INTERACTIVE VISUALIZATION OF ABSTRACT DATA

Martin ŠPERKA, Peter KAPEC

Abstract: Information visualization is a large research area. Currently with more powerful computers and graphic accelerators more and more visualization techniques become part of daily use. In this paper we discus visualization of abstract data – data that is difficult or impossible to manually grasp. Using visualization of abstract data we can gain better insight. We present several experimental methods for visualizing graphs and show possible applications in the software visualization field.

Keywords: Information visualization, visual data mining, software visualization.

1 INTRODUCTION

With the exponential growth of data caused by the penetration of information and communication technologies to every branch of human activities, grows the demand on extracting facts and knowledge from the flood of these data by means of business intelligence and data mining. Visual data mining is the method which is often used in many fields by using different data and science visualization methods and tools. Special category is visualization abstract data in form of graphs.

Abstract data visualization is used in different application areas as organizational structures, family trees, geographical data, transport, communication and social networks, ontologies, information systems – hardware interconnection, software, distributed and collaborative systems and modelling (for example UML).

The most cited visual investigation technique, and not only related to abstract data, is the visualization seeking mantra - *Overview first, zoom, filter, and then focus details-on demand* [22]. Users first see data in general view, then select subset of his/her interest and finally focus on particular attributes of selected data objects. Visualization mantra is an interactive process requiring fast response of computer, which is especially in case of large data sets not trivial. However new multi-core processors and GPU allows to render complex scenes and new forms of interactive visualization are possible. In this paper we discus the visualization process, show several experimental methods of graph visualizations and present applications in software visualization.

2 INFORMATION VISUALIZATION

The visualization process is a transformation of data in one representation to another, mostly to a representation better observable by humans. The following steps of a visualization process can be found in any problem area [21]:

- data preparation;
- encoding;
- presentation and interaction.

Preparation, the first step of the visualization process, is used to identify relevant entities and events that the visualization will deal with.

The second step encoding deals with problems how the data will be displayed. The questions to be considered are oriented to efficiency, aesthetic, understanding, similarity etc. These aspects of visualization play major role for humans that will be involved in the visualization process, because when the visualization is not understandable or uses non-standard visuals, the benefits of visualization may get lost.

The final phase is presentation and interaction and should answer questions about how the visualization objects are displayed and which interaction possibilities are offered to the user. For each specific visualization the vocabulary used is also important. We can identify many aspects for graphical elements of a vocabulary [16].

Visual exploration techniques can be classified according to three orthogonal criteria [13]:

- data to be visualized;
- the visualization technique;
- interaction and distortion technique used.

Data type may be one-, two- and even n-dimensional data, text and hypertext, hierarchies and graphs, temporal data and recently also algorithms and software. Visualization techniques may be classified into standard 2D/3D displays, geometrically transformed displays, icon-based displays, dense pixel displays and stacked displays. Interaction and distortion techniques may be classified into interactive projection, filtering, zooming and distortion techniques

These techniques can be freely combined to form new exploration techniques suitable for specific applications.

3 VISUALIZING GRAPH STRUCTURES

Visualizing graphs is not trivial and involves several theoretical and practical problems. The main problem is the size and density of the visualized graph – when denoting the number of nodes as |N|

and the number of edges as |E|, we can categorize graphs into [16]:

- sparse |E| <= |V|
- normal |V| < |E| <= 3|V|
- dense |E| > 3|V|

The graph size directly affects scalability of graph layout algorithms [2] – they often work for small/sparse graphs but are to slow to be usable or even fail to finish the layout when applied to large/dense graphs. To lower the graph size we can layout only the spanning tree of a graph, ignore edges with edge weight lower than some limit or create clusters.

The problem of display area is also related to graph size and graph layout algorithms. Displaying the whole graph in a limited display area is often not very comprehensible or even technically not possible. This problem can be solved using appropriate visualization technique, e.g. *focus+context* or *distortion techniques.*

Another problem is related to incremental and dynamic changes in graph structures often caused by user interaction. The graph layout algorithm should be capable to handle local graph modifications without the need to modify already finished graph layouts – force-based layout algorithms are very suitable for these cases.

Moving into 3D space may offer "more" space for visualization, however it may introduce more problems. Typically objects in 3D space overlap after projection making them difficult to observe. This can be solved by finding best views in which aesthetic criteria are fulfilled. The most obvious problem is that users often look at 2D projections of 3D space, thus interaction and navigation becomes more complex. The current trend of using 3D displays may provide better insight into visualizations, but effective user interfaces are still an open challenge.

3.1 Criteria for graph visualization

An important criterion for understanding visual information is ordering elements. Basic aesthetic principles and rules for visualization of graphs, according to [1], are related to

- positioning of nodes:
  - balanced node placement - symmetry
  - not overlapping nodes
  - related nodes create clusters
  - nodes are not to close to edges
  - maximize node orthogonality
- edge placement:
  - minimize edge crossing and bending
  - equal edge lengths
  - maximize angles between edges
  - maximize edge orthogonality
- whole graph layout:
  - maximize graph global and local symmetry
  - minimize layout area
  - adjust layout area to display area

In case of trees there are additional rules

- nodes at the same level should be on same horizontal lines
- equal distance among node's children

Several of these criteria are related and can be applied together, but some conflict, thus when developing a graph layout algorithm we have to follow aesthetic criteria that will provide best graph layout for specific applications. Of course there are specific requirements for different graphs and applications. One of them is to minimize the graph area.

Another requirement is predictability; it means that when we render two topologically similar graphs, the results should not be very different. In case of different results, the user could be confused – the observer's mental map would become deformed [17]. In general not all rules and aesthetic criteria can be fulfilled in case of especially large graphs. Interactive 3D graphics and virtual reality helps to overcome the difficulty in visualizing large data sets.

3.2 Graph layout methods

Graph layout algorithm can be categorized into two categories: *deterministic* and *non-deterministic*.

Deterministic layout algorithms use exact equations to place nodes. Typical examples of this approach are layered/hierarchical views [9], Reingold-Tilford's views [19], cone trees [20] and radial views [9], tree-maps [10] etc.

Non-deterministic graph layout algorithms use some physical model in which nodes are positioned by applying forces and their final position is reached when the whole system reaches minimal energy state. These force-based methods consist of a model and algorithm. The model defines graph nodes as physical objects that react on forces. The algorithm then iteratively reassigns node positions according to forces until an equilibrium state is reached. Force-based layout methods have many advantages:

- easily implementable
- very parametrizable
- modifiable by adding new forces
- effective for small graphs
- produce symmetrical layouts
- animation of layout preserves mental map
- easily expendable into 3D

Disadvantages are:

- slow for large graphs
- final layout is not predictable

The first force-based layout method was developed by Eades [6] in which adjacent nodes are attracted by spring forces defined by edges and nodes are repelled when no edge connects them. Various modifications of this approach have been developed [12], each scaling better for larger graphs. Recently implementations of force-based layout algorithms on GPUs allow to layout very large graphs in very short time [7].

3.3 Example graph visualizations

In one project we deal with using virtual reality in graph visualization [3]. The goal of this project was to try to increase the visual perception and help to discover facts about mutual connections between different trees, representing hierarchical structures for example companies, families or clans. Trees are displayed on different planar and semitransparent layers as displayed in Figure 1.

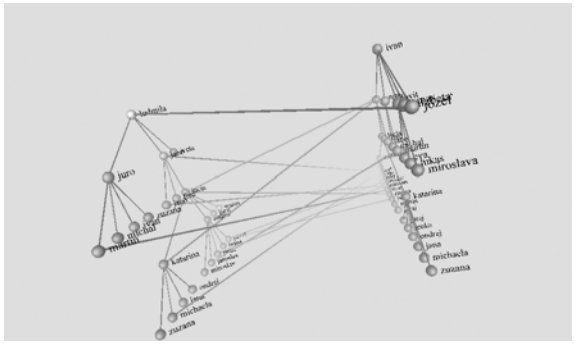

Figure 1 Trees displayed on two layers

Observer can move in Z direction (forward and back) and see different trees individually. It is possible to zoom and see overview or details of the tree. The layer can be opaque and then no other layers (trees) are visible or transparent so other layers are visible and the context with selected tree is observable. The transparency is arbitrary adjustable by the formula $I = I_a * alpha + I_r * (1 - alpha)$ where I_a is luminance of selected layer, I_r is luminance of other layers and alpha is variable in the interval <0,1> . Depth cueing (layered fog) helps to increase visual separation of different trees. When the user wants to see connections between different trees the view is rotated 90 degrees around X or Y axis and different trees are then represented as horizontal lines or vertical lines. The view can be rotated about all axes in arbitrary angle so different axonometric and perspective projections can be used for visual inspection. This tool was implemented in

Java language and the resulting trees are exported to VRML 97 language.

In another project we experimented with a metaphorical visualization of graphs that displays graphs as soap bubble clusters [24]. Nodes are displayed as soap bubbles and edges as thin soap membrane between adjacent nodes. The visualization is expected to produce rather dense cluster, therefore actual edges are displayed as straight lines between soap bubble centres. The bubble layout is based on force-directed placement as described in section 3.2. An example graph visualization using soap bubble metaphor is shown in Figure 2.

To create thin bubble membranes between adjacent bubbles we used the approach by Sunkel et al [23]: firstly intersection planes of colliding bubbles are identified and afterwards vertices of a sphere exceeding into another sphere are projected onto the intersection plane along the membrane normal. To accelerate membrane creation these calculations were implemented as vertex shaders.

Due to the requirements of real-time visualization the visual properties of real soap bubbles are limited to simulating the Fresnel effect, interference and environment reflection and are based on existing approaches [8].

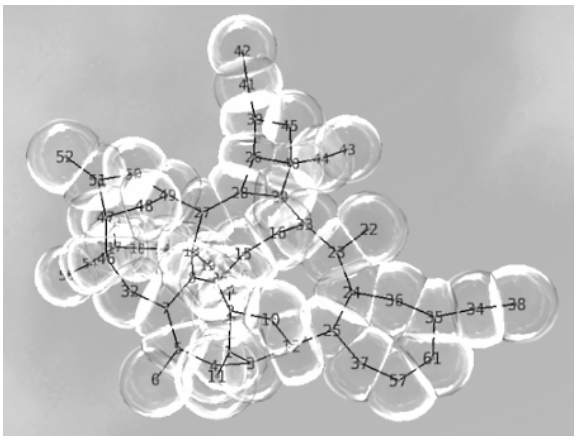

Figure 2 Graph visualized as soap bubble cluster

One problem, which is also in focus of our research, is 3D presentation of trees with perspective stereoscopic projection [18]. Stereo pair of images can be viewed on the desktop computer with the synchronized shutter glasses or red and cyan colours filters. Several VRML or X3D plug-in viewers have this option. Figure 3 illustrates results of an experimental application using WEB3D technology with stereo output – visualization of the binary tree showing results from the ice hockey championship.

Figure 3 Stereo pair of interactive binary tree visualization in 3D space

To accelerate the overview-zoom-detail cycle process we explored methods that map 2D images onto a sphere. The sphere can be interactively rotated and zoomed, allowing to see the whole graph and its details without loosing context. When looking from the outside of the sphere, projection is similar to the fish eye projection, where the observer can concentrate on details of the object at the centre. Distant objects are scaled down and situated at the periphery of the sphere. When looking from inside of the sphere, the centre is far from the viewer and objects on the periphery are bigger. Mapping image stereo pairs on the sphere is one form of visual exploration in an immersive virtual environment – half sphere dome or spherical CAVE. Figure 4 shows graph seen from the inside and outside of the sphere.

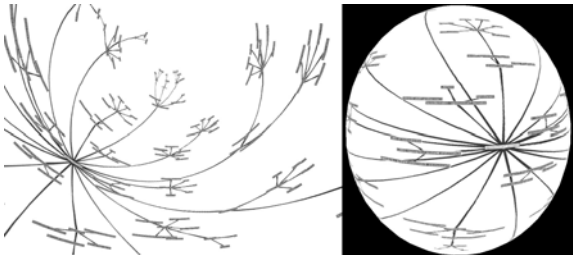

Figure 4 Objects seen from the inside (left) and outside (right) of the sphere

Graph visualizations have found many applications in many areas. Our research applies known and experimental graph visualizations in the software visualization field.

4 SOFTWARE VISUALISATION

Software can be considered as a special data type that is very suitable for visualization. The intangibility of software components makes it very difficult to comprehend all aspects of software systems – especially today when we look at enterprise systems and their increasing complexity. Software is not only source code, but consist of many artefacts including data, algorithms, documentations, user interfaces etc. and all possible documents related to software development. These software artefacts occur in the whole development process. Graphs and their visualizations are often used in the software visualization field.

Software visualization can focus on three main aspects: *software structure*, *behaviour of executing processes* and *evolution of software development* [4]. In our experiments we focused on visualizing structure of existing software systems. To visualize class inheritance we implemented a modified cone-tree visualization that displays namespaces, classes and their inheritance on layered circles [14]. For each class inheritance relations are shown as links to higher layers. Methods and attributes of classes are positioned on the outside of each cone. A visualization of an existing system containing nearly two thousand software artefacts is shown on Figure 5.

For visual data-mining the implemented visualization system allows to filter out not important elements and to sort them according to user preferences.

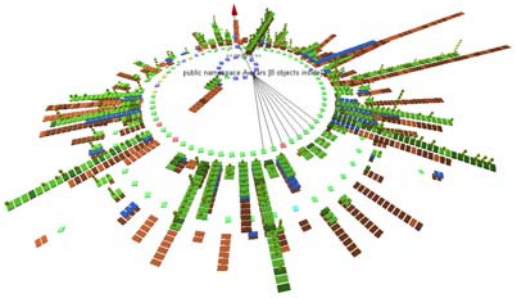

Figure 5 Visualization of an existing system

The Figure 6 displays the visualization of the same system, but with only classes and their methods and sorted by the decreasing number of methods a class has.

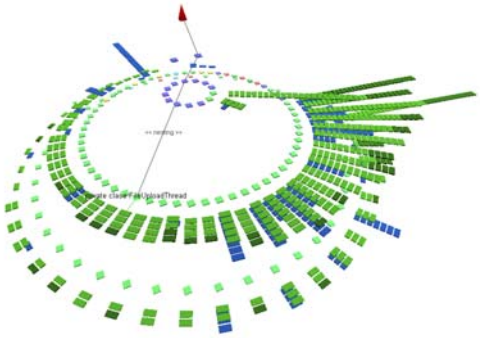

Figure 6 Sorted view of classes and their methods

To represent relations between various types of software artefacts we developed a hypergraph-based model that allows to store relations between semantically different software artefacts [11]. The hypergraph model uses hyperedges that allow to connect more than two nodes with one hyperedge.

This allows storing complicated relations between artefacts with one relation for which standard graphs would need several edges. A typical example of such relations can be found in a call-graph that represents calling relations between functions or methods. Functions can be represented by hypergraph nodes. Using one hyperedge we can connect all those functions that are called by the function of our interest. Based on this hypergraph model we developed a software visualization system that utilizes hypergraphs in the whole visualization process: hypergraphs are used to store software artefacts and their relations, they are used to query and filter the analysed software systems and finally are used in 3D visualization. Force-based graph layout algorithms can be used also for hypergraphs due to a mathematical transformation of hypergraphs into bipartite graphs.

A hypergraph-based visualization of a small open-source system is shown in Figure 7. The visualization shows nearly the whole extracted hypergraph containing more than thousand software artefacts, mostly source code elements and related documentations, and more than five hundred relations related to these artefacts.

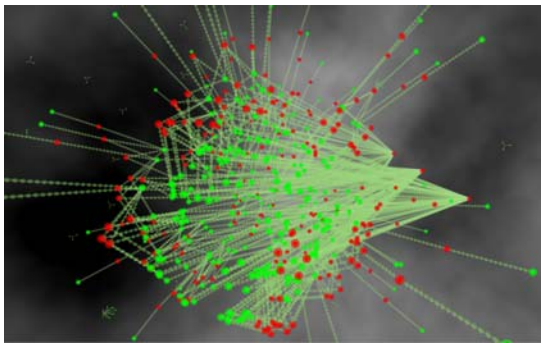

Figure 7 Hypergraph visualization of an existing open-source system

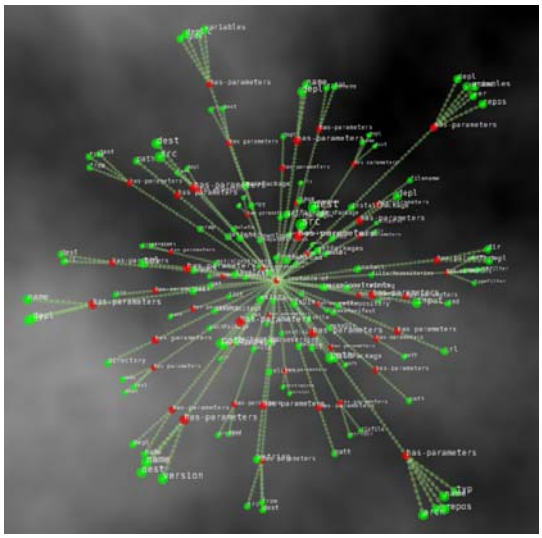

Figure 8 Results of a hypergraph query

The hypergraph model allows us to look at software artefacts as knowledge repository with query functionality. A hypergraph representation of software can be queried by hypergraph queries – the results of these queries are also hypergraphs, thus making this concept transparent. A result of a hypergraph query applied to the whole hypergraph representation of the system in our study is shown in Figure 8. The query extracted all functions, their parameters and return values and related documentation from the whole hypergraph.

Dynamic aspects of software visualization are also in our focus. We developed a visualization system that captures the execution of JAVA programs and stores it for post-mortem analysis. This record is then used for visualization in a separate application that allows to playback this record, fast-forward playback or jump to specific time. The record contains all important information suitable for debugging and analysis of executed programs. The user can step the animation forward and also backward, thus providing advanced debug functionality not available in current development environments. Figure 9 displays a snapshot showing a class in figure centre and living instances of this class at a specific moment during program execution. Important events during program execution are highlighted.

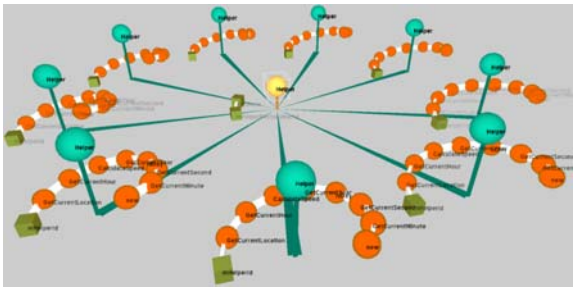

Figure 9 Class and instances during program execution

All presented experimental software visualization systems use 3D space for visualization in which users can interactively navigate using a virtual camera. The virtual camera we use allows fly-mode for free graph exploration and also orbiting around selected nodes. This way the user can zoom to nodes of interest and by orbital movements explore relations originating from selected node.

5 CONCLUSION

In this paper we presented several experimental graph layout algorithms. Our main research is however more focused on software visualization where we focus on structural and behavioural aspects of software systems. For this purpose we developed several visualization systems that display software artefacts and their relations as graphs or

hypergraphs. Proper evaluation of these visualization systems in practice is currently in our main focus.

Future work will be oriented on more effective layout algorithms capable to layout very large graphs containing hundred-thousands of nodes and (hyper)edges, thus allowing to visually analyse enterprise size software systems. Also work is oriented to provide not only visualizations for analysis, but also provide a visual programming environment that seemingly integrates hypergraph visualizations with textual programming in 3D virtual environment.

This work was partially supported by the grant KEGA 244-022STU-4/2010: Support for Parallel and Distributed Computing Education. We would like to thank master degree student Ivan Ruttkay-Nedecký for developing the program runtime visualization system.

References

[1] BENNETT, C., RYALL, J., SPALTEHOLZ, L., GOOCH, A..: The aesthetics of graph visualization. In: *Computational Aesthetics*, pages 57–64. Eurographics Association, 2007. ISBN 978-3-905673-43-2.

[2] CHEN, C.: *Information Visualization*, chapter 3: Graph Drawing Algorithms. Springer, 2004. ISBN 978-1-84628-340-6.

[3] ČÍKOVÁ, Z.: *Interactive Visualization of Large Graphs with Virtual Reality*. Bachelor thesis. FIIT, Slovak University of Technology (supervisor M. Šperka), Bratislava, 2009.

[4] DIEHL, S.: *Software Visualization: Visualizing the Structure,* Behaviour, and Evolution of Software, 1 ed. Springer, May.

[5] EADES, P.: Drawing Free Trees. In *Bulletin of the Institute for Combinatorics and its Applications*, 1992, s. 10– 36.

[6] EADES, P.:. *A heuristic for graph drawing. Congressus Numerantium*. 1984, vol. 42, is. 1, s. 149-160.

[7] FRISHMAN, TAL.: *Multi-level graph layout on the GPU*. IEEE transactions on visualization and computer graphics, vol. 13, no. 6, 2007.

[8] GLASSNER, A.: *Soap Bubbles: Part2. IEEE Computer Graphics and Applications*, (2000), vol. 20, no. 6, pp. 99-109.

[9] HERMAN, I. et al.: Graph *Visualization and Navigation in Information Visualization*. 2000.

[10] JOHNSON, B., SHNEIDERMAN, B.: Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures. In *Proceedings of the 2nd conference on Visualization (Vis'91)*, pp. 284–291, 1991.

[11] KAPEC, P.: *Hypergraph-based software visualization, International Workshop on Computer Graphics*, Computer Vision and Mathematics, GraVisMa'09, pp 149-153, 2009

[12] KAUFMANN, M., WAGNER, D.: *Drawing Graphs: Methods and Models.* 2001.

[13] KEIM, D. A.: *Information Visualization and Visual Data Mining*, IEEE Transactions on visualization and computer graphics, vol. 7, no. 1, pp 100-107, 2002.

[14] KOMOROVSKÝ, F.: *Visual data-mining in software, Master thesis.* FIIT, Slovak University of Technology, Bratislava, (supervisor P. Kapec) 2009.

[15] MACKINLAY, J.: *Automating the design of graphical presentations*. ACM Transactions on Graphics, 5(2):110–141, 1986.

[16] PAJNTAR, B.*: Overview of algorithms for graph drawing.* Conference on Data Mining and Data Warehouses. 2006.

[17] PARKER, G., FRANCK, G., WARE, C.: *Visualization of Large Nested Graphs in 3D: Navigation and Interaction,* Journal of Visual Languages and Computing, vol. 9, pp 299-317, 1998.

[18] POZOR, M.: *Rendering Graphs in Virtual Reality Environment*. Master thesis. FIIT, Slovak University of Technology Bratislava, (supervisor M. Šperka) 2006

[19] REINGOLD, E. M., Tilford. J. S.: Tidier *Drawing Of Trees*, IEEE Transactions on Software Engineering, 1981, Vol. SE-7, No. 2, pp. 223-238.

[20] ROBERTSON, G. G., MACKINLAY, J. D., CARD, S. K.: Cone trees: Animated 3D visualizations of hierarchical information. In *Proceedings of ACM Conference on Human Factors in Computing Systems.* 1991. s. 189-194.

[21] SCHUMANN, H., MULLER, W.: *Visualisierung: Grundlagen und allgemeine Methoden. Springer-Verlag*, Berlin, 2000.

[22] SHNEIDERMAN, B., PLAISANT, C.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (4th Edition). Pearson Addison Wesley (2004).

[23] SUNKEL, M., KAUTZ, J., Seidel, H. P.: Rendering and Simulation of Liquid Foams In: *Proceedings of the Vision, Modelling and Visualization*, (2004), pp. 285-294.

[24] UKROP, Ľ.: Visualizing graph structures using soap bubbles. In: *IIT.SRC: 6th Proceedings in Informatics and Information Technologies*, STU

in Bratislava FIIT, vol. 2, pp 467-474, 2010.
ISBN 978-80-227-3266-3.

Assoc. Prof. Dipl. Eng. Martin ŠPERKA, PhD.
Faculty of Informatics
Bratislava School of Law
Tematínska 10
851 05 Bratislava
Slovak Republic
E-mail: martin.sperka@uninova.sk

MSc. Peter KAPEC, PhD. student
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 3
842 16 Bratislava
Slovak Republic
E-mail: kapec@fiit.stuba.sk