of nodes is huge, the cluster object is located far away from the exploded nodes and it is difficult to click on the buttons near it because of its size. The algorithm must recognize the click to the any cluster related object as selection of the cluster.
- The graph modifications: changes in the layout, hiding unnecessary nodes, adding new nodes, creating of new groups and deleting the group will be available with a possibility to save the modified graph layout.

- The structure of the graph in memory will be replaced with a networkx[16]. This will allow to support main graph theory algorithms in a efficient way (like shortest path finding).
- The nodes will support the drag-and-drop technique to simplify the graph layout modifications.

The project research in data exploration visualization with a combination of latest 3D graphics technologies and HCI solutions can give a magnificent outcome for exploratory search tasks individually or in groups.

Conclusion

The searching tasks become progressively important in the society with a growth of the information overload. The query results in the search engines return a lot of duplicate data and people need to spend a huge amount of time and doing multiple customized searches to find unique “pearls” or the area of information they need to cover. The exploratory search strategy is considered to partially solve these problems. The navigation and visualization of the data exploration is an important part of the exploratory search strategy and can be successfully implemented with a 3D graph.

Different graph layouts give a structurized overview of the data relations and help users to generate hypotheses based on the found node groups. Clustering allows to keep the clear layout and get more details on demand. Users can focus on the desired node group or a set of groups using the navigation mode they like. The utilization of the third dimension gives an extra space for graph layout and allows to make more complex and understandable hierarchical structures. All these advantages allow to represent more complex graph data structure in an understandable format.

In this work I have implemented a 3D graph visualization system with an emphasis on the exploratory search visualization goals. The application can be effectively used for the complex data networks and supports the “learning and investigation” key elements of the exploratory search systems. The application can handle large graphs and provide hierarchical graph layout supporting the better user orientation and hypotheses generation based on the explored graph data structure. The interface and navigation solutions allow to easily navigate by mouse clicks. The application is already used by other projects (see section 2.9 in chapter 2).

Moreover, the implementations for new challenging goals listed in the last chapter will make the complex search tasks easier and introduce collaboration and more native techniques to the process. The 3D models for different type of nodes will make an analogy to the manipulation with physical objects. The newest 3D graphics technologies introduce dynamism and more native interfaces based on the analogies with outside world. The new HCI solutions help users to involve more into the data exploration process and provide an outstanding usability experience.

The 3D graph visualization can give a key to the innovative search systems. The information is usually a network and data relations represent the main point of interest. The 3D graph visualization systems focused on the data exploration can give a renewed impetus to the classic search systems allowing to develop more complex algorithms based on the graph theory and hypotheses control in the 3D graph visualization software.

I plan to continue this project in a PhD program to implement part of the ideas proposed in the last chapter.

3D graafi uurimine

Magistritöö (20 AP)

Dmitri Danilov

Kokkuvõte

Ühiskonna üheks suureks probleemiks on informatsiooni üleküllus ning sellest tulenevalt on otsingutehnoloogia muutunud järjest olulisemaks. Teadlasi motiveerib arendama aina intelligentsemaid otsingusüsteeme see, et informatsioon on tihti korduv ja vajaliku informatsiooni leidmine keeruline. Kui otsingu ülesanded sisaldavad rohkem kui üht probleemi, siis on oluline arendada süsteem, mis suudaks koondada informatsiooni ja lihtsustada kogu otsingu protsessi. Üks paljulubavatest uuringuvaldkondadest on exploratory search (edaspidi uuriv otsing). Selle eesmärgiks on aidata otsijal õppida ja uurida uurimisprotsessi käigus.

Antud magistritöö eesmärgiks on realiseerida uuriva otsingu visuaalne programm. Valminud programm visualiseerib graafi ruumiliselt. Navigatsiooni ja kasutajaliidese lahendused võimaldavad uurida graafi andmeid ja luua hüpoteese, mis baseeruvad leitud andmetel ja andme struktuuridel. Programm joonistab graafi kasutades 2.5D paigutust. Graaf on struktureeritud ja koosneb ringjatest tasapinnalistest alamgraafidest. Antud rakendus loob automaatselt klastreid (cluster) joonistamise käigus. Autori poolt loodud programmi juba kasutatakse mõndades projektides ja seda on lihtne laiendada teistele graafi andmetele. Autor leiab, et antud projekt on suure potentsiaaliga ning tulevikus võib seda programmi effektiivselt kasutada keeruliste andmekogumite uurimiseks. Autor plaanib programmi edasi arendada doktorikraadi raames.

Bibliography

[1] blender.org - home. http://www.blender.org/.

[2] BRAINMAPS.ORG - Nodes3D graph visualisation tool. http://brainmaps.org/index.php?p=desktop-apps-nodes3d.

[3] CAIDA walrus visualisation tool. http://www.caida.org/tools/visualization/walrus/.

[4] Carnegie mellon's entertainment technology center. http://www.etc.cmu.edu/.

[5] Circular tree — NetworkX v1.1 documentation. http://networkx.lanl.gov/examples/drawing/circular_tree.html.

[6] Console at nintendo :: Wii. http://www.nintendo.com/wii/console.

[7] Cube mapping - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Cube_mapping.

[8] H3Viewer. http://graphics.stanford.edu/~munzner/h3/.

[9] HypView documentation. http://graphics.stanford.edu/~munzner/h3/HypView.html.

[10] Interactive visualization of large graphs and networks. http://graphics.stanford.edu/papers/munzner_thesis/.

[11] Managed library for nintendo's wiimote. http://wiimotelib.codeplex.com/.

[12] Microsoft DirectX. www.microsoft.com/windows/directx/.

[13] Models of the hyperbolic plane. http://www.geom.uiuc.edu/docs/forum/hype/model.html.

[14] .NET-based wiimote applications - brian peek's blog - BrianPeek.com. http://www.brianpeek.com/blog/pages/net-based-wiimote-applications.aspx.

[15] OpenGL - the industry standard for high performance graphics. http://www.opengl.org/.

[16] Overview — NetworkX v1.1 documentation. http://networkx.lanl.gov/.

[17] Panda3D - free 3D game engine. http://www.panda3d.org/.

[18] The programming language lua. http://www.lua.org/.

[19] RedfishGroup - santa fe. http://www.redfish.com/.

[20] RedfishGroup: 3D force directed graph layout.
http://www.redfish.com/research/graphLayout.htm.

[21] Tamara munzner's stanford home page.
http://www.graphics.stanford.edu/~munzner/.

[22] WilmaScope. http://wilma.sourceforge.net/main.html.

[23] Wilmascope 3D graph visualisation system. http://wilma.sourceforge.net/.

[24] Interactorium release notes 1.0, 2009.

[25] A. Ahmed, T. Dwyer, M. Forster, X. Fu, J. Ho, S. H Hong, D. Kosch\ützki, C. Murray, N. Nikolov, R. Taib, et al. GEOMI: geometry for maximum insight. In *Graph Drawing*, page 468–479, 2005.

[26] T. Munzner. Drawing large graphs with h3viewer and site manager. In *Graph Drawing*, page 384–393, 1998.

[27] shhong. Gallery of 2.5D graph drawings (VALACON project).

[28] E. Welzl. The number of triangulations on planar point sets. In *Graph Drawing*, page 1–4.

[29] Ryen W. White and Resa A. Roth. Exploratory search: Beyond the Query-Response paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, 2009.

[30] Yose Widjaja and Yose Widjaja. Skyrails graph visualisation system blog.
http://cgi.cse.unsw.edu.au/%7Ewyos/skyrails/.