Drawing Large Graphs with H3Viewer and Site Manager (System Demonstration)

Tamara Munzner

Stanford University, 360 Gates CS Building 3B, Stanford, CA 94305, USA
munzner@cs.stanford.edu, http://graphics.stanford.edu/~munzner

Abstract. We demonstrate the H3Viewer graph drawing library, which can be run from a standalone program or in conjunction with other programs such as SGI's Site Manager application. Our layout and drawing algorithms support interactive navigation of large graphs up to 100,000 edges. We present an adaptive drawing algorithm with a guaranteed frame rate. Both layout and navigation occur in 3D hyperbolic space, which provides a view of a large neighborhood around an easily changeable point of interest. We find an appropriate spanning tree to use as the backbone for fast layout and uncluttered drawing, and non-tree links can be displayed on demand. Our methods are appropriate when node or link annotations can guide the choice of a good parent from among all of the incoming links. Such annotations can be constructed using only a small amount of domain-specific knowledge, thus rendering tractable many graphs which may seem rather densely connected at first glance.

1 Motivation and Context

Software systems which support graph drawing occur in many varieties. Some systems are partially automatic but provide the user with a graphical user interface to fine-tune the resulting image [2]. Others take a data file as input and automatically create a polished drawing as output in a batch process [3]. A common theme that runs through most systems is a strong limit on the number of nodes which can be handled, either because of the presumably bounded patience of the human in the loop or because the totally automatic layout algorithms do not scale to a large number of nodes. Unfortunately, real-world graphs are often several orders of magnitude too large for the capabilities of current software systems, which handle only hundreds or perhaps thousands of edges.

In pushing beyond previous limits on graph size, we explicitly traded off generality for scale. Our approach fills a gap in the design space not addressed by previous systems. We have built a software library which supports interactive exploration of large graphs ranging from thousands to over 100,000 edges. Our library achieves this speedup by using a spanning tree as a backbone for layout and drawing. Tree drawing is a more tractable problem than general graph layout [12, 8]. The user can draw incoming or outgoing non-tree links on demand for any node or subtrees.

GD '98: Symposium on Graph Drawing, Montreal, Canada, August 1998

1.1 Spanning Trees

Clearly a spanning tree approach will work well with trees or directed acyclic graphs. There are definitely graphs where the spanning tree backbone may result in a misleading picture, for example bipartite or fully connected graphs. One might assume at first glance that our method is only appropriate for graphs which are very tree-like and must be inappropriate for more densely connected ones. However, we argue that there are in fact many "quasi-hierarchical" graphs for which our algorithms are well-suited. Some authors use *hierarchical* interchangeably with *directed acyclic* (i.e. [4]), but we have a more general class in mind. There are many graphs which might be quite densely connected from a purely graph-theoretic standpoint, but a small amount of domain-specific knowledge provides a way to make a good choice of a main parent from among all the incoming links to a given node. Nodes or links can be annotated with this information. We argue that the resulting tree-based backbone drawings enrich rather than mislead the user.

1.2 Abstraction

Some interesting attempts have been made to handle complexity through automatic abstraction [6]. The work described here is complementary to such efforts. Abstraction and sheer scale are both important additions to the graph drawing arsenal of methodology. However, the use of a spanning tree can also be thought of as an automatic filter which elides links in the default case.

2 System

A graph viewer is most helpful to a user when it is integrated with other tools. The H3 layout and H3Viewer drawing libraries have been integrated into the Site Manager system from Silicon Graphics, which is aimed at webmasters and content creators. The screen snapshot in Figure 1 shows part of the Stanford Graphics Group web site on display during a Site Manager interactive session. The 3D hyperbolic graph view shows the hyperlink structure of the site, while the 2D browser view shows the directory tree structure in traditional outline format. Selection in one view causes highlighting and transitions in both. The graph structure can thus be used as an index for selecting items. The ability to quickly select a subtree can be useful even for users who already understand the structure of their graph. A more prolonged discussion about the suitability of our methods for various other tasks can be found in a recent paper [10].

2.1 Interaction

When the user clicks on a node, it is highlighted and undergoes an animated transition to the center of the sphere. The importance of smooth

transitions for maintaining a consistent mental model of an onscreen structure is now well known [13]. The transition includes both a translational and rotational component, so that when a node reaches the origin its ancestors always appear on its left and its descendants are to the right. This "butterfly" configuration both provides a canonical local orientation and also serves to minimize occlusion of both nodes and their text labels.
The Site Manager system features a tight integration between the 3D hyperbolic graph browser and a standard 2D file browser. When a node is selected in one, it is highlighted and moved to the center in both. If several nodes are selected at once, by rubberbanding in the outline view or selecting an entire subtree in the graph view, they are highlighted in both but no motion occurs. The support for brushing allows users to correlate information across views.
The same criteria used to color nodes can also be invoked as a filter to show or hide nodes from view. Users can also choose whether to show incoming or outgoing non-tree links for the selected node or set of nodes.

2.2 Speed and Size

The layout phase is linear in the number of spanning tree edges. On an SGI Onyx2 a large graph with 110,000 edges was laid out in 12 seconds. A medium sized graph of 31,000 edges was laid out in 4 seconds, and a small graph of 4000 edges took a fraction of a second. The drawing time is constant - the H3Viewer drawing algorithm always maintains the target guaranteed frame rate. A slow graphics system will simply show less of the context surrounding the node of interest during interactive manipulation. Figure 2 shows two views of the same scene - one corresponding to what the user would see during interaction, and one after the fringe has been filled in when the user was idle.
The layout and drawing algorithms presented here work well up to the limits of main memory, but not beyond: if the entire graph does not fit into main memory, the system is unusable. The graph with 110,000 edges could be manipulated interactively on an SGI with 1 GB of main memory but could not be loaded on a smaller 128 MB machine.
On startup, the initial loading phase includes both file I/O and data structure building. For the graphs mentioned above, this time was approximately 2 minutes, 20 seconds, and 2 seconds, respectively.

2.3 Availability

The Site Manager application can be downloaded for free from `http://www.sgi.com/software/sitemgr.html`. It runs only on Silicon Graphics machines. Version 1.0 included only the H3 layout component, whereas version 1.1 also incorporates the H3Viewer guaranteed frame rate libraries.
The underlying H3/H3Viewer library source will soon be released for free non-commercial use. Commercial use must be licensed through Silicon Graphics, Inc. It will run on any machine that has OpenGL, which includes most Unix and Windows boxes.

3 Hyperbolic Layout

Our layout approach can be briefly described as a “second-generation 3D hyperbolic cone tree”. The basic scheme is related to the influential cone tree method of recursively drawing trees in 3D space [12]. Our novel H3 layout algorithm distributes child nodes on the surface of a hemisphere on the mouth of the cone rather than around its linear circumference. Details on the two-pass layout technique and spanning tree construction are in a recent paper [9].

We exploit two key properties of hyperbolic space: exponential room and an outsider’s view. The surface of a sphere or the circumference of a circle grows exponentially rather than geometrically as its radius increases in hyperbolic space, because of the non-Euclidean distance metric. The layout problem of an exponential explosion of tree nodes can be nicely accommodated by the exponential amount of space in which to place them.

The mathematical literature documents several standard projections which map an infinite amount of hyperbolic space to a finite section of Euclidean space [14]. We use the Klein model, where the projected area is a ball in three-dimensional space with straight lines preserved but distorted angles. Objects near the center of the ball are full size, but their projected size shrinks as they approach (but never reach) the “sphere at infinity” which is the surface of the ball. This outsider view provides a natural way to focus interest in one part of the graph structure and see a large amount of context surrounding it.

These two convenient properties of hyperbolic space were also exploited in the two dimensional hyperbolic tree browser developed at Xerox PARC and productized through Inxight [7]. Their system only handles trees, while ours supports general graphs. Moreover, our layout algorithm is quite different. Using a spanning tree backbone in a 2D layout algorithm would necessarily lead to edge crossings when drawing the non-tree edges. The 3D tree layout strategy can accommodate non-tree edges without actual edge crossings, although from any single viewpoint there will be occlusion. We rely on interactive navigation to help the user understand the 3D structure.

Our first straightforward attempt at drawing cone trees in 3D hyperbolic space, the Geometry Center *webviz* system, suffered from rather low information density [11]. The H3 layout strikes a reasonable balance between information density and clutter. The traditional cone tree layout in both the Xerox PARC Cone Tree and the Geometry Center *webviz* system places nodes on a circle – a 1D line. In H3, nodes are placed on a hemisphere – a 2D surface. In a paper from Carpendale et al [1], nodes are placed in a 3D grid – a 3D volume. In all three of these examples, the space in which nodes are laid out is a 3D volume. When the dimension of the surrounding space is the same as the dimension of the node structures, occlusion becomes the overriding issue. The entire focus of the Carpendale paper is proposing various solutions for the occlusion problem. We choose to lay out nodes on a surface, which is a happy medium between the sparseness of a line and the density of a volume. An excessively sparse layout like the *webviz* system wastes screen real estate. If our layout was too dense, the leaf nodes near the surface of the ball would block our view of the rest of the structure, since we are outside of the ball looking in.

4 Drawing

Our drawing algorithm depends on the number of visible, not total, number of nodes and edges. The projection from hyperbolic to Euclidean space guarantees that nodes sufficiently far from the center will project to less than a single pixel. Thus the visual complexity of the scene has a guaranteed bound - only a local neighborhood of nodes in the graph will be visible at any given time.

4.1 Adaptive Drawing

A guaranteed frame rate is extremely important for a fluid interactive user experience. The H3Viewer adaptive drawing algorithm is designed to always maintain a target frame rate even on low-end graphics systems. A high frame rate is maintained by drawing only as much of the neighborhood around a center point as is possible in the allotted time. The drawing algorithm incorporates knowledge of both the graph structure and the current viewing position. We use the link structure of the spanning tree to guide additions to a pool of candidate nodes and the projected screen area of the nodes to choose from among the candidates. The link structure of the spanning tree provides us with a graph-theoretic measure of the distance between two nodes: the number of *hops* from one node to another is the integer number of links between them. Nodes which are a short number of hops from the center will usually be more visible than those which are a large number of hops away. If we have just drawn a node, its parents and direct spanning tree children, which are just one hop away, are reasonable candidates for drawing next. However, if we simply rely on graph structure, we may waste time filling in sections that are less visible while neglecting those which are more prominent. While the hop count between two nodes does not change during navigation, the projected screen area of nodes does vary. Two nodes an equal number of hops from the center node may have much different projected screen areas.

The projected screen area of a node depends on the current position of the graph in hyperbolic space. Navigation occurs by moving the object in hyperbolic space, which is always projected into the same fixed area of Euclidean space. Motion on the surface of the 3-hyperboloid brings some node of the graph closest the pole, so its projection into the ball is both closest to the origin and largest. Most previous systems for adaptive drawing [5] state the viewpoint problem in terms of camera location, but since we have a hyperbolic viewer we need to consider the projection of a moving object onto a fixed camera.

The amount of time devoted to a frame should depend on the activity of the user. The H3Viewer library allows separate control over the drawing frame time, picking frame time, and idle frame time. The rendering frame time is simply the time budget in which to draw a single frame during user mouse movement or an animated transition. It is clear that the drawing time should be explicitly bounded instead of increasing as the node/edge count rises. The time spent casting pick rays into the scene must be similarly bounded. Rendering and picking should be accomplished somewhere between five and thirty frames per second - our current default is 20 FPS for rendering and 10 FPS for picking. When the user and application are idle, the system can fill in more of the surrounding scene. However, the time spent on this, the idle frame time, should still be bounded to eventually free the CPU for other tasks. On a low-end SGI machine, a few seconds is often enough time to fill in most of the desired detail. Our current default is 2 seconds.

4.2 Drawing Implementation

The heart of our drawing algorithm is a loop which draws nodes from the center of the sphere outwards until the time budget for a frame is exhausted. The *ActionQueue* is cleared at the beginning of a frame. It is initialized with a new center node, which for frame f_t is the node which was closest to the origin at frame f_{t-1}. It is safe to rely on frame-to-frame coherency since our algorithm guarantees high frame rates. At the top of the loop we pop the top node of the ActionQueue. The candidate entities to draw are the node itself, its incoming and outgoing links, the parent node, and the direct child nodes. Any of these items which have not yet been drawn are rendered and marked as drawn. The parent and child nodes are then inserted into the ActionQueue. Note that although only nodes which are one hop away in the spanning tree are enqueued, the links which are drawn may include non-tree links if the user has enabled them. As nodes are drawn, their projected screen area is recorded and used to sort the ActionQueue. At the bottom of the loop, if the elapsed time is greater than the allotted frame time the loop returns. Otherwise, the loop is traversed again with the next node on the ActionQueue.

If the system is in idle mode, control may be returned to the drawing loop through the invocation of an idle callback. In that case, the old ActionQueue is retained rather than being cleared and initialized with a new center node. The drawing loop is invoked again, and is returns after another drawing frame time's worth of activity. Control can be returned to the idle callback several times in a row, until the idle frame time is exceeded. Then the idle callback is cleared so that control returns to the application program until further user or program activity warrants new drawing activity. It is important to use the idle callback mechanism to trigger many invocations of the short drawing frame time loop, rather than indulge in drawing for the entire unbroken idle time, since the user may act at any time.

All drawn nodes are kept in an additional queue, the *PickQueue*, which is also sorted by projected screen area. This queue is used by the picking routines, which must also have a guaranteed termination time. The pick could be occurring after the system has drawn several drawing frame time's worth of nodes while in idle mode, so there may be more onscreen nodes than can be tested for intersection with the pick ray in the allotted time. The right strategy to minimize user frustration is to ensure that large visually distinguishable nodes can be successfully picked, since smaller nodes with only a few pixels of screen presence are less likely to be the target of user interaction. If a pick ray does collide with a node, that node alone is redrawn in a highlighted color to instantly provide visual feedback. The picking cycle of the H3Viewer is fast enough to allow locate highlighting whenever the user simply moves the mouse over an object, and of course can also be used to determine whether the user has selected a node in response to an explicit click.

4.3 Attributes

The basic geometry of nodes and links can be augmented through decorations such as color coding, changes of linewidth, and text labels. Such additions greatly increase the utility of the system. Text labels are drawn for nodes whose projected screen areas are greater than a user-specified number of pixels. The current implementation draws spanning tree links with a linewidth of two and non-tree links with a single-pixel line. The directionality of the links is subtly color-coded, with the reddish end emerging from the parent and the bluish end terminating at the child. This color coding is non-intrusive when only the spanning tree is shown, but allows the user to distinguish between incoming links and outgoing links when non-tree link drawing is enabled.

Many Euclidean graph drawing systems also encode information in the shape of the drawn node, but we have chosen to avoid this modality. The user is much less able to distinguish node shapes in a hyperbolic viewer than in a Euclidean viewer, since the number of pixels devoted to a node shrinks very rapidly as that node moves away from the focus point.

In the Site Manager system the default node color coding is according to the MIME type of the document: HTML documents are cyan, images are purple, VRML is blue, and so on. When traffic statistics are inspected, the nodes are coded on a red to grey color gradient, where red represents the most number of hits and grey the least. Color coding can also be used to show dynamic data. A site's traffic logs can be used to show the paths taken by Web users. A hit from one page to another is shown by briefly highlighting the link between them, which may have previously been drawn in the default link color if part of the spanning tree or may not have been drawn.

5 Conclusion

Our H3Viewer library can handle graphs two orders of magnitude larger than previous systems by manipulating a backbone spanning tree instead of the full graph. We carry out both layout and drawing in 3D hyperbolic space in order to see a large amount of context around a focus point. Our layout is tuned for a good balance between information density and clutter, and our adaptive drawing algorithm provides a fluid interactive experience for the user by maintaining a guaranteed frame rate.

6 Acknowledgements

We acknowledge the efforts of the entire Site Manager team at Silicon Graphics: Ken Kershner, Greg Ferguson, Alan Braverman, Donna Scheele, Doug O'Morain, and Julie Brodeur. We thank François Guimbretière and Pat Hanrahan for many productive discussions. Thanks also to Anwar Ghuloum of the Stanford SUIF compilers group for the function call graph data. This work was supported in part by Silicon Graphics, the NSF Graduate Research Fellowship Program, and ARPA.

References

[1] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Extending distortion viewing from 2D to 3D. *Computer Graphics and Applications*, pages 42–51, 1997.

[2] Michael Fröhlich and Mattias Werner. Demonstration of the interactive graph visualization system *da Vinci*. In *Proceedings of Graph Drawing '94, Lecture Notes in Computer Science 894*, pages 266–269. Springer-Verlag, 1994.

[3] Emden R. Gansner, Eleftherios Koutsofois, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–229, March 1993.

[4] Ashima Garg and Roberto Tamassia. Giotto3d: A system for visualizing hierarchical structures in 3d. In Steven North, editor, *Proceedings of Graph Drawing '96), Lecture Notes in Computer Science 1190*. Springer-Verlag, 1996.

[5] Hugues Hoppe. View-dependent refinement of progressive meshes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1997.

[6] Doug Kimelman, Bruce Leban, Tova Roth, and Dror Zernik. Reduction of visual complexity in dynamic graphs. In *Proceedings of Graph Drawing '94, Lecture Notes in Computer Science 894*, pages 218–225. Springer-Verlag, 1994.

[7] John Lamping, Ramana Rao, and Peter Pirolli. A focus+content technique based on hyperbolic geometry for viewing large hierarchies. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 401–408, Denver, May 1995. ACM.

[8] Sven Moen. Drawing dynamic trees. *IEEE Software*, pages 21–28, july 1990.

[9] Tamara Munzner. H3: Laying out large directed graphs in 3D hyperbolic space. *Proceedings of the 1997 IEEE Symposium on Information Visualization*, pages 2–10, 1997.

[10] Tamara Munzner. Exploring large graphs in 3D hyperbolic space. *Computer Graphics and its Applications*, 8(4):18–23, July/August 1998.

[11] Tamara Munzner and Paul Burchard. Visualizing the structure of the world wide web in 3D hyperbolic space. In *Proceedings of the*

VRML '95 Symposium (San Diego, CA, December 13-16, 1995), pages 33–38. ACM SIGGRAPH, 1995.
[12] George Robertson, Jock Mackinlay, and Stuart Card. Cone trees: Animated 3D visualizations of hierarchical information. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 189–194. ACM, April 1991.
[13] George G. Robertson, Stuart K. Card, and Jock D. Mackinlay. Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):57–71, April 1993.
[14] William P. Thurston. *Three-Dimensional Geometry and Topology*, volume 1. Princeton University Press, 1997.

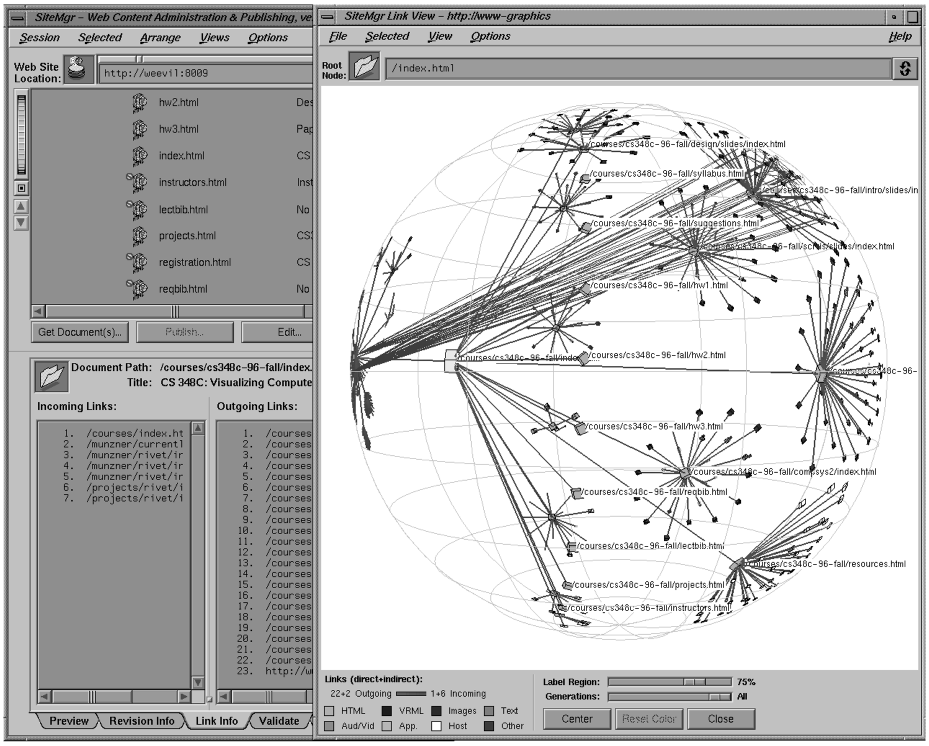

Fig. 1. Site Manager shows the hyperlink structure of the Stanford Graphics Group Web site drawn as a graph in 3D hyperbolic space next to the directory structure of the site drawn with a traditional browser. The entire site has over 14,000 nodes and 29,000 edges, of which only some in the neighborhood of a course homepage are drawn in this snapshot. In addition to the main spanning tree, we draw the non-tree outgoing links from a subtree corresponding to the first day's lecture. The tree is oriented so that ancestors of the course page appear on the left and its descendants grow to the right.

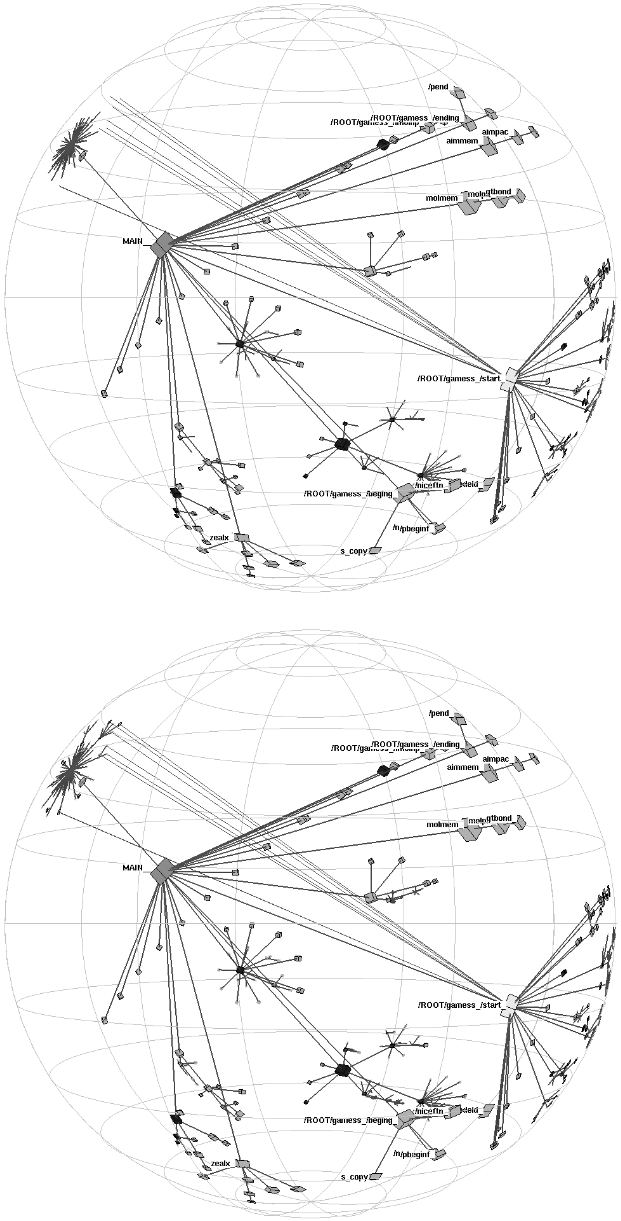

Fig. 2. A function call graph of a small FORTRAN benchmark with 1000 nodes and 4000 edges. Above a single frame has been drawn in 1/20th of a second, while below the entire graph is drawn after the user has stopped moving the mouse. Non-tree links from one of the functions are drawn.