A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies.

John Lamping, Ramana Rao, and Peter Pirolli

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
<lamping, rao, pirolli>@parc.xerox.com

ABSTRACT

We present a new focus+context (fisheye) technique for visualizing and manipulating large hierarchies. Our technique assigns more display space to a portion of the hierarchy while still embedding it in the context of the entire hierarchy. The essence of this scheme is to lay out the hierarchy in a uniform way on a hyperbolic plane and map this plane onto a circular display region. This supports a smooth blending between focus and context, as well as continuous redirection of the focus. We have developed effective procedures for manipulating the focus using pointer clicks as well as interactive dragging, and for smoothly animating transitions across such manipulation. A laboratory experiment comparing the hyperbolic browser with a conventional hierarchy browser was conducted.

KEYWORDS: Hierarchy Display, Information Visualization, Fisheye Display, Focus+Context Technique.

INTRODUCTION

In the last few years, Information Visualization research has explored the application of interactive graphics and animation technology to visualizing and making sense of larger information sets than would otherwise be practical [11]. One recurring theme has been the power of focus+context techniques, in which detailed views of particular parts of an information set are blended in some way with a view the of the overall structure of the set. In this paper, we present a new technique, called the hyperbolic browser, for visualizing and manipulating large hierarchies.

The hyperbolic browser, illustrated in Figure 1, was originally inspired by the Escher woodcut shown in Figure 2. Two salient properties of the figures are: first that components diminish in size as they move outwards, and second that there is an exponential (devilish) growth in the number of components. These properties—"fisheye" distortion and the ability to uniformly embed an exponentially growing structure—are the aspects of this construction (the Poincaré mapping of the hyperbolic plane) that originally attracted our attention.

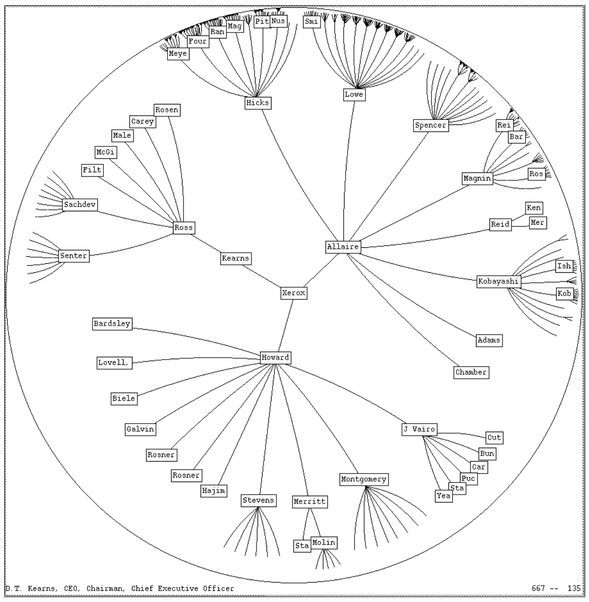

Figure 1: An organization chart.

The hyperbolic browser initially displays a tree with its root at the center, but the display can be smoothly transformed to bring other nodes into focus, as illustrated in Figure 3. In all cases, the amount of space available to a node falls off as a continuous function of its distance in the tree from the point in the center. Thus the context always includes several generations of parents, siblings, and children, making it easier for the user to explore the hierarchy without getting lost.

The hyperbolic browser supports effective interaction with much larger hierarchies than conventional hierarchy viewers and complements the strengths of other novel tree browsers. In a 600 pixel by 600 pixel window, a standard 2-d hierarchy browser can typically display 100 nodes (w/ 3 character text strings). The hyperbolic browser can display 1000 nodes of which about the 50 nearest the focus can show from 3 to

†Reprinted From: Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, Denver, May 1995, ACM

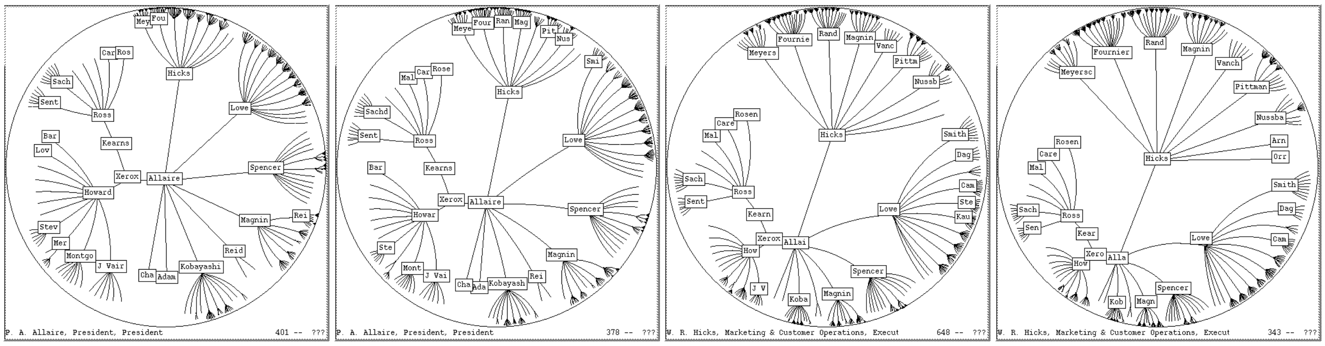

Figure 3: Changing the focus.

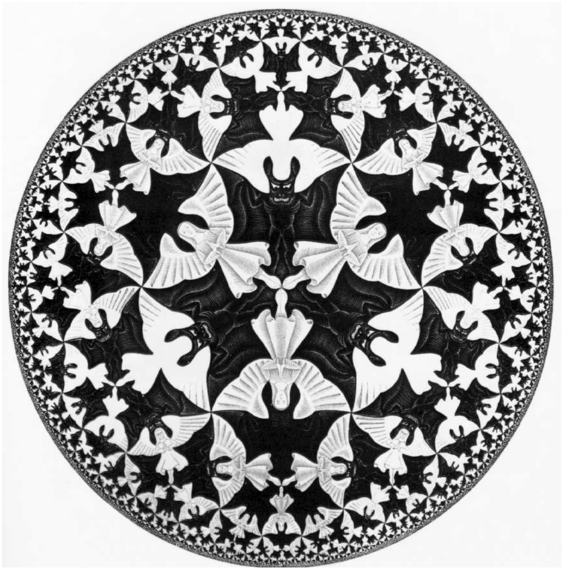

Figure 2: Original inspiration for the hyperbolic tree browser. Circle Limit IV (Heaven and Hell), 1960, (c) 1994 M.C. Escher / Cordon Art – Baarn – Holland. All rights reserved. Printed with permission.

dozens of characters of text. Thus the hyperbolic browser can display up to 10 times as many nodes while providing more effective navigation around the hierarchy. The scale advantage is obtained by the dynamic distortion of the tree display according to the varying interest levels of its parts.

Our approach exploits hyperbolic geometry [2, 9]. The essence of the approach is to lay out the hierarchy on the hyperbolic plane and map this plane onto a circular display region. The hyperbolic plane is a non-Euclidean geometry in which parallel lines diverge away from each other. This leads to the convenient property that the circumference of a circle on the hyperbolic plane grows exponentially with its radius, which means that exponentially more space is available with increasing distance. Thus hierarchies—which tend to expand exponentially with depth—can be laid out in hyperbolic space in a uniform way, so that the distance (as measured in the hyperbolic geometry) between parents, children, and siblings is approximately the same everywhere in the hierarchy.

While the hyperbolic plane is a mathematical object, it can be mapped in a natural way onto the unit disk, which provides a means for displaying it on an ordinary (Euclidean) display. This mapping displays portions of the plane near the origin using more space than other portions of the plane. Very remote parts of the hyperbolic plane get miniscule amounts of space near the edge of the disk. Translating the hierarchy on the hyperbolic plane provides a mechanism for controlling which portion of the structure receives the most space without compromising the illusion of viewing the entire hyperbolic plane. We have developed effective procedures for manipulating the focus using pointer dragging and for smoothly animating transitions across such manipulation.

We have implemented versions of the hyperbolic browser that run on Unix/X and on Macintoshes. We conducted an experiment with 4 subjects to compare the hyperbolic tree browser with a conventional browser on a node location task. Though no statistically significant performance difference was identified, a strong preference for the hyperbolic tree browser was established and a number of design insights were gained.

PROBLEM AND RELATED WORK

Many hierarchies, such as organization charts or directory structures, are too large to display in their entirety on a computer screen. The conventional display approach maps all the hierarchy into a region that is larger than the display and then uses scrolling to move around the region. This approach has the problem that the user can't see the relationship of the visible portion of the tree to the entire structure (without auxiliary views). It would be useful to be able to see the entire hierarchy while focusing on any particular part so that the relationship of parts to the whole can be seen and so that focus can be moved to other parts in a smooth and continuous way.

A number of focus+context display techniques have been introduced in the last fifteen years to address the needs of many types of information structures [7, 15]. Many of these focus+context techniques, including the document lens [13], the perspective wall [8], and the work of Sarkar et al [14, 16], could be applied to browsing trees laid out using conventional 2-d layout techniques. The problem is that there is no satisfactory conventional 2-d layout of a large tree, because of its exponential growth. If leaf nodes are to be given adequate spacing, then nodes near the root must be placed very far apart, obscuring the high level tree structure, and leaving no nice way to display the context of the entire tree.

The Cone Tree[12] modifies the above approach by embedding the tree in a three dimensional space. This embedding of the tree has joints that can be rotated to bring different parts of the tree into focus. This requires currently expensive 3D animation support. Furthermore, trees with more than approximately 1000 nodes are difficult to manipulate. The hyperbolic browser is two dimensional and has relatively modest computational needs, making it potentially useful on a broad variety of platforms.

Another novel tree browsing technique is treemaps [5] which allocates the entire space of a display area to the nodes of the tree by dividing the space of a node among itself and its descendants according to properties of the node. The space allocated to each node is then filled according to the same or other properties of the node. This technique utilizes space efficiently and can be used to look for values and patterns amongst a large collection of values which agglomerate hierarchically, however it tends to obscure the hierarchical structure of the values and provides no way of focusing on one part of a hierarchy without losing the context.

Some conventional hierarchy browsers prune or filter the tree to allow selective display of portions of the tree that the user has indicated. This still has the problem that the context of the interesting portion of the tree is not displayed. Furnas [3] introduced a technique whereby nodes in the tree are assigned an interest level based on distance from a focus node (or its ancestors). Degree of interest can then be used to selectively display the nodes of interest and their local context. Though this technique is quite powerful, it still does not provide a solution to the problem of displaying the entire tree. In contrast, the hyperbolic browser is based on an underlying geometry that allows for smooth blending of focus and context and continuous repositioning of the focus.

Bertin[1] illustrates that a radial layout of the tree could be uniform by shrinking the size of the nodes with their distance from the root. The use of hyperbolic geometry provides an elegant way of doing this while addressing the problems of navigation. The fractal approach of Koike and Yoshihara [6] offers a similar technique for laying out trees. In particular, they have explored an implementation that combines fractal layout with Cone Tree-like technique. The hyperbolic browser has the benefit that focusing on a node shows more of the node's context in all directions (i.e. ancestors, siblings, and descendants). The fractal view has a more rigid layout (as with other multiscale interfaces) in which much of this context is lost as the viewpoint is moved to lower levels of the tree.

There have been a number of projects to visualize hyperbolic geometry, including an animated video of moving through hyperbolic space [4]. The emphasis of the hyperbolic browser is a particular exploitation of hyperbolic space for information visualization. We don't expect the user to know or care about hyperbolic geometry.

HYPERBOLIC BROWSER BASICS

The hyperbolic browser replaces the conventional approach of laying a tree out on a Euclidean plane by doing *layout* on the hyperbolic plane and then *mapping* to the unit disk (which is straightforwardly mapped to the display). *Change of focus* is handled by performing a rigid transformation of the hyperbolic plane, moving the laid out tree in the process. Thus layout is only performed once. Space for displaying *node information* is also computed during layout and automatically transformed with each change of focus.

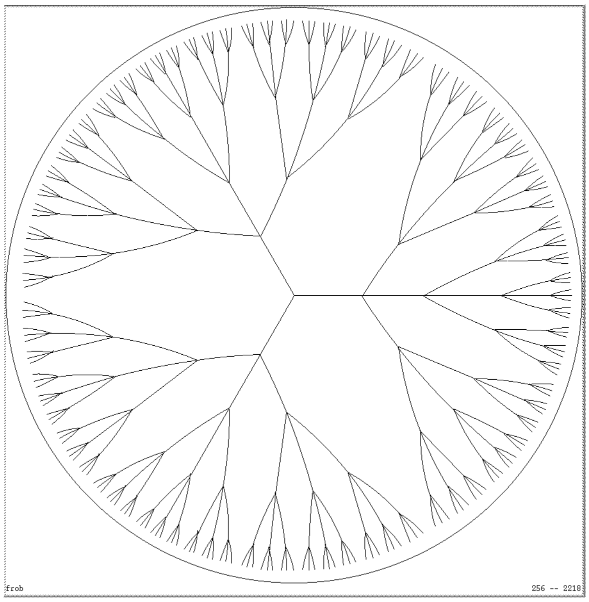

Figure 4: A uniform tree of depth 5 and branching factor 3 (364 nodes).

The implementation of points and transformations on the hyperbolic plane is briefly discussed in the appendix. The rest of this section presumes an implementation of the hyperbolic plane and discusses higher level issues.

Layout

Laying a tree out in the hyperbolic plane is an easy problem, because the circumference and area of a circle grow exponentially with its radius. There is lots of room. We use a recursive algorithm that lays out each node based on local information. A node is allocated a wedge of the hyperbolic plane, angling out from itself, to put its descendants in. It places all its children along an arc in that wedge, at an equal distance from itself, and far enough out so that the children are some minimum distance apart from each other. Each of the children then gets a sub-wedge for its descendants. Because of the way parallel lines diverge in hyperbolic geometry, each child will typically get a wedge that spans about as big an angle as does its parent's wedge, yet none of the children's wedges will overlap.

The layout routine navigates through the hyperbolic plane in terms of operations, like moving some distance or turning through some angle, which are provided by the hyperbolic plane implementation.

Figure 4 shows what the layout of a uniform tree looks like. Notice how the children of each node span about the same angle, except near the root, where a larger wedge was available initially. To get a more compact layout for non-uniform trees, we modify this simple algorithm slightly, so that siblings that

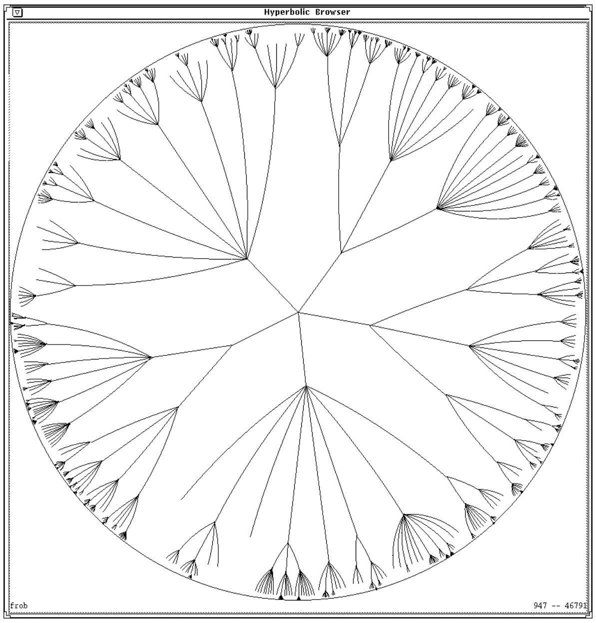

Figure 5: The initial layout of a tree with 1004 nodes using a Poisson distribution for number of children. The origin of the tree is in the center.

themselves have lots of children get a larger wedge than siblings that don't (the wedge size grows logarithmically). This effect can be seen in Figure 5, where, for example, the five children of the root get different amounts of space. This tends to decrease the variation of the distances between grandchildren and their grandparent.

Another option in layout (in contrast to all examples so far illustrated) is to use less than the entire 360 degree circle for spreading out the children of the root node. With this option, children of the root could all be put in one direction, for example to the right or below, as in conventional layouts. An example of this option, discussed below, appears in Figure 8.

Mapping and Representation

Once the tree has been laid out on the hyperbolic plane, it must be mapped in some way to a 2-d plane for display. (We can barely imagine the hyperbolic plane, not to mention see it.) There are two canonical ways of mapping the hyperbolic plane to the unit disk. In both of them, one vicinity in the hyperbolic plane is in focus at the center of the disk while the rest of the hyperbolic plane fades off in a perspective-like fashion toward the edge of the disk, as we desire. We use the conformal mapping, or Poincaré model, which preserves angles but distorts lines in the hyperbolic space into arcs on the unit disk, as can be seen in the figures. The other possibility, the projective mapping, or Klein model, takes lines in the hyperbolic plane to lines in the unit disk, but distorts angles. You can't have it both ways.

We tried the Klein model. But points that are mapped to near the edge by the Poincaré model get mapped almost right on the edge by the Klein model. As a result, nodes more than a link or two from the node in focus get almost no screen real-estate, making it very hard to perceive the context.

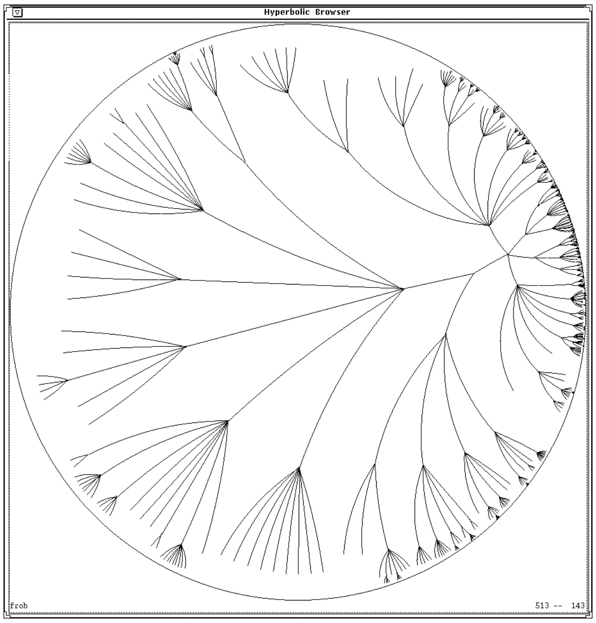

Figure 6: A new focus.

Change of Focus

The user can change focus either by clicking on any visible point to bring it into focus at the center, or by dragging any visible point interactively to any other position. In either case, the rest of the display transforms appropriately. Regions that approach the center become magnified, while regions that were in the center shrink as they are moved toward the edge. Figure 6 shows the same tree as Figure 5 with a different focus. The root has been shifted to the right, putting more focus on the nodes that were toward the left.

Changes of focus are implemented as rigid transformations of the hyperbolic plane that will have the desired effect when the plane is mapped to the display; there is never a need to repeat the layout process. A change of focus to a new node, for example, is implemented by a translation in the hyperbolic plane that moves the selected node to the location that is mapped to the center of the disk.

To avoid loss of floating point precision across multiple transformations, we compose successive transformations into a single cumulative transformation, which we then apply to the positions determined in the original layout. Further, since we only need the mapped positions of the nodes that will be displayed, the transformation only needs to be computed for nodes whose display size will be at least a screen pixel. This yields a constant bound on redisplay computation, no matter how many nodes are in the tree. And, the implementation of translation can be fairly efficient; we require about 20 floating point operations to translate a point, comparable to the cost of rendering a node on the screen.

Node Information

Another property of the Poincaré projection is that circles on the hyperbolic plane are mapped into circles on the Euclidean disk, though they will shrink in size the further they are from the origin. We exploit this property by calculating a circle

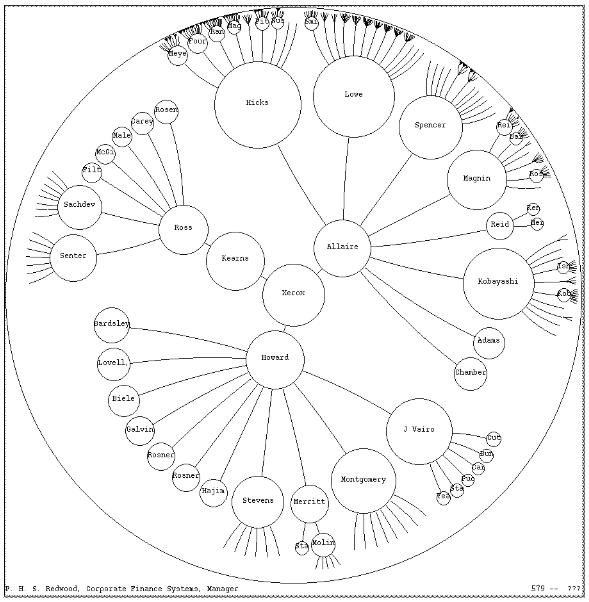

Figure 7: The display regions of nodes.

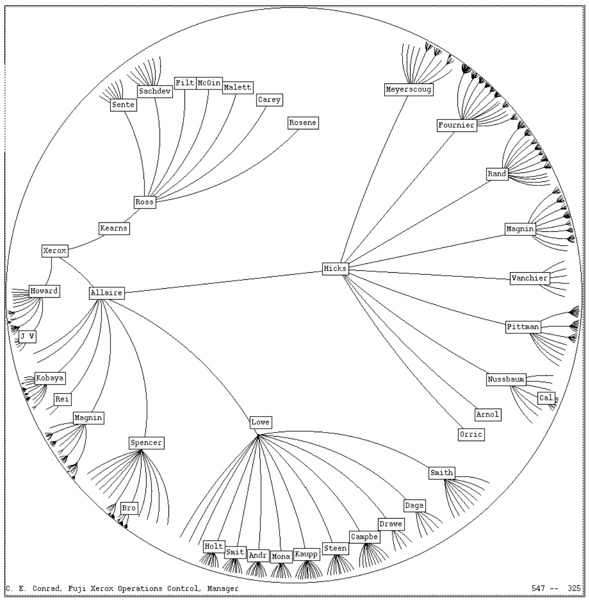

Figure 8: Putting children toward the right.

in the hyperbolic plane around each node that is guaranteed not to intersect the circle of any other node. When those circles are mapped onto the unit disk they provide a circular display region for each node of the tree in which to display a represenation of the node. This can be combined with a facility that uses different representations for nodes depending on the amount of real space they receive. Figure 7 shows the same tree as Figure 1, with the display region of each node indicated.

PRESERVING ORIENTATION

Orientation presents an interesting issue for the hyperbolic browser, because things tend to get rotated. For example, most nodes rotate on the display during a pure translation. There is a line that doesn't rotate, but the farther nodes are on the display from that line, the more they rotate. This can be seen in the series of frames in Figure 3. The node labeled "Lowe", for example, whose children fan out to the upper right in the top frame ends up with its children fanning out to the right in the bottom frame. These rotations are reasonably intuitive for translations to or from the origin. But if drags near the edge of the disk are interpreted as translations between the the source and the destination of the drag, the display will do a counter-intuitive pirouette about the point being dragged.

There is a fundamental property of hyperbolic geometry that is behind this and that also causes another problem. In the usual Euclidean plane, if some graphical object is dragged around, but not rotated, then is always keeps its original orientation—not rotated. But this is *not true* in the hyperbolic plane. A series of translations forming a closed loop, each preserving the orientation along the line of translation, will, in general, cause a rotation. (In fact the amount of rotation is proportional to the area of the closed loop and is in the opposite direction to the direction the loop was traversed.) This leads to the counter-intuitive behavior that a user who browses around the hierarchy can experience a different orientation each time they revisit some node, even though all they did was translations.

We address both of these problems by interpreting the user's manipulations as a combination of both the most direct translation between the points the user specifies and an additional rotation around the point moved, so that the manipulations and their cumulative effects are more intuitive. From the user's perspective, drags and clicks move the point that the user is manipulating where they expect. The additional rotation also appears natural, as it is designed to preserve some other property that the user expects. The user need not even be particularly aware that rotation is being added.

We have found two promising principles for adding rotations. In one approach, rotations are added so that the original root node always keeps its original orientation on the display. In particular, the edges leaving it always leave in their original directions. Preserving the orientation of the root node also means that the node currently in focus also has the orientation it had in the original image. The transformations in the examples presented so far all worked this way. It seems to give an intuitive behavior both for individual drags and for the cumulative effect of drags.

The other approach we have taken is to explicitly not preserve orientation. Instead, when a node is clicked on to bring it to focus, the display is rotated to have its children fan out in a canonical direction, such as to the right. This is illustrated in Figure 8 and also in the animation sequence in Figure 9. This approach is aided when the children of the root node are all laid out on one side of that node, as also illustrated in the two figures, so that the children of the root node can also fan out in the canonical direction when it is in focus.

ANIMATED TRANSITIONS

As demonstrated by recent work on information visualizations, animated transitions between different views of a structure can maintain object constancy and help the user assimilate

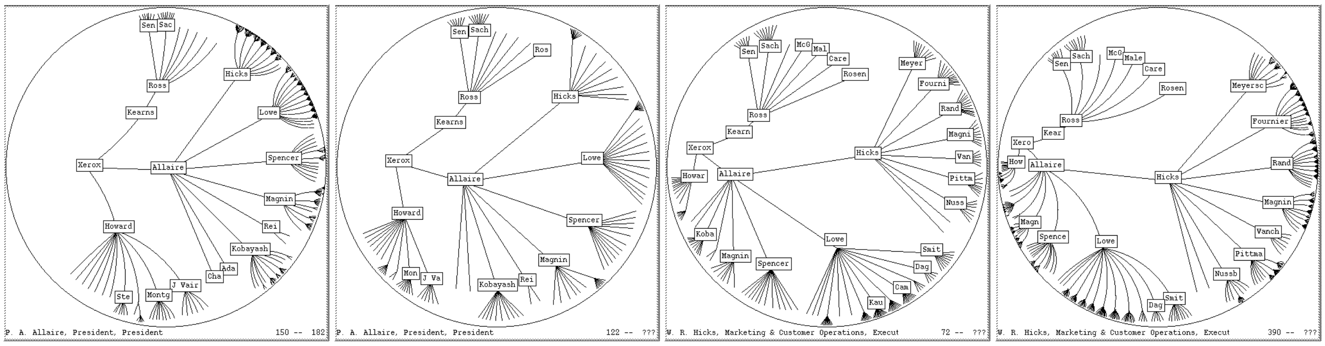

Figure 9: Animation with compromised rendering.

the changes across views. The smooth continuous nature of the hyperbolic plane allows for performing smooth transitions of focus by rendering appropriate intermediate views.

Animation sequences are generated using the so-called "nth-root" of a transition transformation, i.e. the rigid transformation that applied n times will have the same effect as the original. Successive applications of the "nth-root" generate the intermediate frames. The sequences in Figure 3 and Figure 9 were generated this way.

Responsive display performance is crucial for animation and interactive dragging. This can be a problem for large hierarchies on standard hardware. We achieve quick redisplay by compromising on display quality during motion. These compromises provide options for use in a system that automatically adjusts rendering quality during animation, e.g. the Information Visualizer governor [10] or Pacers [17]. Fortunately, there are compromises that don't significantly affect the sense of motion. Figure 9 shows an animation sequence with the compromises active in the intermediate frames. Unless specifically looked for, the compromises typically go unnoticed during motion.

One compromise is to draw less of the fringe. Even the full quality display routine stops drawing the fringe once it gets below one pixel resolution. For animation, the pruning can be strengthened, so that descendants of nodes within some small border inside the edge of the disk are not drawn. This tremendously increases display performance, since the vast majority of nodes are very close to the edge. But it doesn't significantly degrade perceptual quality for a moving display, since those nodes occupy only a small fraction of the display, and not a part that the user is typically focusing on.

The other compromise is to draw lines, rather than arcs, which are expensive in the display environments we have been using. While arcs give a more pleasing and intuitive static display, they aren't as important during animation. This appears to be the case both because the difference between arcs and lines isn't as apparent in the presence of motion, and because the user's attention during motion tends to be focused near the center of the display, where the arcs are already almost straight.

One other possible compromise is to drop text during animation. We found this to be a significant distraction, however. And text display has not been a performance bottleneck.

EVALUATION AND FUTURE WORK

A laboratory experiment was conducted to contrast the hyperbolic browser against a conventional 2-d scrolling browser with a horizontal tree layout. Our subjects preferred the hyperbolic browser in a post-experimental survey, but there was no significant difference between the browsers in performance times for the given task, which involved finding specific node locations. The study has fueled our iterative design process as well as highlighted areas for further work and evaluation.

The two browsers in the study support mostly the same user operations. "Pointing" provided feedback on the node under the cursor in a feedback area at the bottom of window. "Clicking" moved a point to the center. "Grabbing" any visible point allowed interactive dragging of the tree within the window. The 2-d scrolling browser provides conventional scrollbars as well.

The experiment was based on the task of locating and "double-clicking" on particular nodes in four World-Wide-Web hierarchies identified by their URLs (the application intent being that a Web browser would jump to that node). Though this particular task and application were adequate for a rough baseline evaluation, there are problematic aspects. Typically, this task would better be supported by query-by-name or even an alphabetical listing of the nodes. Furthermore the WWW hierarchy (based on breadth-first flattening of the network) contained many similarly-named nodes and the hierarchy wasn't strongly related to a semantic nesting.

After pilot trials, we added to both browsers a feature to rotate the names of children of a pointed-to node through the feedback area and then jump to the current child. We also added a toggleable "long text" mode in which all nodes beyond an allocated space threshold disregard their boundaries and display up to 25 characters. Despite the overlapping of the text, this leads to more text being visible and discernible on the screen at once (see Figure 10).

The experiment used a within-subject design with four subjects, and tested for the effects of practice. We found no significant difference in time or number of user actions in performing the tasks across the browsers. There was a significant practice effect in which practice produced a decrease in the number of user actions required to perform the task for both

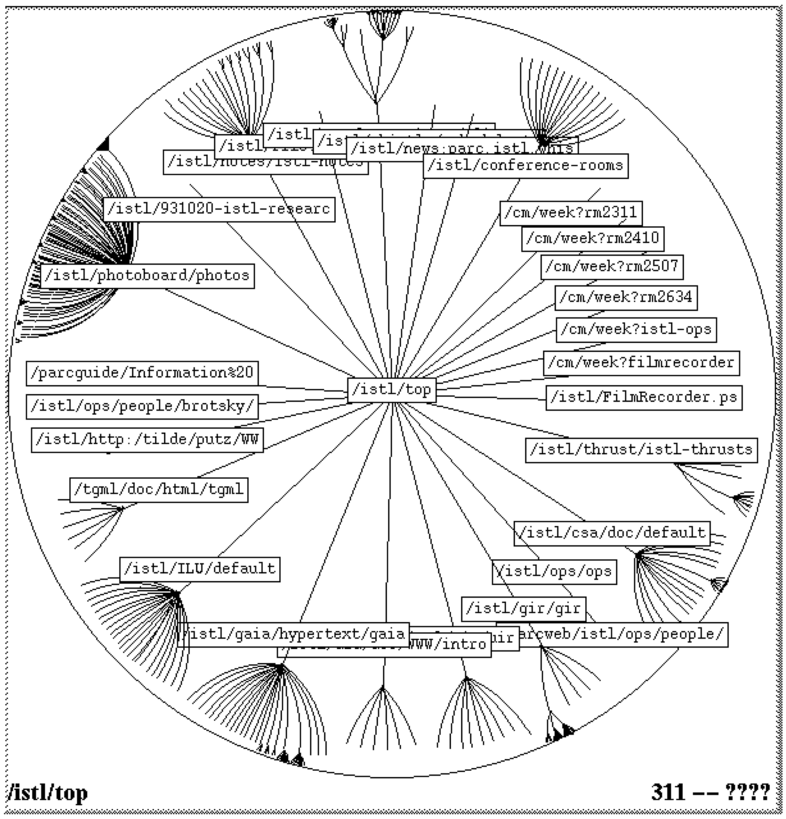

Figure 10: Hyperbolic browser in long text mode in World Wide Web hierarchy utilized in laboratory experiment.

browsers, but there was no practice effect on task completion time for either browser. These practice effects did not differ significantly between the browsers.

Our post-experimental survey showed that all four subjects preferred the hyperbolic browser for "getting a sense of the overall tree structure" and "finding specific nodes by their titles," as well as "overall." In addition, specific survey questions and our observations identified relative strengths and weaknesses of the hyperbolic browser. Three of the subjects liked the ability to see more of the nodes at once and two mentioned the ability to see various structural properties and a better use of the space.

The amount of text that the hyperbolic browser displays was a problem. The experimental task was particularly sensitive to this problem because of the length and overlap of URLs, and the ill-structured nature of the WWW hierarchy. Though the long text mode was introduced before the study, none of the subjects used this feature during the study, preferring to point at the parent and then rapidly rotate the children through the much larger feedback area.

Problem areas mentioned by one subject were that the hyperbolic browser provide a weaker sense of directionality of links and also of location of a node in the overall space (because shapes changed). Layout in a canonical direction as shown in Figure 8 addresses the first of these problems, but may worsen the second. In particular, for applications in which access to ancestors or to the root node is particularly important, this layout makes it easy to find and navigate toward these nodes.

A number of refinements may increase the value of the browser for navigating and learning hierarchies. For example, landmarks can be created in the space by utilizing color and other graphical elements (e.g. we painted http, gopher, and ftp links using different colors). Other possibilities are providing a visual indication of where there are nodes that are invisible because of the resolution limit, using a ladder of multiscale graphical representations in node display regions, and supporting user control of trade-off between node display region size and number of visible nodes (i.e. packing). The effective use of these variations are likely to be application or task dependent and so best explored in such a design context.

CONCLUSION

Hyperbolic geometry provides an elegant solution to the problem of providing a focus+context display for large hierarchies. The hyperbolic plane has the room to lay out large hierarchies, and the Poincaré map provides a natural, continuously graded, focus+context mapping from the hyperbolic plane to a display. The hyperbolic browser can handle arbitrarily large hierarchies, with a context that includes as many nodes as are included by 3d approaches and with modest computational requirements. Our evaluation study suggested this technique could be valuable, and has identified issues for further work. We believe that the hyperbolic browser offers a promising new addition to the suite of available focus+context techniques.

ACKNOWLEDGEMENTS

We would like to thank the reviewers, our four subjects, and the colleagues who made suggestions for the prototype. Xerox Corporation is seeking patent protection for technology described in this paper.

REFERENCES

1. J Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983.
2. H. S. M. Coxeter. *Non-Euclidean Geometry*. University of Toronto Press, 1965.
3. George W. Furnas. Generalized fisheye views. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 16–23. ACM, April 1986.
4. C. Gunn. Visualizing hyperbolic space. In *Computer Graphics and Mathematics*, pages 299–311. Springer-Verlag, October 1991.
5. B. Johnson and B. Shnedierman. Tree-maps: A space-filling approach to the visualization of hierarchical information. In *Visualization 1991*, pages 284–291. IEEE, 1991.
6. Hideki Koike and Hirotaka Yoshihara. Fractal approaches for visualizing huge hierarchies. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*. IEEE, 1993.
7. Y.K. Leung and M.D.Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, June 1994.
8. J. D. Mackinlay, G. G. Robertson, and S. K. Card. The perspective wall: Detail and context smoothly integrated. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 173–179. ACM, April 1991.
9. E. E. Moise. *Elementary Geometry from an Advanced Standpoint*. Addison-Wesley, 1974.
10. G. G. Robertson, S. K. Card, and J. D. Mackinlay. The cognitive coprocessor architecture for interactive user interfaces. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 10–18. ACM Press, Nov 1989.
11. G. G. Robertson, S. K. Card, and J. D. Mackinlay. Information visualization using 3d interactive animation. *Communications of the ACM*, 36(4), 1993.
12. G. G. Robertson, J. D. Mackinlay, and S. K. 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 and J. D. Mackinlay. The document lens. In *Proceedings of the ACM Symposium on User Interface Software and Technology*. ACM Press, Nov 1993.
14. Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 83–91. ACM, April 1992.
15. Manojit Sarkar and Marc H. Brown. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, December 1994.
16. Manojit Sarkar, Scott Snibbe, and Steven Reiss. Stretching the rubber sheet: A metaphor for visualizing large structure on small screen. In *Proceedings of the ACM Symposium on User Interface Software and Technology*. ACM Press, Nov 1993.
17. Steven H. Tang and Mark A. Linton. Pacers: Time-elastic objects. In *Proceedings of the ACM Symposium on User Interface Software and Technology*. ACM Press, Nov 1993.

APPENDIX: IMPLEMENTING HYPERBOLIC GEOMETRY

We use the Poincaré model for our underlying implementation of the hyperbolic plane, because that makes translation between the underlying representation and screen coordinates trivial. We represent a point in hyperbolic space by the corresponding point in the unit disk, which is represented by a floating point complex number of magnitude less than 1. Rigid transformations of the hyperbolic plane become circle preserving transformations of the unit disk.

Any such transformation can be expressed as a complex function of z of the form

$$z_t = \frac{\theta z + P}{1 + \overline{P} z}$$

Where P and θ are complex numbers, $|P| < 1$ and $|\theta| = 1$, and $\overline{P}$ is the complex conjugate of P. This transformation indicates a rotation by θ around the origin followed by moving the origin to P (and $-P$ to the origin).

The composition of two transformations can be computed by:

$$P = \frac{\theta_2 P_1 + P_2}{\theta_2 P_1 \overline{P_2} + 1} \qquad \theta = \frac{\theta_1 \theta_2 + \theta_1 \overline{P_1} P_2}{\theta_2 P_1 \overline{P_2} + 1}$$

Due to round-off error, the magnitude of the new θ may not be exactly 1. Accumulated errors in the magnitude of θ can lead to large errors when transforming points near the edge, so we always normalize the new θ to a magnitude of 1.

As an aside, on graphics hardware that has fast support for 3×3 matrix multiplication, it might be faster to use the Klein model, as done in [4], because rigid transformations can then be expressed in terms of linear operations on homogeneous coordinates. Display then requires computing the Poincaré mapping of points represented in the Klein model, which is just a matter of recomputing the distance from the origin according to $r_p = r_k/(1 + \sqrt{1 - r_k^2})$.