Interactive Visualization of Complex Graphs

Marc Streit*

Institute of Computer Graphics and Vision
Graz University of Technology
Graz / Austria

Abstract

We aim at a visualization framework that empowers the user to reveal information that is not obvious just by looking at the graph. The keys to reach that goal are interactive graphs in combination with well established visualization methods. Techniques such as multiple views, linking & brushing, and focus + context are already implemented in the system. Furthermore the application is designed for fast and modular integration of alternative and/or new visualization concepts.

Keywords: graph, information visualization, interactive, application, Java, JOGL, OpenGL, linking, brushing

1 Introduction

The visualization of complex graphs is not trivial but the graph drawing research area is already well explored [9, 12, 13]. However when it comes to display connections among several mutual dependent graphs it gets even more challenging. For theses purposes we implemented a framework that tries to handle these issues. The results are shown on the basis of test graphs from three different fields of application. In this paper we rather focus on visualization and interaction of graphs than on the process of creation, layouting and editing those.

2 Methods

Each application domain requires a set of well established visualization methods to be successful. The planning of a visualization system highly depends on the used data. In most cases it is hard to determine which combination of visualization methods suits best for a particular data domain. In the upcoming section we will first describe the data on which our system is based and the applied visualization techniques.

2.1 Underlying data

We focus on partially cyclic directed graphs. Graphs consist of several basic node types. Graphs can contain other graphs and users may wants to switch between them. Multiple occurrence of nodes in the same graph as well as in foreign graphs are possible. Nodes and edges can have factors of influence and are therefore parameterized. We have to handle directed as well as undirected edges. In addition data can be hierarchically organized which extends the complexity by data abstraction.

Graphs are often layouted by experts from that particular domain because a lot of meta-knowledge might be needed to position nodes and route edges. The alignment and positioning of captions is problematic as well. In the last decades a lot of effort has been put in automated graph layouting algorithms [3, 4, 5, 15]. However in some cases it still gives better results to carry out the layouting by hand.

Our framework is capable of all kind of data that fits this data definition. Fields of application are manifold. We have chosen sample datasets from different domains that imply miscelaneous properies.

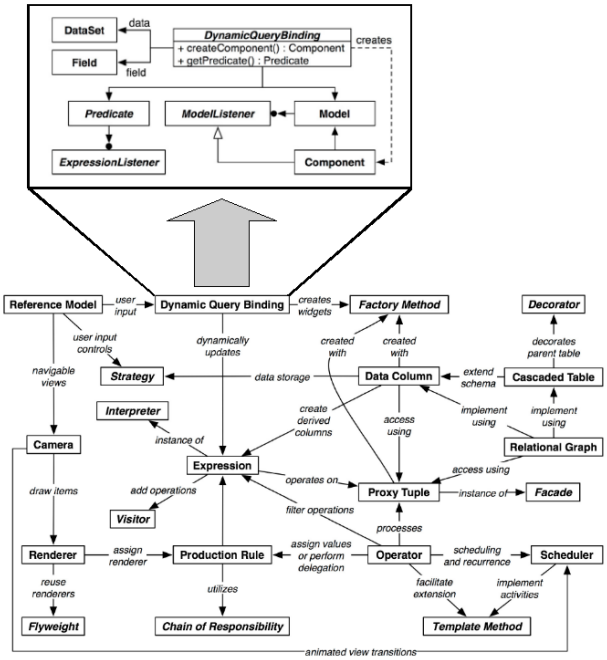

Figure 1: Sample graph comprises dependencies of a set of abstracted software design patterns used for information visualization [1]).

*mstreit@icg.tugraz.at

Software design graphs (Type A)

The sample dataset is a network of interacting software design patterns used in the field of information visualization [1]. Figure 1 shows a modified version of the abstracted graph from [1] figure 14 that contains the overall relationships. In addition each design pattern can be examined in more detail by looking at the subgraph. The interaction of the design patterns are visualized by directed relations. Elements of detailed design pattern graphs are contained in other design pattern subgraphs as well. Therefore the data are hierarchically directed graphs with inter-relationships.

Biomedical networks (Type B)

The biomedical graph consists of three node types that are connected by directed and undirected edges. Figure 2 shows a portion of a sample graph from the biomedial domain.

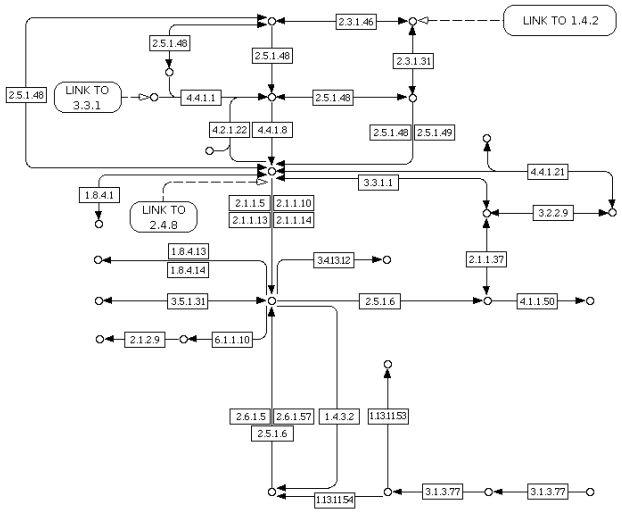

Figure 2: Sample of a biomedical graph. Circular nodes are alternating connected with rectangular nodes. Rounded rectangles denote links to related graphs. Edges are directed as well as undirected.

2.2 Visualization methods

Multiple views widen the space for displaying data and keep a great potential to provide different aspects of the same data at a time [2, 8]. In the upcoming sections various information visualization methods are discussed and the usage in conjunction with our particular data is considered as well.

2-D vs. 3-D

The presentation of data in 2-D and 3-D in parallel can increase the user's ability to percept the given information. It is important to benefit from the advantages of one visualization technique and compensate the weak points with other methods. This approach can lead to a powerful visualization tool. For example 2-D views lack of space for displaying huge networks but are perfectly suited for presenting plain graphs and editing them. When it comes to show relations to other graphs 2-D views reach their limits. A very simple use case that underlines this fact is the visualization of identical nodes or somehow related nodes in other graphs inside the network. For this purpose in a 3-D view several graphs can be arbitrary positioned in space where it is much easier to visualize these relations. On the other hand 3-D views imply problems such as depth perception and occlusion. Occlusion and depth perception are inherent problems of three dimensional representations [8, 10]. There are ways to diminish these problems. For example the usage of blending can counteract the occlusion problem.

Linking and Brushing

The need of interactive connection of multiple views lends itself to the employment of the linking and brushing technique (L&B). Brushing is a method where a subset of the data can be interactively selected using so called brushes. Linking is the procedure of highlighting the selected data portion in related views by for example changing the color and/or the shape [8, 10]. Assuming that view A and view B are displaying the same data but using a different way of representation all changes in view A need to be published to view B immediately and vice versa.

Focus + Context

The Focus + Context approach aims on providing detailed information and an overview over the data simultaneously [10]. Kosara and Hauser classify the F+C techniques in four subgroups [10]:

- Distortion-oriented
- Overview method
- Filtering
- In-Place Techniques

In this paper we want to depict the overview and the filtering technique. The overview method provides contextual information in a separate view. The filtering technique shows the information using the same space but in a different representation or by filtering portions of the data.

Detail on demand

Meta-data about the graph and its elements is essentially needed by the user to explore and reveal information. Detail on demand is a method that provides information by user request. There are various strategies for providing contextual information which depends on the amount. For instance in a graph some kind of identification number is used for the caption of a vertex. However triggering a mouse-event signals the user's interest in the full name of the node. In this case a plain pop-up tool-tip window

might be appropriate. In contrast if the context information is more comprehensive a sort of data explorer in a separate view might be a better solution.

Neighborhood visualization

Visualizing neighborhoods in graphs means that after selection of a node all adjacent nodes need to be highlighted. Neighborhood algorithms are possible in arbitrary depth and can either consider edge directions or not. After picking a node by triggering a mouse-event the neighborhood algorithm returns a subset of nodes which can then be highlighted according to the distance of the root node (e.g. by changing their color). In an environment with multiple views it is important that the result is synchronized with all views that operate on the same graph data.

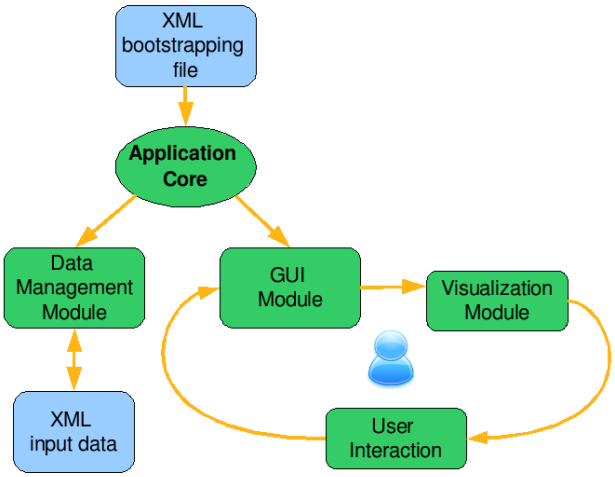

Figure 3: Overall module design

3 Framework implementation

The prototype application is written in Java. It is divided in several seperated sub-modules. Figure 3 shows the framwork design. The core application triggers the bootstrapping. The XML bootrstrapping file contains information about how the Graphical User Inface should be built up, which data should be initially loaded during start-up and how the views are supposed to be connected together. After the start-up process is finished the core module passes on the control to the user. At that point the user is in the center of an interaction loop with the visualization module. On user's request new data can be reloaded to the system during run-time.

3.1 Detailed system architecture

The design of the framework follows the Model-View-Controller (MVC) design pattern [1, 6, 11]. According to the MVC metaphor the model that is the data is separated

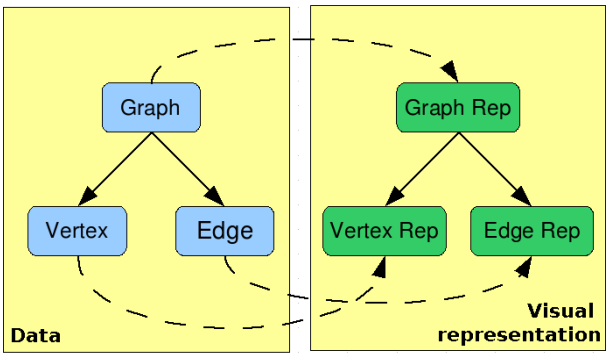

Figure 4: Model abstraction into data and visual representations

from the views. This strict distinction between model and view allows several views to access same data at a time. When in a view data get changed the dataset can be updated and other views that operate on the same data can perform a refresh without knowing each other. The data in the model part can be further split up into actual data and visual models of the data [1, 16]. The visual model can be seen as a special data representation that extends the data by visual attributes. In case of graph nodes the data itself may consist of an identification number, a label and its relations to other nodes while the visual data representation typically contains attributes like color, shape and layouting information. Hence the usual situation that 2-D graph views need another data representation than an 3-D OpenGL view. In figure 4 this data abstraction is applied to graph data structures.

3.2 Graphical User Interface (GUI)

The Graphical User Interface (GUI) is implemented using the Standard Widget Toolkit (SWT)[1] from the Eclipse Foundation [7]. We used the open source library JGraph[2] for building plain graphs in 2-D. JGraph is a well designed library that fits best our needs regarding graph drawing. For the 2.5-D visualization we used Java OpenGL (JOGL)[3] ([17, 19]) which can be embedded in SWT container widgets without problems. The GUI itself can be fully customized via a XML configuration file. By using XML documents it is possible to combine 2-D and 3-D views in an arbitrary layout inside the framework.

3.3 Data handling

One of our test datasets (Type B) consists of about 400 graphs including meta-data about the graphs. Due to this fact special considerations regarding the data handling are essential.

[1] http://www.eclipse.org/swt
[2] http://www.jgraph.com
[3] http://jogl.dev.java.net

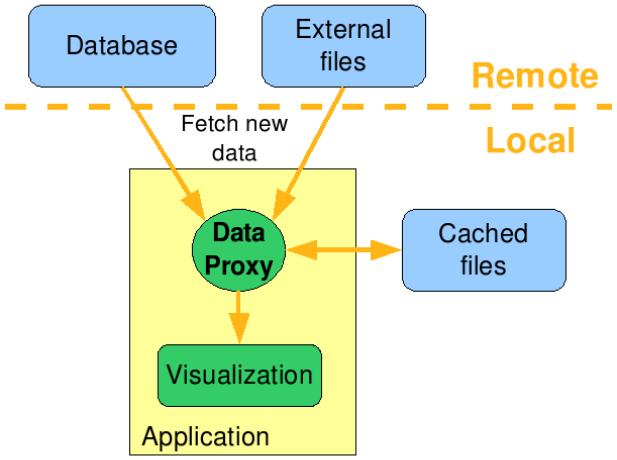

Figure 5: Data block diagram showing the data loading process

Availability vs. performance

Graph data are read from external data files or an online database. The latter is updated on a daily basis by the community and therefore contains up-to-date information. The needed time period to either fetch a whole data snapshot or else to update the local data is not suitable for a real-time environment at all. As a result we had to take care of this fact by integrating a data proxy. Figure 5 shows a data block diagram including the proxy. The data are then stored in an internal file format which can then be used on application startup for pre-fetching the data into the applications memory. When it comes to visualization keeping the whole dataset on short-call in local memory is mandatory because the field of application demands real-time behavior.

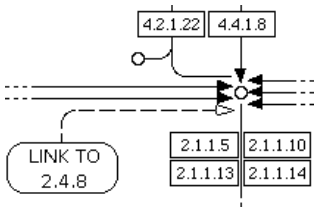

Figure 6: Problematic cut-out of a Type B graph that shows which problems an automated layouting algorithm would have to face.

3.4 Rendering using hierarchial display lists

We packed the rendering of the graphs in display lists in order to increase the performance. Using display lists it is possible to store OpenGL commands for later execution [14, 18]. Specially in the domain of graphs display lists are worth to use because the geometry in graphs is very limited and it must be reused very often. The graphs can be statically built during initialization phase before the rendering loop gets executed. It turned out that our data is perfectly applicable for hierarchical display lists. With hierarchical display lists a cascade of display list can be realized. This concept enables us to build the static graphs completely during the initialization. The drawback of display lists is that they cannot be modified after creation. This facts needs to be considered in applications' design especially when it comes to user interaction and brushing (i.e. modifying geometry) of graph portions.

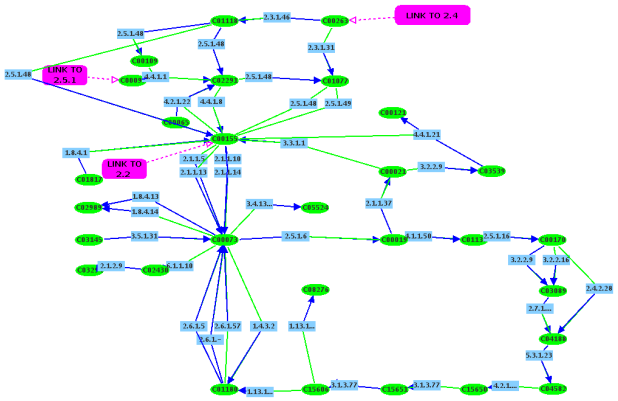

Figure 7: Type B sample graph without any layouting modification.

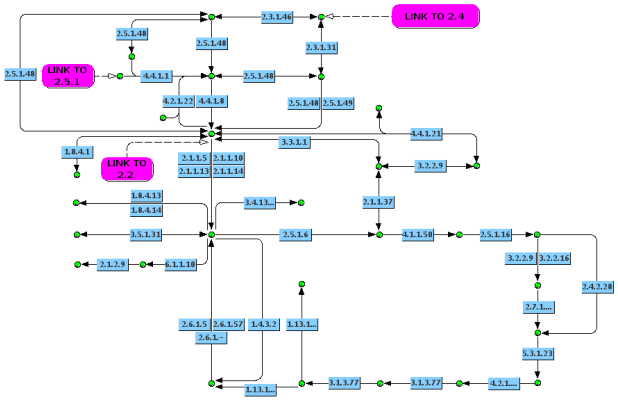

Figure 8: Type B sample graph with background texture overlay.

4 Visualization module

4.1 Texture overlay technique

To overcome the problem of placing and routing the graphs we applied several approaches. For positioning the nodes we utilized a XML file that contains all information

needed to draw nodes and connect them by edges. Also viewing information like screen coordinates of nodes from the handmade layouted graph can be parsed. Therefore we are able to build the graphs exactly as the user community inside our test domain are used to them. This solves the issue of node positioning but not the routing of the edges and positioning of the labels in 2-D.

Figure 6 shows a portion of a graph that contains problematic situations where routing is not trivial. To solve the layouting problem we took the static images of the generated graphs and put them in the background of the graph in our system. This approach enables us to present the graphs in the way the users are used to while extending the static graphs with the power of all visualization and interaction possibilities in 2-D and 3-D. In 2-D we work with the images while in OpenGL the images are loaded as textures. Figure 7 shows the sample graph without any modification after parsing from XML file and drawn by using the JGraph library. Whereas in figure 8 the same graph with a background overlay is shown. In this case the visualization of the system internal edges is turned off. As a consequence the user just sees the edge lines that come from the well-known background image.

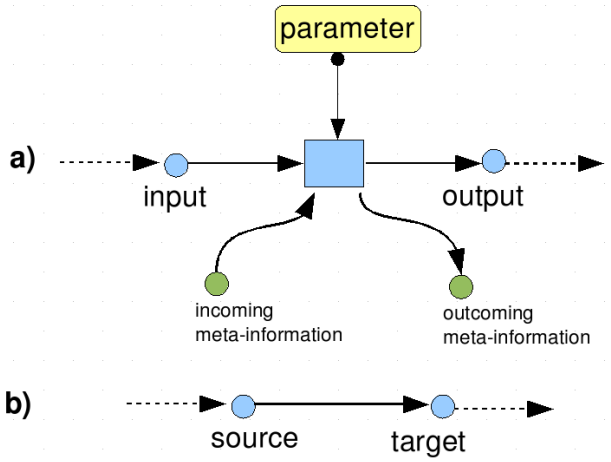

Figure 9: **a)** shows the original graph portion while figure **b)** shows the same data after application of the F+C filtering

4.2 Deployment of F+C techniques

The filtering method is realized by removing a class of nodes. One of the two basic node types is just a intermediate state. This circumstance is illustrated in figure 9. Think of a chemical reaction where each reaction needs a substrate and a product. The user could only be interested in the products and/or substrates and not in the chemical reaction itself. This would be a perfect case for filtering the reaction nodes. Another example are data flow diagrams. The actions might be removed and only the data entities should be visible.

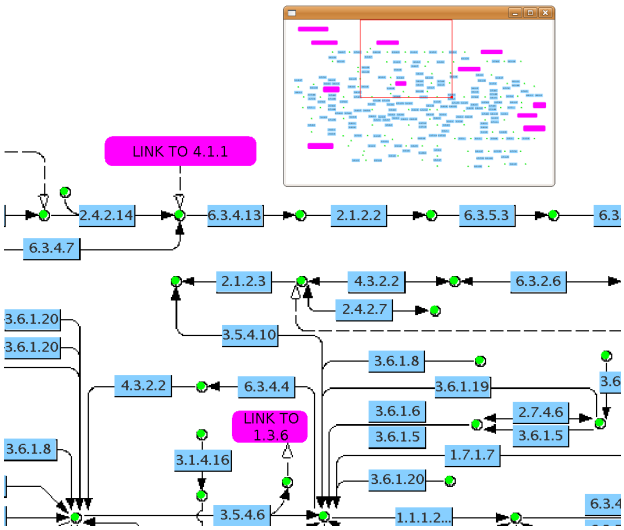

Figure 10: Portion of a sample graph that can be interactively selected in the overview map.

As a second F+C technique we implemented the overview method. A separate view (in our case a new window) contains a overview map in which an area can be depicted that is supposed to be visible in the graph displaying view. An example is shown in figure 10.

4.3 Neighborhood visualization

The system is designed to perform k-depth neighborhood visualization. Neighborhoods up to a distance of 3 seems to be the limit within the algorithm makes sense because of the cyclic characteristic of the graphs. Figure 11 shows a portion of the sample graph on which a 3-times neighborhood algorithm is performed. The used color coding indicates the selected node in red and adjacent nodes from a dark orange to yellow. In the prototype implementation we used a Breadth-first-search (BFS) algorithm. Interesting effects are caused by the cycles inside the graph. It is possible that nodes are visited several times by the algorithm. To consider these cycles we have to remember already visited nodes during running the BFS.

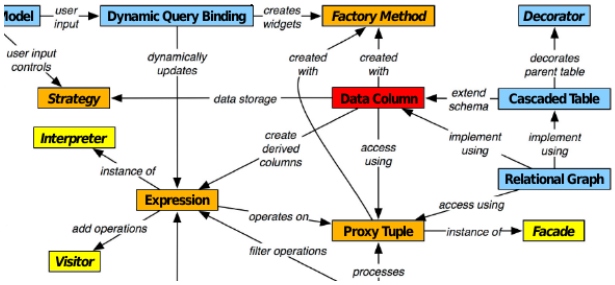

Figure 11: Depth 2 neighborhood visualization in a Type A graph

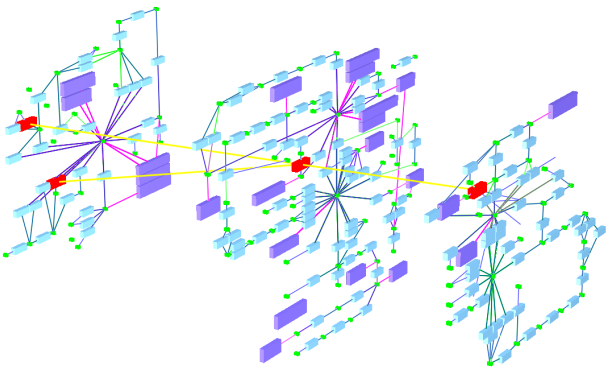

Figure 12: 2.5-D OpenGL layered Type B graph with a user selection

4.4 Layered graph view

We annotate several dependent graphs on different levels in 3-D space. This gives the user the ability to identify identical nodes in foreign graphs. The highlighted nodes change their color and are additionally accentuated by a pulsing effect. Figure 12 contains a screenshot of 3 graphs in a layered view. The graph layers are see-through. Highlighted nodes are connected by a linking line. A more sophisticated relation visualization is eligible.

The perception of the layers can be supported by adding textures of the handmade graph on demand. A screenshot of the layered view including the texture overlay is provided in figure 13. By implementing the textures as semi-transparent layers the occlusion effect is alleviated.

5 Conclusion and Future Work

In this paper we propose a framework that enables users to reveal dependencies and connections in complex graphs that are hidden under the surface. However the assembly of the network in small more or less separated graphs is only a makeshift because of the lack of a good visualization method. Better results could be achieved by implementing a real 3-D solution where node are positioned in space. The user would need time to get used to the new layout and different way of navigation. But the preparation of the data in 3-D space is more likely to the nature of graphs than the artificial split-up in subgraphs.

6 Acknowledgements

The author wants to express his gratitude to Michael Kalkusch for his supervision and many fruitful discussions. Futhermore many thanks to Dieter Schmalstieg for giving me the opportunity to perform research on this interesting topic.
This research was sponsored in part by Zukunftsfond Steiermark.

References

[1] Maneesh Agrawala. Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):853–860, 2006. Student Member-Jeffrey Heer.

[2] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. Guidelines for using multiple views in information visualization. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, pages 110–119, New York, NY, USA, 2000. ACM Press.

[3] Giuseppe Di Battista, Ashim Garg, and Giuseppe Liotta. An experimental comparison of three graph drawing algorithms (extended abstract). In *SCG '95: Proceedings of the eleventh annual symposium on Computational geometry*, pages 306–315, New York, NY, USA, 1995. ACM Press.

[4] Giuseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7(5-6):303–325, 1997.

[5] Josep Diaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[7] Bobby Harris, Rob Warner, and Robert Harris. *The Definitive Guide to Swt and Jface*. APress, 2004.

[8] Helwig Hauser and Robert Kosara. Interactive analysis of high-dimensional data using visualization. In *Workshop on Robustness for High-dimensional Data (RobHD 2004)*, Vorau, Austria, May 2004.

[9] Patrick Healy and Nikola S. Nikolov, editors. *Graph Drawing, 13th International Symposium, GD 2005, Limerick, Ireland, September 12-14, 2005, Revised Papers*, volume 3843 of *Lecture Notes in Computer Science*. Springer, 2006.

[10] Robert Kosara, Helwig Hauser, and Donna L. Gresh. An interaction view on information visualization. In *State-of-the-Art Proceedings of EUROGRAPHICS 2003 (EG 2003)*, pages 123–137, 2003.

[11] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, 1988.

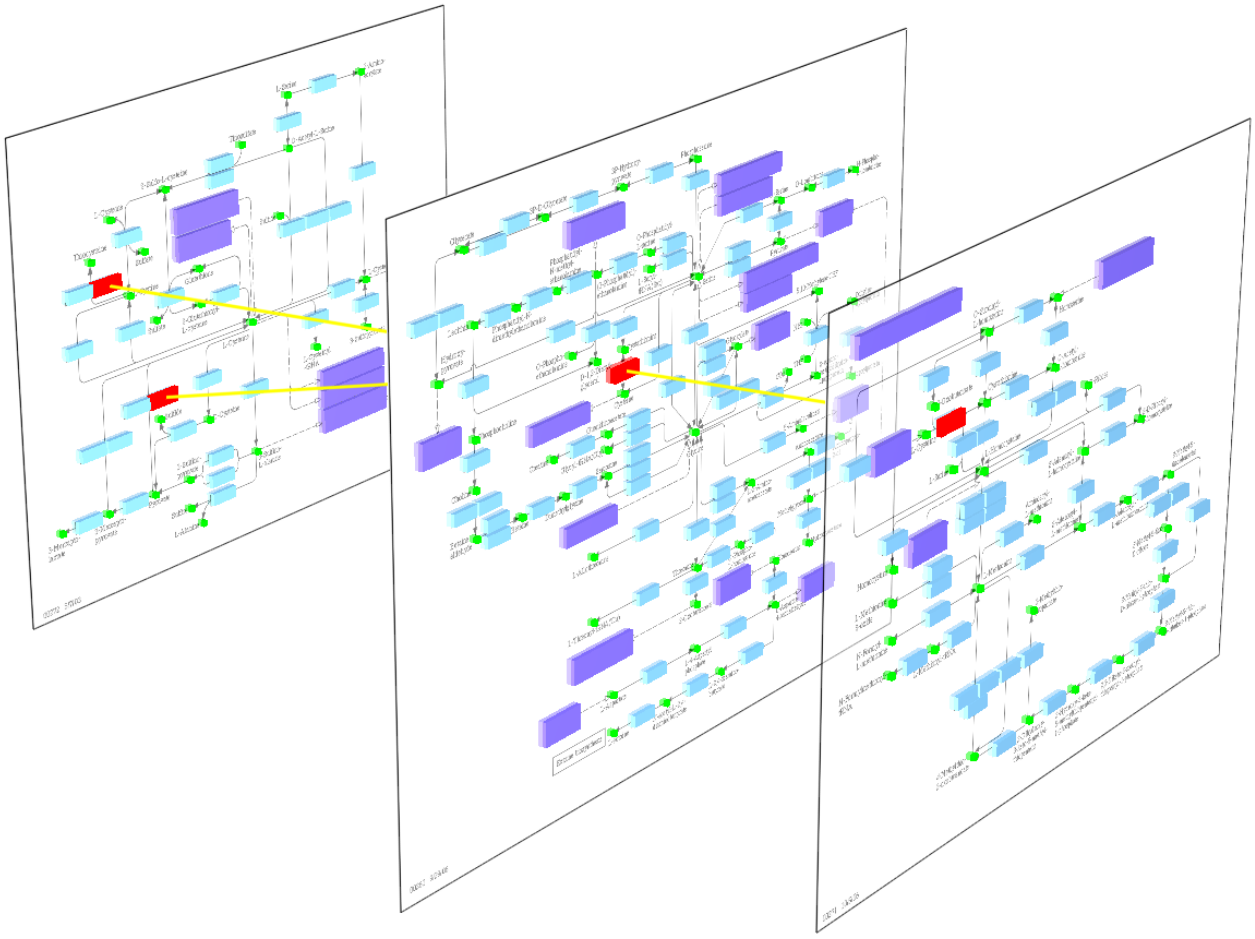

Figure 13: 2.5-D OpenGL layered Type B graph with a blended texture overlay. The node in the middle layer is picked by a mouse-click event. Identical nodes in dependent graphs are interactively highlighted and connected to the selected one.

[12] Giuseppe Liotta, editor. *Graph Drawing, 11th International Symposium, GD 2003, Perugia, Italy, September 21-24, 2003, Revised Papers*, volume 2912 of *Lecture Notes in Computer Science*. Springer, 2004.

[13] János Pach, editor. *Graph Drawing, 12th International Symposium, GD 2004, New York, NY, USA, September 29 - October 2, 2004, Revised Selected Papers*, volume 3383 of *Lecture Notes in Computer Science*. Springer, 2004.

[14] Dave Schreiner, Dave (Ed.) Schreiner, and Dave Shreiner. *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[15] Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, 18(1):61–79, 1988.

[16] Diane Tang, Chris Stolte, and Robert Bosch. Design choices when architecting visualizations. *Information Visualization*, 3(2):65–79, 2004.

[17] David Wolff. Using opengl in java with jogl. *Journal of Computing Sciences in Colleges*, 21(1):223–224, 2005.

[18] Mason Woo, Davis, and Mary Beth Sheridan. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[19] Zhigen Xu, Yusong Yan, and Jim X. Chen. Opengl programming in java. *Computing in Science and Engg.*, 7(1):51–55, 2005.