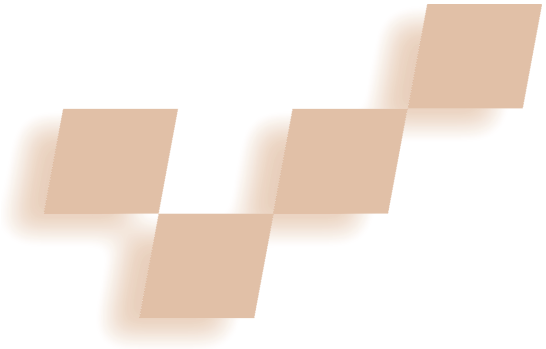


Exploring Large Graphs in 3D Hyperbolic Space

Tamara Munzner
Stanford University

Drawing graphs as nodes connected by links is visually compelling but computationally difficult. Hyperbolic space and spanning trees can reduce visual clutter, speed up layout, and provide fluid interaction.

A graph is a simple, powerful, elegant abstraction with broad applicability in computer science and many related fields. Algorithms that operate on graphs see heavy use in both theoretical and practical contexts. Graphs have a very natural visual representation as nodes and connecting links arranged in space. Seeing this structure explicitly can aid tasks in many domains. Many people automatically sketch such a picture when thinking about small graphs, often including simple annotations.

The pervasiveness of visual representations of small graphs testifies to their usefulness. On the other hand, although many large data sets can be expressed as graphs, few such visual representations exist. What causes this discrepancy? For one thing, graph layout poses a hard problem,[1] one that current tools just can't overcome. Conventional systems often falter when handling hundreds of edges, and none can handle more than a few thousand edges.[2]

However, nonvisual manipulation of graphs with 50,000 edges is commonplace, and much larger instances exist. We can consider the Web as an extreme example of a graph with many millions of nodes and edges. Although many individual Web sites stay quite small, a significant number have more than 20,000 documents. The Unix file system reachable from a single networked workstation might include more than 100,000 files scattered across dozens of gigabytes worth of remotely mounted disk drives.

Computational complexity is not the only reason that software to visually manipulate large graphs has lagged behind software to computationally manipulate them. Many previous graph layout systems have focused on fine-tuning the layout of relatively small graphs in support of polished presentations. A graph drawing system that focuses on the interactive browsing of large graphs can instead target the quite different tasks of browsing and exploration. Many researchers in scientific visualization have recognized the split between explanatory and exploratory goals. This distinction proves equally relevant for graph drawing.

Contribution

This article briefly describes a software system that explicitly attempts to handle much larger graphs than previous systems and support dynamic exploration rather than final presentation. I'll then discuss the applicability of this system to goals beyond simple exploration.

A software system that supports graph exploration should include both a layout and an interactive drawing component. I have developed new algorithms for both layout and drawing—H3 and H3Viewer. A paper from InfoVis 97 contains a more extensive presentation of the H3 layout algorithm.[3] The H3Viewer drawing algorithm remains under development, so this article presents preliminary results.

I have implemented a software library that uses these algorithms. It can handle graphs of more than 100,000 edges by using a spanning tree as the backbone for the layout and drawing algorithms. We draw the graph structure in 3D hyperbolic space to show a large neighborhood around a node of interest. This also allows for quick, fluid changes of the focus point. The H3Viewer drawing algorithm uses both graph-theoretic and view-dependent information to achieve a high guaranteed frame rate.

The library has been incorporated into Site Manager (http://www.sgi.com/software/sitemgr.html), a free Web publishing product from Silicon Graphics aimed at Webmasters and content creators. The first version of Site Manager incorporated only the H3 layout algorithm, while the current release also includes the H3Viewer adaptive drawing function. Users can also access the library from a stand-alone viewer.

Spanning trees

The H3 layout algorithm finds a spanning tree from an input graph and then computes its layout. A span-

0272-1716/98/$10.00 © 1998 IEEE

ning tree touches every node in a graph, but only a subset of the links. In a graph a node can have many incoming links, but in a tree a canonical parent is chosen for each child. We call links that appear in the graph but not in the spanning tree *nontree links*. These links do not affect the layout computation and are drawn for a selected node or nodes only on demand.

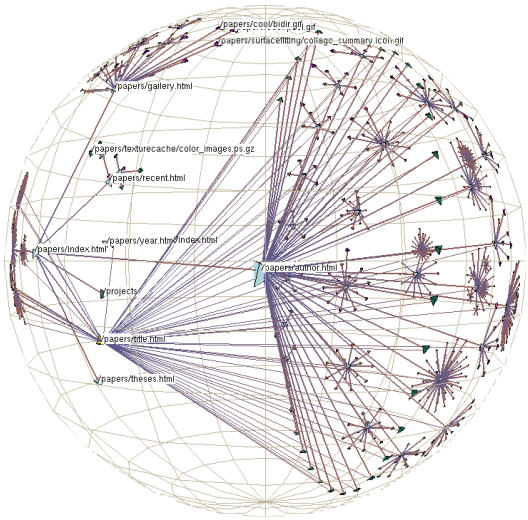
The backbone spanning tree used by the layout and drawing algorithms strongly influences our system's visual impact. As a fallback, we can always find a default spanning tree using a breadth-first search from a root node. However, exploiting a small amount of domain-specific knowledge lets us construct a better spanning tree, one that provides a more useful mental model. If the node identifier has a hierarchical structure, the library will determine parentage based on a best match rather than a breadth-first search. Such structure is available for Web sites like the one shown in Figure 1. In this domain the URL encodes the site's directory structure (often a deliberate choice by the site creator). That directory structure serves to determine which of the incoming hyperlinks to a document should be chosen as its main parent in the spanning tree.

A hierarchical identifier is not trivially available in the case of a function call graph—we must construct it. We can use a combination of compiler analysis and runtime profiling to find the calling procedure responsible for the majority of the child's execution time. Figure 2 shows an example of a graph where this technique was used to construct hierarchical identifiers for the nodes. Software engineers who must modify or optimize unfamiliar code can browse through a call graph's H3 layout of to discover a complex program's structure. Such graphs are notoriously difficult to understand when all the links appear in a 2D nonplanarized layout.

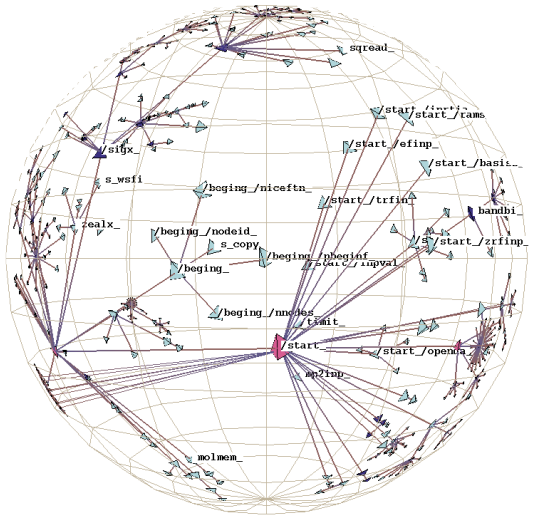
Our reliance on a spanning tree is both the algorithm's strength and its weakness. If we can use domain-specific information to derive a good spanning tree, then our methods work very well up to the limits of main memory. If we must fall back to a breadth-first search for a fully connected graph, the resulting visualization may not convey any useful information. In such cases a spring-force graph drawing system like Frick's Gem3D[4] would serve better in principle. However, in practice this class of methods does not scale—Frick's measure of "large" is only 500 edges.

The key idea is that many nontree graphs exist for which the right spanning tree can provide a useful mental model of the entire structure. Given a good domain-specific way to decide which incoming link should be a node's parent in the spanning tree, our method would work well. Many graphs, densely connected by graph-theoretic standards, fall into this category. In the extreme, trivial case of a tree, our system certainly suits the task well. On the other hand, a bipartite graph will almost certainly result in a misleading picture.

The particular domain-specific methods for finding spanning trees mentioned here offer only one possibility. We want mainly to develop fast, robust layout and drawing algorithms. If and when other researchers determine more appropriate spanning trees for these or other domains, they can explore those trees using the infrastructure presented here.



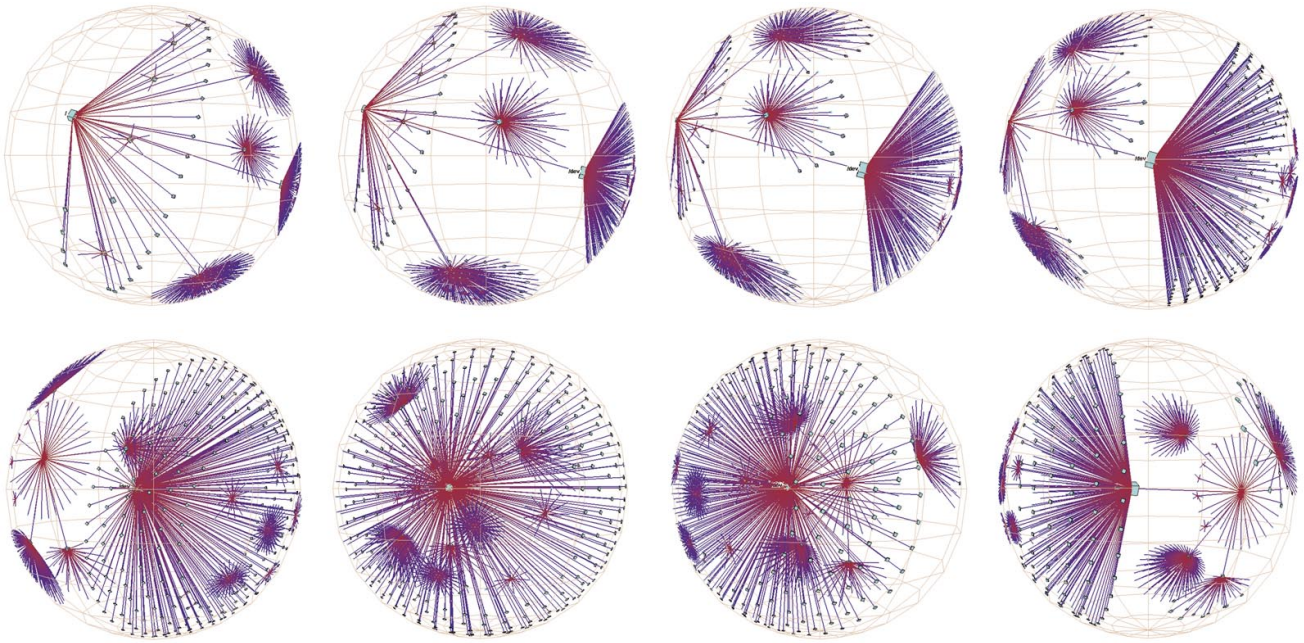
1 Part of the Stanford graphics group Web site drawn as a graph in 3D hyperbolic space. The entire site has more than 20,000 nodes. About 4,000 of them in the neighborhood of the papers archive appear in this frame. In addition to the main spanning tree, the image shows the nontree outgoing links from an index of every paper by title. The tree is oriented so that ancestors of a node appear on the left and its descendants grow to the right.



2 The function call graph structure for a Fortran scientific computing benchmark, where compiler and runtime analysis determined the spanning tree. The node coloring indicates whether a particular global variable was untouched (cyan), referenced (blue), or modified (pink).

Layout

Once we have determined a spanning tree, we must find positions in space for the nodes and edges. Our approach uses the influential cone tree method as a



3 Hyperbolic motion over a 30,000-element Unix file system. Many nodes and edges project to subpixel areas and are not visible. The left column of images shows translation of a node to the center, while the right shows rotation around that node. The rotation clarifies that objects lie inside a ball, not on the surface of a hemisphere. The file system has a strikingly large branching factor when compared with the Web sites in Figure 1 or the call graphs in Figure 2. The directory that approaches the center, /usr/lib, contains a large number of files and subdirectories.

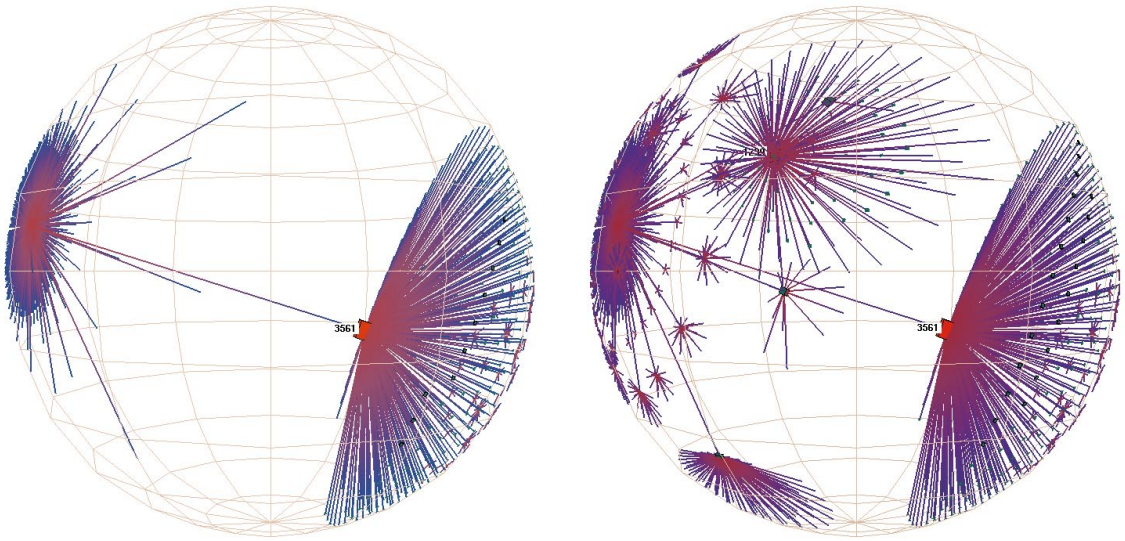
springboard.[5] The standard cone tree method lays out nodes on the linear circumference of a cone's mouth. The H3 layout makes two changes. First, the nodes are laid out on the surface of a hemisphere instead of a linear circumference. We use hemispheres, not full spheres, since the process is recursive. If a child used an entire hemisphere, the back half would intersect the area used by the parental hemisphere. Second, the cone widens to its maximum extent, spanning a full 180 degrees. The cone body proper no longer takes up space but flattens out into a disk at the base of the hemisphere. The child hemispheres lie directly on the parental hemisphere's tangent plane, with no visible intervening cone body.

The layout algorithm requires two passes: a bottom-up pass to estimate the radius needed for each hemisphere to accommodate all of its children, and a top-down pass to place each child node on its parental hemisphere's surface. These steps cannot be combined because we need the radius of the parental hemisphere before we can compute the final position of the children.

Laying out child hemispheres on the surface of their parent introduces the circle packing problem, which has received much attention from the mathematical community.[6] We strike a balance between optimality and simplicity by laying out the children in concentric bands around the hemisphere's pole. The amount of room each node needs is directly proportional to the total number of its descendants. We lay them out in sorted order to avoid wasting space within the bands, and thus the disk at the pole is the node with the most progeny (either direct children or indirect descendants). A full exposition of the layout algorithm appears elsewhere,[3] along with an appendix including a detailed derivation.

Our hemispherical layout is particularly effective because we lay out our tree in a mathematical space having an exponential "amount of room" in the direction of the hemisphere's growth. The area of a hemisphere, $2\pi r$, increases polynomially with respect to its radius in Euclidean space. In hyperbolic space—one of the non-Euclidean geometries—the formula for hemisphere area is $2\pi \sinh^2(r)$. Since the hyperbolic sine and cosine functions (sinh and cosh) are exponential, this space can easily accomodate a layout of an exponential number of nodes (a classic problem in Euclidean tree layout). Hyperbolic space is infinite in extent, just like Euclidean space. However, you can map the entire infinite space into a finite portion of Euclidean space. It might surprise you that you can map an infinite amount of space with "more room" into a finite piece of a space with "less room," but non-Euclidean geometries have many unexpected consequences for Euclidean intuitions. Several standard mappings are used in the mathematical literature.[7] We chose the projective (Klein) model, which supports fast drawing because motions can be expressed as standard 4×4 matrices.[8] Figure 3 shows navigation in 3D hyperbolic space through a Unix file system of more than 31,000 nodes.

While not yet commonplace, hyperbolic space has appeared in the information visualization literature. A hyperbolic browser from Xerox PARC handled trees in two dimensions.[9] The Webviz system from the Geometry Center drew graphs in three dimensions, but the lay-



4 The H3Viewer guaranteed frame rate mechanism ensures interactive response for large graphs, even on slow machines. On the top is a frame drawn in 1/20th of a second during user interaction. On the bottom is a frame filled in by the idle callbacks for a total of 2 seconds after user activity stopped. The graph shows the peering relationships between the Autonomous Systems, which comprise the backbone of the Internet. The 3,000 routers shown here are connected by more than 10,000 edges in the full graph.

out algorithm did not exploit 3D hyperbolic space to its full potential. The amount of information displayed was quite sparse compared to the amount of white space.[10]

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. Carpendale et al.[11] placed nodes in a 3D grid—a 3D volume. In all three examples, the space in which nodes are laid out is a 3D volume. When the dimension of the surrounding space equals the dimension of the node structures, occlusion becomes the overriding issue. We lay out nodes on a surface, which offers 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. With too dense a layout, the leaf nodes near the ball's surface would block our view of the rest of the structure, since we are outside of the ball looking in.

Drawing

The H3Viewer drawing algorithm depends on the number of visible, not total, 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.

A guaranteed frame rate is extremely important for a fluidly interactive user experience. We designed our adaptive drawing algorithm to always maintain a target frame rate even on low-end graphics systems. A high constant frame rate results from drawing only as much of the neighborhood around a center point as the allotted time permits. When the user is idle, the system fills in more of the surrounding scene. A slow graphics system will simply show less of the context surrounding the node of interest during interactive manipulation, as in Figure 4.

The drawing algorithm incorporates knowledge of both the graph structure and the current viewing position. We use the spanning tree's link structure to guide additions to a pool of candidate nodes and the nodes' projected screen area to choose from among the candidates. The largest projected area node from the previous frame serves as a seed for the tree traversal on the next frame.

The current version of the drawing algorithm succeeds in maintaining the target interactive frame rate nearly all the time. However, we have only partially addressed one important issue mentioned in the H3 layout paper.[3] Any single mapping from hyperbolic to Euclidean space permits drawing only a limited number of nodes before the drawing system succumbs to precision problems. The previous drawing system simply truncated nodes beyond this limit. The H3Viewer system will instead compute a remapping from hyperbolic to Euclidean space as necessary when the cumulative error becomes too great. This remapping is a global operation that depends on the total number of nodes in the scene instead of only the visible nodes. When drawing large graphs, this remapping will cause a temporary interruption in the otherwise smooth frame rate or a jump instead of an animated transition. A true solution to this problem would require an incremental mapping algorithm, which falls under the heading of future work.

Discussion

Our layout algorithm's computed overview of the graph structure provided by the geometry of nodes and links offers a way for a user to explore a graph too large to manipulate with traditional methods. However, an interactive graph exploration system can offer a user more than a global overview. Our layout and drawing system has applicability for the following tasks:

- Scaffolding for attributes
- Local orientation
- Context of part in whole
- Graph as index

Scaffolding for attributes

When used as a scaffolding, the structure can show static or dynamic attributes. Many graph drawing systems support color and line-width coding, text labels, and filtering—as does our own. These filtering and coding capabilities can be very powerful when used to show dynamic data. For instance, in the new Site Manager release a Web site's traffic logs can help show the paths taken by Web users. A hit from one page to another is shown by briefly highlighting those nodes and the link between them, for a laser-like effect.

Local orientation

Our drawing and layout approaches also support finding interesting places when browsing through an unfamiliar graph. When the user clicks on a node, it is highlighted and undergoes an animated transition to the center of the sphere. Animation proves critical in helping the user maintain a sense of context.

The transition includes both a translational and a rotational component. Thus, when a node reaches the origin, its ancestors always appear on its left and its descendants on the right. This "butterfly" configuration provides a canonical local orientation and also serves to minimize occlusion of both nodes and their text labels. If the structure were aligned with the principal axes of the window, the text labels would often occlude each other, so we add a slight tilt. Our layout algorithm always places the node with the most descendants at the "pole" of the hemisphere along the same axis as the incoming link from the parent node. Thus a simple and effective navigation strategy for finding potentially interesting complexity is to click on polar nodes after their parents move into their canonical orientation.

In our drawing algorithm, we explicitly chose to draw the links into and out of a node, even if we don't have time to draw the node at the other end. The presence of an unterminated link during motion hints to the user of something interesting in that direction. This situation occurs frequently when drawing nontree links, whose other end often lies far enough away from the center that the drawing loop ends before the terminating node can be drawn.

Context of part in whole

Hyperbolic space very effectively presents a large area around a focus node. For instance, in Figure 1 the user can see enough of the distant subtrees to identify dense and sparse ones. The destinations of nontree links are distorted, but the rough sense of their destination helps the user construct and maintain a mental model of the larger graph structure. Figure 3 shows how the details become clear in a smooth transition when an area of the structure moves towards the center. The context shows up on several levels: the local parent-child relationships of the spanning tree, the nontree links between disparate nodes in the graph, and the rough structure far away from the current focus of interest.

Graph as index

Although you could use the H3Viewer to create a stand-alone application, it is most effective when integrated with other tools. If a graph viewer is one of several views that all support linked selection and filtering, the graph structure becomes one way to index the information. Such an index proves useful for selecting items in a known graph in addition to discovering patterns in an unfamiliar one. In the Site Manager system, we tightly integrate the 3D hyperbolic browser with a 2D file browser and a search window that shows all the matches of some string as a 1D list. The user can choose the appropriate display—the one having the appropriate attributes for the task at hand.

Conclusion

Our implementation can handle graphs two orders of magnitude larger than previous systems by manipulating a backbone spanning tree instead of the full graph. Carrying out both layout and drawing in 3D hyperbolic space lets us 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. ■

Acknowledgments

I appreciate the efforts of the following people and organizations in collecting the data used here: function call graph data from Anwar Ghuloum of the Stanford University Intermediate Format (SUIF) compilers group; Autonomous Systems data from Hans-Werner Braun of the National Laboratory for Applied Network Research (NLANR) and David M. Meyer of the University of Oregon Route Views Project. Thanks to François Guimbretière and Pat Hanrahan for their advice and ideas. I gratefully acknowledge the efforts of the rest of the Site Manager team at Silicon Graphics: Ken Kershner, Greg Ferguson, Alan Braverman, Donna Scheele, and Doug O'Morain. This work was supported in part by Silicon Graphics, the National Science Foundation (NSF) Graduate Research Fellowship Program, and Advanced Research Projects Agency (ARPA).

References

1. F.J. Brandenburg, "Nice Drawing of Graphs are Computationally Hard," in *Visualization in Human-Computer Interaction*, Lecture Notes in Computer Science 439, P. Gorney

and M.J. Tauber, eds., Springer-Verlag, Berlin, 1988, pp. 1-15.
2. G. Di Battisata et al., "Annotated Bibliography on Graph Drawing Algorithms," *Computational Geometry: Theory and Applications*, Vol. 4, No. 5, 1994, pp. 235-282.
3. T. Munzner, "H3: Laying out Large Directed Graphs in 3D Hyperbolic Space," *Proc. 1997 IEEE Symp. on Interactive 3D Graphics*, IEEE Computer Society Press, Los Alamitos, Calif., 1997, pp. 2-10.
4. I. Bruss and A. Frick, "Fast Interactive 3D Graph Visualization," *Proc. Graph Drawing 95*, Lecture Notes in Computer Science 1027, Springer-Verlag, Berlin, 1995, pp. 99-110.
5. G. Robertson, J. Mackinlay, and S. Card, "Annotated 3D Visualizations of Hierarchical Information," *Proc. ACM SIGCHI Conf. on Human Factors in Computing Systems*, ACM Press, New York, April 1991, pp. 189-194.
6. J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices, and Groups*, Springer-Verlag, Berlin, 1988.
7. G.E. Martin, *The Foundations of Geometry and the Non-Euclidean Plane*, Springer-Verlag, Berlin, 1975.
8. M. Phillips and C. Gunn, "Visualizing Hyperbolic Space: Unusual Uses of 4 × 4 Matrices," *1992 Symp. on Interactive 3D Graphics*, special issue of Computer Graphics, Vol. 25, ACM Siggraph, ACM Press, New York, 1992 pp. 209-214.
9. J. Lamping, R. Rao, and P. Pirolli, "A Focus + Content Technique Based on Hyperboic Geometry for Viewing Large Hierarchies," *Proc. ACM SiGCHI Conf. on Human Factors in Computing Systems*, ACM Press, New York, May 1995, pp. 401-408.
10. T. Munzner and P. Burchard, "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space," *Proc. VRML 95 Symp.*, ACM Siggraph, ACM Press, New York, 1995, pp. 33-38.
11. M.S.T. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia, "Extending Distortion Viewing from 2D to 3D," *IEEE Computer Graphics and Applications*, Vol. 17, No. 4, July/August 1997, pp. 42-51.

Tamara Munzner *is currently a PhD candidate at Stanford University, where she received a BS in computer science in 1991. In the intervening years she was a member of the technical staff at the Geometry Center, a mathematical visualization research group at the University of Minnesota. Her technical interests include graph drawing, information visualization, mathematical visualization, interactive 3D graphics systems, and pedagogical video creation.*

Contact Munzner at the Department of Computer Science, 360 Gates Building 3B, Stanford University, Stanford, CA 94305, e-mail munzner@cs.stanford.edu, http://graphics.stanford.edu/~munzner.