Fisheye Tree Views and Lenses for Graph Visualization

Conference Paper · August 2006

DOI: 10.1109/IV.2006.54 · Source: IEEE Xplore

4 authors, including:

James Abello
Rutgers, The State University of New Jersey
94 PUBLICATIONS **2,347** CITATIONS
SEE PROFILE

Frank E Ham
Stanford University
132 PUBLICATIONS **3,933** CITATIONS
SEE PROFILE

H. Schumann
University of Rostock
286 PUBLICATIONS **4,402** CITATIONS
SEE PROFILE

Fisheye Tree Views and Lenses for Graph Visualization

Christian Tominski
University of Rostock
ct@informatik.uni-rostock.de

James Abello
DIMACS, Rutgers University and Ask.com
abello@dimacs.rutgers.edu

Frank van Ham
IBM Research
fvanham@us.ibm.com

Heidrun Schumann
University of Rostock
schumann@informatik.uni-rostock.de

Abstract

We present interactive visual aids to support the exploration and navigation of graph layouts. They include Fisheye Tree Views and Composite Lenses. These views provide, in an integrated manner, overview+detail and focus+context. Fisheye Tree Views are novel applications of the well known fisheye distortion technique. They facilitate the exploration of the hierarchy trees associated with clustered graphs. Composite Lenses are the result of the integration of several lens techniques. They facilitate the display of local graph information that may be otherwise difficult to grasp in large and dense graph layouts.

1. Introduction

Graph structures are ubiquitous in our world. To support the analysis of graphs, a variety of visual methods (i.e. methods that map abstract graph structures to spatial layouts) have been developed in recent years [1]. However, for large graph data, the layouts, and hence the visual representations, tend to become dense and cluttered. Virtually all of the known approaches to address this issue are based on the computation of a suitable hierarchy tree (hierarchical clustering [2], [3]) that can be used as a mental map to drive the navigation of a graph layout [4]. Interaction techniques are also useful tools for supporting the exploration of large graphs. Specifically, overview+detail techniques [5] provide users with a coarse overview of a graph and allow detailed views of portions of the graph on demand. Focus+context techniques aim at integrating both, detailed views (focus) and overview (context).

In this paper we offer fisheye tree views and lenses as aids to interactive navigations of large clustered graphs. We enhance the interaction by providing overview+detail and focus+context in a seamless fashion. In Section 3 we explain how to apply the well-known fisheye distortion approach to the familiar vertical indentation-based tree layouts so prevalent in most desktop interfaces today. The result is what we term *Textual Fisheye Tree Views*. We also describe *Non-Textual Fisheye Tree Views* that are based on the classic Reingold & Tildford layout [6]. Both proposed views allow an intuitive interactive navigation of arbitrary trees (e.g. file systems, classifications), and more specifically of hierarchy trees of clustered graphs.

In Section 4 we propose novel lens techniques for supporting the exploration of graph layouts. These lenses are helpful tools for interactively accessing local graph information that might be difficult to perceive without a lens. Specifically, we introduce a *Local Edge Lens* that supports the detection of edges in dense graph layouts and a *Bring Neighbors Lens* that can be used to interactively bring to the current view the neighbors of some focused vertices irrespective of the positions of these neighbors in the original graph layout. We will also indicate how to combine the proposed lenses and a fisheye lens into a *Composite Lens*.

2. Related work

A *graph* consists of a set of *vertices* and a collection of pairs of vertices called *edges*. A *rooted tree* is an undirected connected graph without cycles with one particular vertex marked as the *root*.

To enable the visual analysis of a graph that is large (either with respect to the available display or with respect to the available random access memory [3]), hierarchical clustering can be applied to create a hierarchical partition of the graph's vertices. The outcome of this process is a so called *clustered* (or *hierarchical*) *graph* [2], [3]. It is commonly represented by a *hierarchy tree* whose leaves are in one to one correspondence with the original graph vertices. Each non-leaf node in the hierarchy tree represents the input subgraph induced by its set of descendant leaves. The approaches presented in [4] and [7] illustrate how a hierarchy tree can be used as a mental map to drive the navigation of a large graph.

To render a graph structure on a computer display, a spatial layout of the graph vertices must be computed. Depending on several graph characteristics (e.g. size) and aesthetics criteria, various algorithms can be applied to accomplish this task [1]. Since real-world graph data is usually large, its visual representation, even if generated by sophisticated methods, may result in an undesirably dense and cluttered display. Therefore, approaches have been developed that address the visual exploration of large graphs [2]. Commonly, zooming and panning of the graph representation is provided to allow users to switch between overview and detailed representations (overview+detail). More sophisticated interactive graph visualizations (e.g. [8], [9], or [10]) provide an integrated representation of details (focus) and overview (context). By operating either on a graphical or semantic level, such techniques realize the focus+context concept.

Both, overview+detail and focus+context, are established concepts that ease the navigation, and thus, the visual exploration of information spaces. However, in overview+detail techniques, users are obliged to mentally combine the provided overview and detailed views. When zooming and panning, users have to switch frequently between overview and zoomed detail views when exploring the data. On the other hand, focus+context techniques provide detailed views integrated into an overview. However, this usually involves some degree of distortion in the overview (i.e. the context). Especially for large datasets, the information presented in the distorted context is difficult to interpret.

Current graph visualization approaches usually focus on only one concept – either overview+detail or focus+context. To our knowledge, their combination is pursued only rarely. We present novel techniques that consider both, overview+detail as well as focus+context, in a seamless fashion. The obtained visual representations can be zoomed and panned at all times for overview+detail and provide enhanced interactive focus+context techniques on demand. Fisheye Tree Views and Graph Lenses encapsulate our approach and are introduced in the following sections.

3. Overviews for navigating clustered graphs

Many practical applications are based on a hierarchical structure (e.g. company organization, file systems, categorizations). Depending on the application domain different visual representations of trees are useful (e.g. [1], H3 [11], Beamtrees [12], InterRings [13], Magic Eye View [14], or DOI trees [15]).

Hierarchical structures, particularly hierarchy trees, are essential for the visualization of clustered graphs. Specifically, a hierarchy tree, if visualized adequately, can serve as an overview of the underlying graph structure. Therefore, we follow the suggestion of [4] and use separate tree views to support the navigation of clustered graphs. Most of the tree views proposed in the literature create visual representations that use most of the available display space. In our scenario, we need the tree views to be compact, since multiple views have to share a common display. In what follows we introduce two interactive tree representations – a *Textual Fisheye Tree View* and a *Non-Textual Fisheye Tree View* that can be used for this purpose. They can be applied to arbitrary hierarchical datasets. Specifically Textual Fisheye Tree Views could be used for a variety of daily tasks like browsing a file system or a web site.

3.1. Textual Fisheye Tree View

Most of today's users are familiar with the navigation of tree views as used to represent the files and directories on their computers. This suggests that a visual interface for navigating clustered graphs should provide these familiar views as well. However, common tree views are somewhat limited in their interaction capabilities. Our goal is to enhance common tree views with overview+detail and focus+context interaction techniques. We develop Textual Fisheye Tree Views for the visual exploration and navigation of trees, particularly hierarchy trees of clustered graphs.

The vertical indentation-based tree layout of common tree views has been retained for Textual Fisheye Tree Views. Furthermore, the common interactions – expanding and collapsing tree vertices as well as scrolling the entire tree layout are provided as usual.

In common tree views child-vertices appear or disappear immediately when expanding or collapsing a parent-vertex. For our Textual Fisheye Tree Views we utilize animation to create smooth transitions between different navigation steps. By doing so, the mental map of the tree is maintained, and it is easier for users to understand which child-vertices have been added to or removed from the view as a result of a collapse or expansion operation.

A disadvantage of common tree views is that they are capable of representing only a limited number of vertices at once. Even though a vertical and a horizontal scroll bar can be used to explore a tree in its breadth and depth, i.e. to bring different parts of a tree structure to the view, the general overview of a tree is still difficult to grasp. Consequently, it makes sense to provide full support for the overview+detail concept.

As a first novel interaction option for Textual Fisheye Tree Views, zoom functionality has been integrated. Figure 1 exemplifies how users are enabled to zoom out a tree view by using the mouse's scroll wheel. The advantage of the zooming functionality is that a larger number of vertices can be brought to the display. This provides a better overview and helps reveal the tree structure more easily. This aspect is essential when using a hierarchy tree as a mental map to navigate a clustered graph.

A draw back of zooming out a textual representation is that after a certain point the vertex labels can no longer be read. This is due to the tradeoff between the two contradictory requirements of providing an overview and maintaining readability. As common to many overview+detail techniques, users are forced to frequently switch between normal views and zoomed-out views

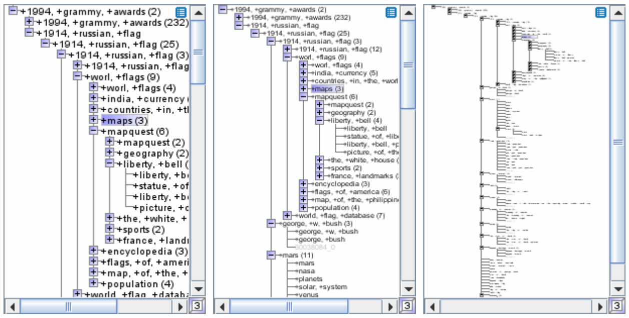

Figure 1: Differently zoomed Fisheye Tree Views.

during data exploration. To alleviate this problem we incorporate focus+context during navigation by applying a 1D fisheye transformation similar to Fisheye Menus [16]. This gives users the option of restoring the readability of a vertex label of interest and its immediate context.

By applying a 1D fisheye transformation, the label at the focus point is magnified up to a user-defined amount (depicted at the lower right corner of the Textual Fisheye Tree View). Neighboring labels gradually decrease in size, by which a smooth integration of the focus label into the context is achieved. It must be mentioned that in contrast to other approaches that utilize fisheye transformations, we apply this transformation only to the vertices currently displayed rather than to the entire dataset. Therefore, Textual Fisheye Tree Views instead of having the focus point linked to a particular vertex of interest focus on an entire row of the tree view irrespective of which node is currently displayed within this row. This allows scrolling the tree view at all times, even when the fisheye transformation is applied. As such, the Textual Fisheye Tree Views realize a combination of the overview+detail and focus+context concepts. This is achieved by automatic panning based on the mouse cursor position. We do not expect the described enhancements to decrease the user acceptance of tree views, since the interaction techniques that we provide are only active on demand.

Figure 2 illustrates the effect of applying a fisheye transformation as used by Textual Fisheye Tree Views. As demonstrated in the figure, different amounts of magnification can be used to control the readability of labels in the context of the focused vertex. The vertical and horizontal scroll bars can always be accessed to bring further vertices to the display.

We have indicated how common tree views can be enhanced with interaction techniques and a 1D fisheye transformation. Textual Fisheye Tree Views are especially useful for trees structures that have associated with each vertex a label that has to be displayed. However, they are limited in their ability to represent deep hierarchical structures. In the next section we present a complementary tree view that is better suited to deal with this case.

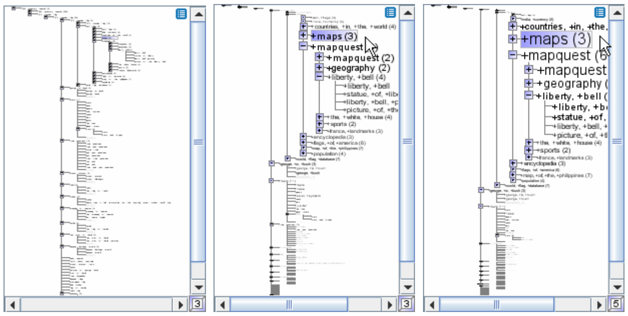

Figure 2: When the Fisheye Tree View is zoomed out, the vertex labels cannot be read (left). Applying different amounts of fisheye magnification re-enables readability for a focus vertex and its context (middle and right).

3.2. Non-Textual Fisheye Tree View

Since Textual Fisheye Tree Views are not very effective for the visual navigation of deep hierarchy trees, we suggest the use of an alternative complementary tree layout for this purpose. This alternative layout, when enhanced with the interaction techniques described in the previous section, is referred to as Non-Textual Fisheye Tree View.

We apply the classic Reingold & Tildford layout algorithm [6] to spatially lay out the tree vertices. Since in the resulting layout, tree vertices are aligned horizontally and deeper hierarchy levels are located below higher levels (in terms of their y-coordinate), it is possible to render the tree by using either dots & lines or triangles. Although it is difficult to display vertex labels for these representations, the depth of the hierarchical structure is more apparent and it is now possible to color code a variety of parameters in the dots or the triangles. This is especially useful when clustered graphs are explored according to certain attribute values associated with the vertices of a graph, e.g. vertex degree, number of users logged on at different servers in a computer network, CPU load (Figure 3 exemplifies color coded degrees). To support users during graph exploration and navigation, an expressive color coding is required that takes into account the characteristics of the currently visualized parameter [17] (e.g. in Figure 3, red vertices represent large degree).

Since aggregation operations are applied when generating a clustered graph, details about the underlying (unclustered) graph can be found only at the leaves of the hierarchy tree. Since details are hard to see in dense layout regions or due to occlusion, we provide zoom and pan functionality. Furthermore, similar to the previous section, users can on demand apply a 1D fisheye transformation to the tree layout. In this case, however, the fisheye transformation is applied horizontally (see Figure 3). By interactively moving the transformation's focus point, users can search dense regions of the tree

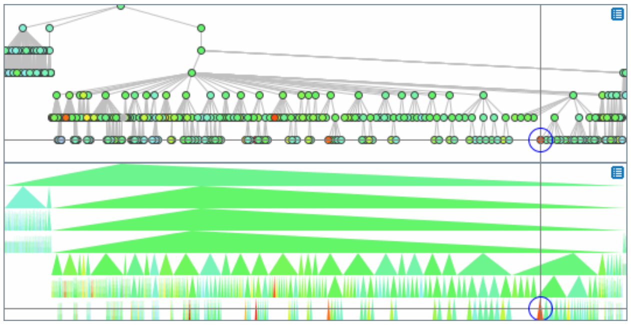

Figure 3: Interesting vertices (marked with a blue circle) and their immediate context can be perceived more easily when interactively applying a 1D fisheye transformation.

representation for interesting details or uncommon vertices.

Textual Fisheye Tree Views (inspired by [16]) and Non-Textual Fisheye Tree Views (based on [6]), exemplify how quite standard tree representations can be extended with useful interaction techniques. We hope that the simplicity of these representations ensures a wide range of applicability and high user acceptance. The on-demand character of the introduced interaction methods allows the application of the described approaches to more complicated tasks. Fisheye Tree Views are currently being integrated in a graph exploration system currently under development [18].

4. Lenses for graph exploration

The previous section was dedicated to illustrate how intuitive overviews can be used to support the navigation of hierarchical structures. In this section concepts are developed that aim at supporting the visual exploration of graph layouts. A noteworthy difference between tree layouts and graph layouts is that the structure of an arbitrary graph is more difficult to convey. Whereas layout algorithms for trees can take into account the inherent hierarchical structure, when laying out arbitrary graphs such assumptions cannot be made. Therefore, an important task is to support users in exploring unfamiliar graphs especially when their layouts are dense or cluttered. As described for instance in [9] or [10], lens techniques can be used to alleviate this problem. In what follows we present novel interactive lens techniques for supporting the visual exploration of graph layouts. We also introduce a Composite Lens, i.e. a lens that is composed of several elementary lenses.

We assume that the coordinates for the graph vertices have been calculated by an adequate layout algorithm (see [1]). To enable an easy navigation of the graph, common zoom and pan functionality is provided and smooth animated transitions [19] are used to switch between overview and detailed views.

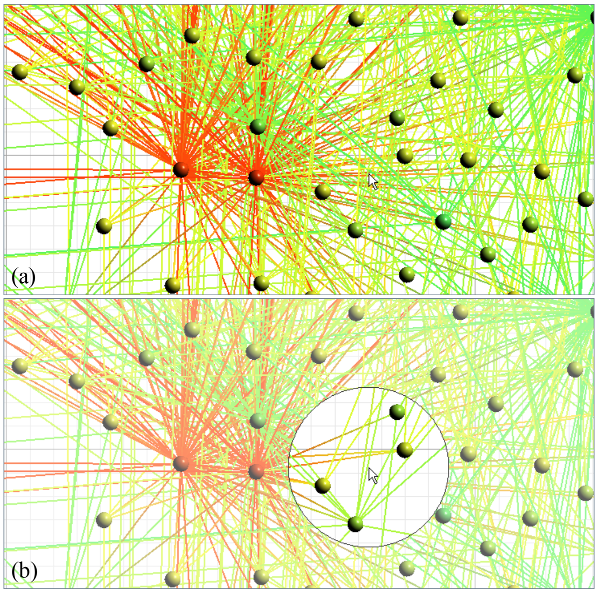

Figure 4: If graph visualizations are cluttered (a), it is hard to identify which edges are connected to particular nodes. Applying the interactive Local Edge Lens (b) enables users to accomplish this task easily. The color of an edge is interpolated between the colors of the vertices connected by the edge.

4.1. Local Edge Lens

Similar to EdgeLenses [9], the first lens we present addresses the heavy clutter of edges that might occur when visualizing large graphs. As shown in Figure 4(a), when visualizing non-planar graphs consisting of large numbers of edges, the display can end up completely covered with lines. This renders the identification of edges connected to a particular vertex nearly impossible to accomplish. Since this is an essential task when exploring large graphs, we developed a *Local Edge Lens*. This lens aims at enabling users to identify edges connected to particular vertices currently on the display. To achieve this, users are provided with a spatially confined, interactively movable lens similar to [20]. When rendering the graph layout outside the lens, all edges are shown. However, inside the lens area (i.e. the focus) only edges connected to vertices within the lens scope are drawn. By doing so, the cluttering of edges is removed for a local focus region defined by the position and scope of the lens. As can be seen in Figure 4(b), the Local Edge Lens tidies up the lens area, and thus, enables users to identify local edges of the vertices within the lens scope. Additionally, a semitransparent pane has been drawn outside the lens area. By this milk glass-like effect, the attention of the user is attracted to the relevant information presented inside the lens.

The Local Edge Lens operates exclusively in the representation space, i.e. it does not alter the layout of the graph. As such, the Local Edge Lens is purely graphical and can be quickly computed by today's graphics cards. More complicated lenses that operate exclusively on the graph layout are presented next.

4.2. Bring Neighbors Lens

The second lens presented in this paper addresses another issue that arises when navigating large graphs – users commonly need to locate the neighbors of a particular vertex of interest, i.e. the vertices that have an edge with the vertex of interest. Depending on the layout algorithm used, neighboring vertices are not necessarily close in the graphical representation. This is particularly apparent when working on a deeply zoomed representation of a graph layout. In this case, very often neighboring vertices are even off screen. To give users an interactive tool that brings all neighbors of a vertex of interest to the display, regardless of their actual position in the graph layout, the *Bring Neighbors Lens* has been developed. Since the Bring Neighbors Lens alters the positions of vertices in the graph layout, this lens is really a layout lens.

Again, the lens is defined by a focal point and has a scope determined by a radius from the focal point. The aim is now to bring in the neighbors of a vertex that is approached by the lens. In other words, when a user moves the lens (using the mouse) towards a vertex, the neighboring vertices should be brought to the lens in a continuous fashion. This is achieved by relating the layout distance between the vertex of interest and its neighbors to the layout distance between the focal point of the lens and the vertex of interest. Specifically, if the layout distance between the focal point and the vertex of interest is zero (i.e. the lens is directly on top of the vertex of interest), then all neighboring vertices are brought to the lens area such that the farthest neighbor (with respect to the original layout) is positioned on the lens perimeter and all other neighbors are positioned closer to the lens center. On the other hand, if the vertex of interest is on the perimeter of the lens, no change to the positions of the neighboring vertices is applied. For an intermediate position of the vertex of interest (i.e. a position between the focal point and the lens perimeter), the positions of the neighboring vertices are linearly interpolated. By applying this interpolation, while moving the lens towards a vertex of interest, users get an impression of the positions of neighboring vertices in the original layout. Moreover, when all neighbors have been brought to the lens (the lens is directly on top of the vertex of interest), users can estimate the distance of the neighbors to the vertex of interest in the original layout. Figure 5 demonstrates the use of the Bring Neighbors Lens. The figure shows on the left an initial display where some of the neighbors of the vertex of interest are off screen. On the right hand side, the figure shows how the display looks like when the lens has been moved directly on top of the vertex of interest. Additionally, several intermediate steps of the movement of the lens towards the vertex of interest are shown.

In the previous description we have assumed that only one vertex is within the scope of the lens. However, the lens scope can be taken to focus on more than one vertex. Namely, all vertices inside the lens perimeter could be considered. To this end, we developed a

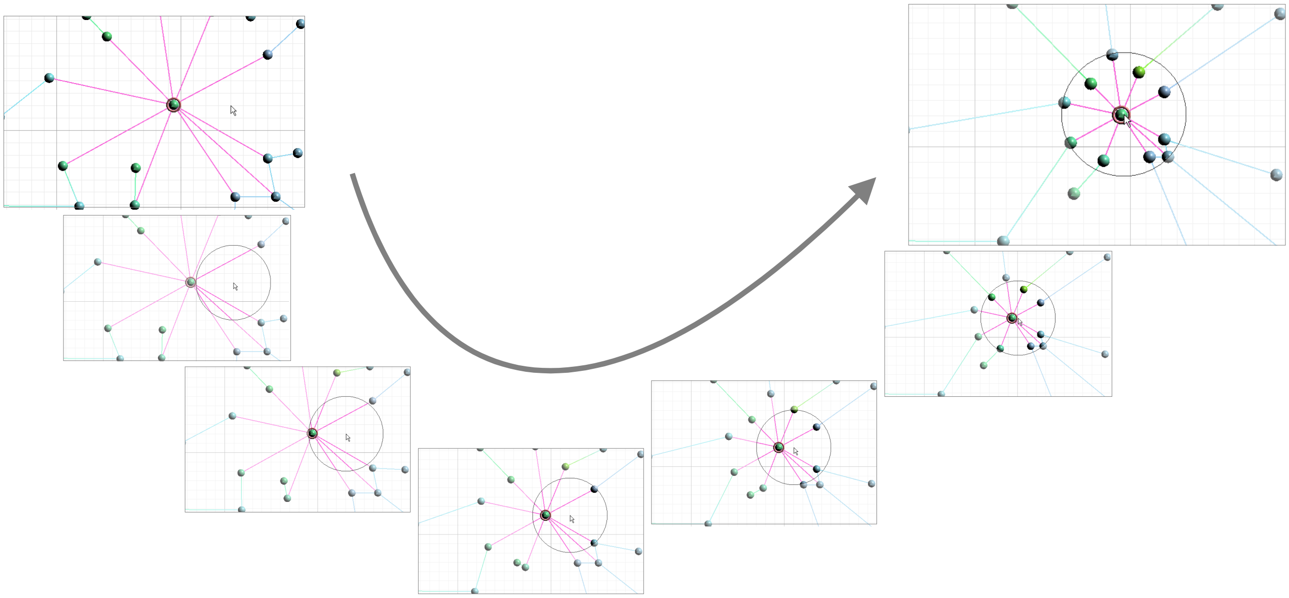

Figure 5: The figure demonstrates how the Bring Neighbors Lens can be used to interactively bring the neighbors of a vertex of interest to the display.

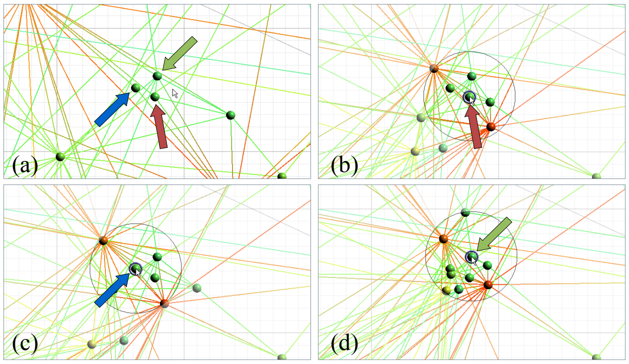

Figure 6: This figure demonstrates the effect of applying the Bring Neighbors Lens for three vertices of interest. The image (a) shows the three relevant vertices without the lens applied. The images (b), (c), and (d) show the lens positioned on the respective vertices of interest.

Generalized Bring Neighbors Lens that is able to handle multiple in-lens vertices. The main idea is to calculate the attraction of neighboring vertices towards the lens depending on the number of vertices within the lens scope and their respective distances to the focal point of the lens. This enables us to calculate attraction weights for each considered vertex. The attraction weights determine how much the neighbors of the respective vertex are attracted towards the lens. To achieve adequately attracted neighbors, the attraction weights must obey the following five constraints:

1. The sum of all attraction weights must be one.
2. If the distance between a vertex of interest and the focal point of the lens is zero, then the respective attraction weight shall be one, resulting in all neighbors being brought within the lens perimeter.
3. If a vertex of interest lies on the perimeter of the lens, then the respective attraction weight becomes zero, resulting in no attraction of neighbors.
4. If two vertices of interest have the same distance to the focal point of the lens, their attraction weights are the same.
5. If one vertex of interest has a smaller distance to the focal point of the lens compared to a second vertex of interest, the attraction weight of the first node is greater than the attraction weight of the second node.

Having calculated the attraction weights for each vertex of interest within the stated constraints, it is now possible to compute the overall neighbor attraction as a weighted sum of the original one-vertex attraction case. Figure 6 shows how three different vertices of interest attract their respective neighbors differently, depending on the focal point of the lens. It must be mentioned that using still images of the Bring Neighbors Lens it is difficult to completely convey the lens effect, since much information (e.g. from where do vertices move in, how fast do they move in compared to other vertices) is communicated only when the lens is in motion.

4.3. Composite Lens

In the previous sections we presented lenses that work either on the representation level (Local Edge Lens) or the layout level (Bring Neighbors Lens). However, in user tests, we found out that this separation is not always the best solution. Quite the contrary, users suggested an integrated lens. Moreover, our user tests revealed a limitation of the Bring Neighbors Lens. If a neighboring vertex is far away from the focal point of the lens while all other neighbors are close by, the closer neighbors are attracted so much towards the vertex of interest that it is impossible to discriminate the neighboring vertices from each other (see Figure 7(c)). To alleviate this problem, we subsequently apply the well known fisheye lens. Since a fisheye lens moves vertices away from the focal point, this spreads the neighbors attracted “too much” to the respective vertex of interest.

To combine the two previously introduced lenses and the well known fisheye lens into an enhanced *Composite Lens*, it is necessary to determine the lens composition order. The visualization pipeline as proposed in [21] describes visualization as a line of steps – *data analysis*, *filtering*, *mapping*, *rendering*. We adhere to this pipeline, and consequently, apply lenses that work on the layout level (mapping step) before lenses that operate on the representation level (rendering step). Since we want neighboring vertices to be attracted first by the Bring Neighbors Lens, and then slightly spread them by a fisheye lens, the overall order for the Composite Lens is determined: We apply the Bring Neighbors Lens first, the fisheye lens second, and the Local Edge Lens last.

The result of this consecutive application of lenses is what we call a Composite Lens. The visual appearance of the Composite Lens is compared to the Local Edge Lens and the Bring Neighbors Lens in Figure 7. Specifically, Figure 7(d) depicts the effect of the Composite Lens. Comparing figures (c) and (d) it becomes clear that the additional fish eye lens separates the neighboring vertices that were attracted towards the vertex of interest. Figure (d) shows that by integrating the Local Edge Lens, the lens area is clear from edges that are not currently relevant.

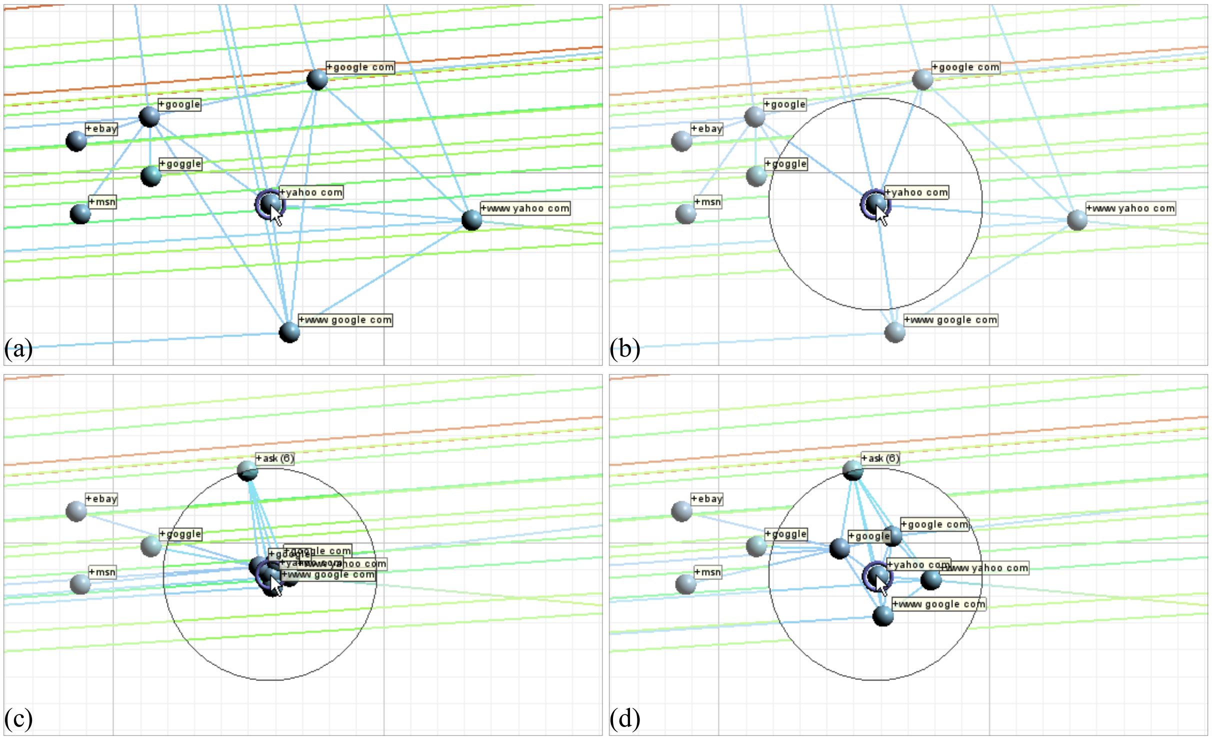

Figure 7: The figure provides a comparative view of the presented lens types. (a) No lens; (b) Local Edge Lens; (c) Bring Neighbors Lens; (d) Composite Lens which integrates (b) and (c) as well as an additional fisheye lens.

5. Conclusion and discussion

In this paper we have addressed aspects related to the navigation and visual exploration of graph structures. An adequate visual interface for this task must provide different interactive views that allow one to easily navigate the visualized structures and to intuitively explore large graph data according to different aspects.

We described Textual and Non-Textual Fisheye Tree Views as visual interfaces to trees. The simplicity of these interfaces opens up the possibility of using them for a variety of daily tasks. The presented interaction methods, specifically the on-demand 1D fisheye transformation, provide support for more complicated visualization tasks. This is achieved by incorporating overview+detail and focus+context concepts in an integrated fashion.

As a second contribution of this paper, we have exemplified the usefulness of lens techniques for displaying, on demand, local information that may be otherwise difficult to grasp in large and dense graph layouts. While the Local Edge Lens is a lens that supports the detection of edges within the lens scope, the Bring Neighbors Lens is a tool to bring in neighboring vertices of the vertices within the lens perimeter. The developed Local Edge Lens and the novel Bring Neighbors Lens have been combined with a fisheye lens to illustrate the composition of different lens types. By this composition, the advantages of the original lenses can be accommodated in one advanced lens.

The described methods have been incorporated in a framework for visualizing clustered graphs. We are currently evaluating and tuning the proposed methods for the exploration and navigation of clustered graphs with up to 100,000 vertices.

Although initial user tests have been conducted, which led to the development of the Composite Lens, it is necessary to further evaluate the proposed visual interfaces and interaction techniques with concrete observable graph exploration tasks. Among the concerns that surfaced in the initial user test were the coordination of the different views and the consistency of interaction among the different techniques. These will be addressed in future work. Further user tests could also lead to a demand for additional views or new lens techniques. Another interesting issue to investigate in the future is to develop a generalized model for integrating lenses at the different stages of the visualization pipeline.

6. Acknowledgements

Special thanks go to Konrad Borys for his advice regarding the Bring Neighbors Lens. The support provided by the DIMACS center at Rutgers University for this project is gratefully acknowledged.

7. References

[1] di Battista, G., Eades, P., Tamassia, R., and Tollis, I.G. *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1998.

[2] Herman, I., Melançon, G., and Marshall, M.S. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 6, No.1, 2000.

[3] Abello, J. and Korn, F. MGV: A System for Visualizing Massive Multidigraphs. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 8, No. 1, 2002.

[4] Abello, J., Korn, J., and Kreuseler, M. Navigating Giga-Graphs. *Proceedings of Advanced Visual Interfaces*, 2002.

[5] Shneiderman B. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualization. *Proceedings of IEEE Symposium on Visual Languages*, 1996.

[6] Reingold, E.M. and Tildford, J.S. Tidier Drawing of Trees. *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 2, 1981.

[7] Abello, J. and van Ham, F. Matrix Zoom: A Visual Interface to Semi-External Graphs. *Proceedings of IEEE Symposium on Information Visualization*, 2004.

[8] Brown, M.H. and Sarkar, M. Graphical Fisheye Views of Graphs. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,1992.

[9] Wong, N., Carpendale, S., and Greenberg, S. EdgeLens: An Interactive Method for Managing Edge Congestion in Graphs. *Proceedings of IEEE Symposium on Information Visualization*, 2003.

[10] van Ham, F. and van Wijk, J.J. Interactive Visualization of Small World Graphs. *Proceedings of IEEE Symposium on Information Visualization*, 2004.

[11] Munzner, T. H3: Laying out Large Directed Graphs in 3D Hyperbolic Space. *Proceedings of IEEE Symposium on Information Visualization*, 1997.

[12] van Ham, F. and van Wijk, J. J. Beamtrees: Compact Visualization of Large Hierarchies. *Proceedings of IEEE Symposium on Information Visualization*, 2002.

[13] Yang, J., Ward, M.O., and Rundensteiner, E. InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures. *Proceedings of IEEE Symposium on Information Visualization*, 2002.

[14] Kreuseler, M. and Schumann, H. A Flexible Approach for Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 8, No. 1, 2002.

[15] Herr, J. and Card, S.K. DOITrees Revisited: Scalable, Space-Constrained Visualization of Hierarchical Data. *Proceedings of Advanced Visual Interfaces*, 2004.

[16] Bederson, B.B. Fisheye Menus. Proceedings of ACM Conference on User Interface Software and Technology, 2000.

[17] Schulze-Wollgast, P., Tominski, C., and Schumann, H. Enhancing Visual Exploration by Appropriate Color Coding. Proceedings of International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG), 2005.

[18] Abello, J. and van Ham, F. AskGraphView: Designing a Large Scale Graph Visualization System, *under revision*, 2006.

[19] van Wijk, J.J. and Nuij, W.A. A Model for Smooth Viewing and Navigation of Large 2D Information Spaces. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 10, No. 4, 2004.

[20] Stone, M., Fishkin, K., and Bier, E. The Movable Filter as a User Interface Tool. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1994.

[21] dos Santos, S. and Brodlie, K. Gaining understanding of multivariate and multidimensional data through visualization. *Computers & Graphics*, Vol. 28, No. 3, 2004.