Journal of Visual Languages and Computing (1996) 7, 33–55

The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies

JONH LAMPING AND RAMANA RAO

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, U.S.A. (lamping, rao) @parc.xerox.com

Received 3 April 1995 and accepted 29 September 1995

We present a new focus + context technique based on hyperbolic geometry 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. We lay out the hierarchy in a uniform way on a hyperbolic plane and map this plane onto a display region. The chosen mapping provides a fisheye distortion that supports a smooth blending of focus and context. We have deveoped effective procedures for manipulating the focus using pointer clicks as well as interactive dragging and for smoothly animating transitions across such manipulation. Enhancements to the core mechanisms provide support for multiple foci, control of the tradeoff between node density and node display space, and for visualizing graphs by transforming them into trees. © 1996 Academic Press Limited

1. 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 [17]. An important aspect of this work has been the development of focus + context techniques for various classes of information structures, for example, hierarchical [18], chronological [11], calendar [12] and tabular information [15]. In these techniques, a detailed view of a portion of an information set is blended with a view of the overall structure of the set typically using some kind of 'fisheye' distortion of the entire structure. In addition, manipulation operations for controlling the mapping and navigating around the structures are provided. In this paper, we present a new focus + context 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 properties of the figures are salient: first, components diminish in size as they move outwards and, second, there is an exponential growth in the number of components with increasing radius. 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.

The hyperbolic browser initially displays a tree with its root at the center, but the

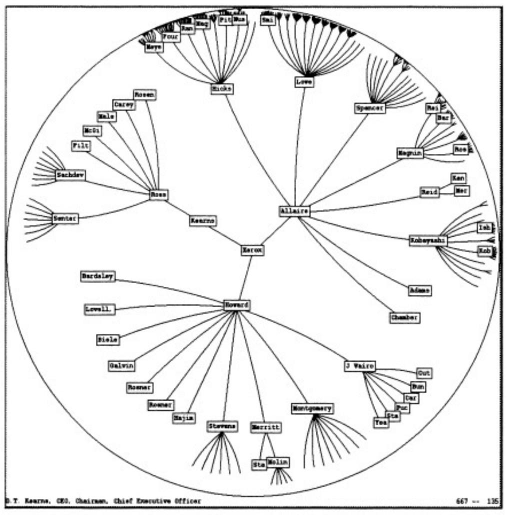

Figure 1. A partial organization chart of Xerox (*ca.* 1988)

display can be transformed smoothly 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 node in focus. 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 windows, a standard 2D hierarchy browser

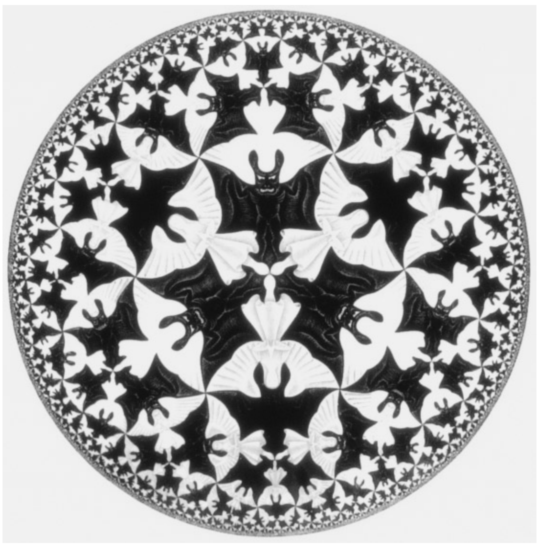

Figure 2. Original inspiration for the hyperbolic browser. Circle Limit IV (Heaven and Hell), 1960, ©1994 M.C. Esher/Cordon Art—Baarn—Holland. All rights reserved. Printed with permission

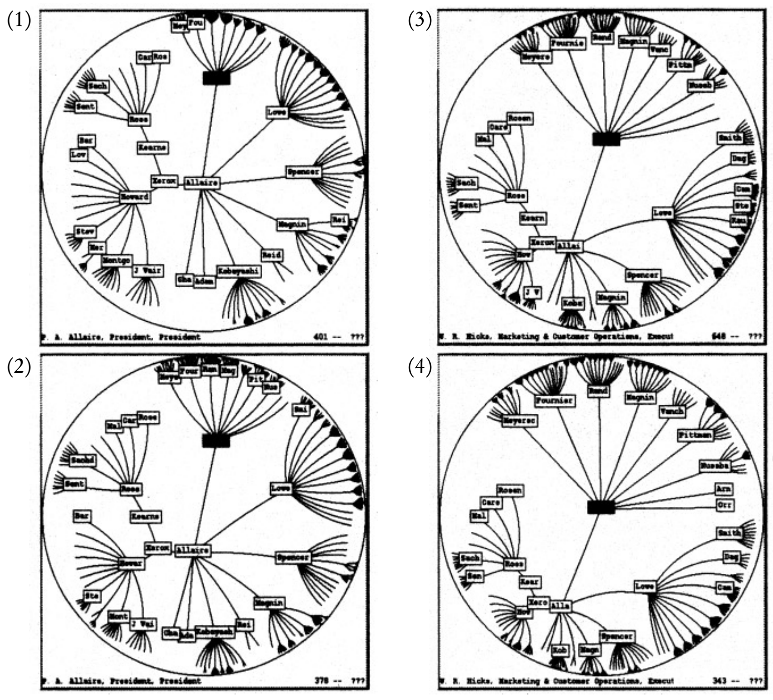

Figure 3. Clicking on the blackened node brings it into focus at the center

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 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 distortion of the elements of the tree display according to their distance from the focus, while easy navigation is obtained by interactive mechanisms for controlling the target area of focus.

Our approach is based on hyperbolic geometry [3, 13], though fortunately it does not require users to understand hyperbolic geometry. The essence of the approach is to lay out the hierarchy on the hyperbolic plane and map this plane onto a display region. On the hyperbolic plane (a construct of a non-Euclidean geometry) parallel lines diverge away from each other. This leads to the convenient property that the circumference of a circle 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 uniformly in hyperbolic space, such that the distance between parent and child and between siblings (as measured in the hyperbolic geometry) is approximately the same everywhere in the hierarchy.

While the hyperbolic plane is a mathematical abstraction, it can be mapped in a natural way onto the Euclidean unit disk, which provides a basis for display on conventional screens. The mapping focuses on one point on the hyperbolic plane by using more of the disk for portions of the plane near that point than on other portions of the plane; remote parts of the hyperbolic plane get miniscule amounts of space near the edge of the disk. Moving the focus point over the hyperbolic plane—equivalent to 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. Other transformations of the mapping from the hyperbolic plane to the display can yield other effects, including changing the relative amount of the display dedicated to the focus nodes and providing multiple foci.

Motion in the hyperbolic plane can yield unintuitive results but these problems can be avoided by careful design. We have developed effective procedures for manipulating the focus using pointing and dragging and for smoothly animating transitions across such manipulation.

The performance requirements of the hyperbolic browser are relatively modest and can be achieved on today's median personal computer. In particular, this is true because our approach supports incremental layout and allows bounding the maximum cost of redisplay by truncating redisplay of nodes below a given resolution limit. Our original Commonlisp prototype runs adequately on low-end Unix workstations by using rendering degradation during animation. A portable C++ implementation (which supports Unix/X, Windows 3.1, and Windows NT) achieves frame rates of under 50 milliseconds for 1000 node trees on an Iris Indigo and a Pentium PC.

2. Problem and Related Work

Many hierarchies, such as organization charts of 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 cannot 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 [10, 21]. Many of these focus + context techniques, including the document lens [19], the perspective wall [11], and the work of Sarkar *et al.* [20, 22], could be applied to browsing trees laid out using conventional 2D layout techniques. The problem is that there is no satisfactory conventional 2D 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 [18] 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 [7] 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 [4] 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 [2] 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 [8] 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.

Hopkins' Pseudo Scientific Visualizer [6] also exploits radial layout and diminishing scale. A tree is laid out by laying out the children of a node at an equal distance in a circle around the node and recursing to each child with a smaller distance for its children. All graphical elemets (e.g. fonts, glyphs) are scaled during the recursion. This technique can be extended to support interactive navigation by descending into any subtree and scaling it up to fill the window. However, further design work is needed to provide ancestral context during descent into the tree.

There have been a number of projects to visualize hyperbolic geometry, including an animated video of moving through hyperbolic space [5]. The emphasis of the hyperbolic browser is a particular exploitation of hyperbolic space for information visualization. We do not expect the user to know or care about hyperbolic geometry.

3. Hyperbolic Browser Basics

The essential operations of the hyperbolic browser can be understood without detailed understanding of hyperbolic geometry. The mathematical details of the implementation are deferred to a later section. The hyperbolic browser *lays out* a tree on the hyperbolic plane and then *maps* the structure to the Euclidean plane during the display operation. *Change of focus* is handled by changing the mapping from the hyperbolic plane to the Euclidean plane. Thus, node positions in the hyperbolic plane need not be altered during focus manipulation. Yet, the mapping is inexpensive. Further, it need be applied only to nodes currently visible at screen resolution. Thus, display cost converges to a constant[a]. Space for displaying *node information* is also computed during layout and mapped through the mapping, again avoiding exhaustive update during change of focus.

3.1. Layout

Laying a tree out in the hyperbolic plane is easier than on a Euclidean plane because the circumference and area of a circle grow exponentially with its radius. There is lots of room. Our recursive algorithm 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 its parent's wedge yet none of the children's wedges will overlap. To compute children's positions in terms of parent's positions, the layout routine navigates through the hyperbolic plane in terms of operations like moving some distance or turning through some angle. These operations are provided by the underlying implementation of the hyperbolic plane.

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

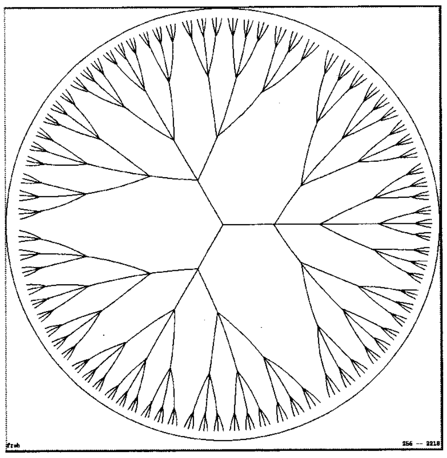
Figure 4. A uniform tree of depth 5 and branching factor 3 (364 nodes)

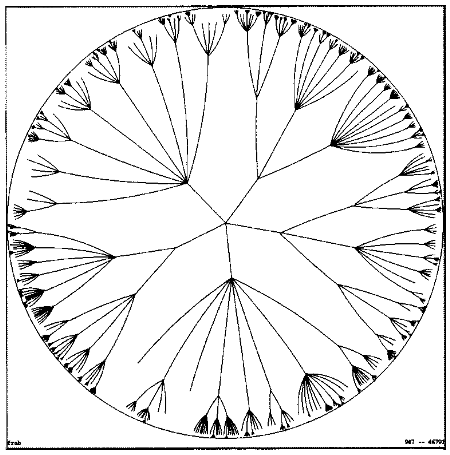
Figure 5. The initial layout of a tree with 1004 nodes using a poisson distribution for number of children. The origin of the tree is the center

[a] Our performance measurements using uniform trees on our portable C++ implementation show a logarithm growth in display time up to several thousands nodes, after which display time approaches a constant.

wedge was available initially. To get a more compact layout for non-uniform trees, we modify this simple algorithm slightly so that siblings that themselves have lots of children get a larger wedge than siblings that do not (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.

The layout routine has the convenient property that the layout of a node depends only on the layout of its parent and on the node structure of two (or maybe three) generations starting from the parent. In particular, there are no global considerations in the layout; the roominess of hyperbolic space renders that unnecessary. As a result, the layout need not be done all at once but can be done incrementally. For example, if a user requests to browse a directory structure, there is no need to traverse the entire structure before displaying anything. Instead, the nodes nearest the root can be laid out and displayed, and then more nodes added as more of the structure is traversed. If the user adjusts the focus, the traversal can give priority to the part of the directory near the focus so that the region in focus is always populated.

An important parameter to the layout routine is the minimum spacing (in the hyperbolic plane) between siblings. A small value for this parameter, as seen on the top of Figure 6, results in nodes being relatively close to each other and with the children of a node subtending a rather small angle. This also puts relatively more nodes in the focus region, but gives each less space. A large value has the opposite effect, as seen in the bottom part of Figure 6. The preferred value depends, in part, on the tradeoff between showing overall tree structure vs. more information about nodes.

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 10.

3.2. Mapping

Once the tree has been laid out on the hyperbolic plane, it must be mapped in some way to the ordinary Euclidean 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 Euclidean plane. Both map the hyperbolic plane to the unit disk and put one vicinity of the hyperbolic plane in focus at the center of the disk while having the rest of the hyperbolic plane fade off in a perspective-like fashion toward the edge of the disk. One mapping, the projective mapping or Klein model, preserves straightness: lines in the hyperbolic plane become chords across the unit disk. The other mapping, called the Poincaré model, is conformal: it preserves angles but maps lines in the hyperbolic space into arcs on the unit disk (as can be seen in the figures).

The Poincaré model worked more effectively for our purposes. Points that are mapped near to 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, thus limiting the context. Furthermore, the Klein mapping severely distorts angles towards the edge of the disk. The Poincaré model, in

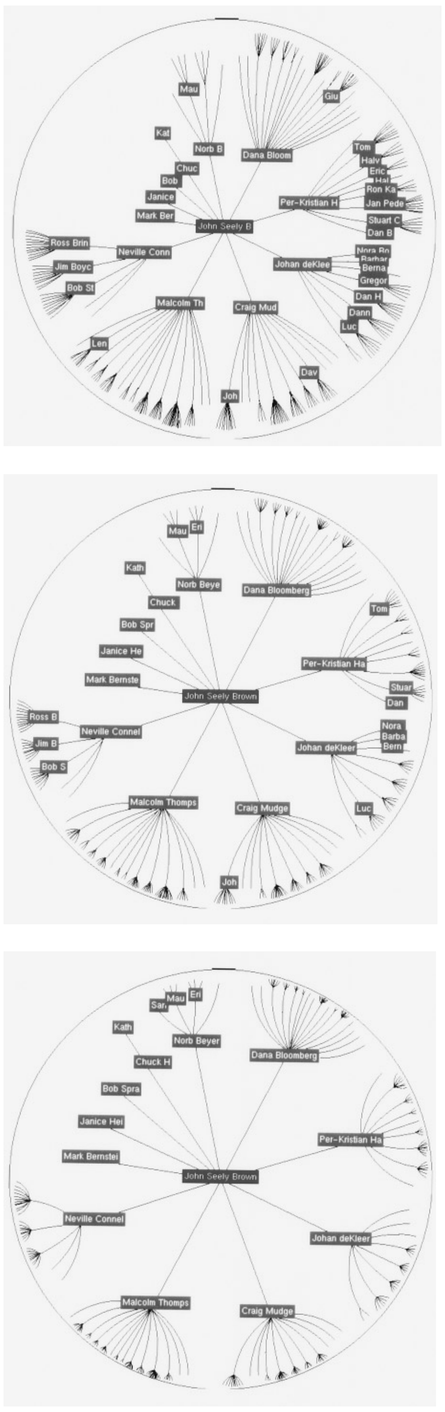

Figure 6. Layout using small, medium and large values for minimum spacing between siblings

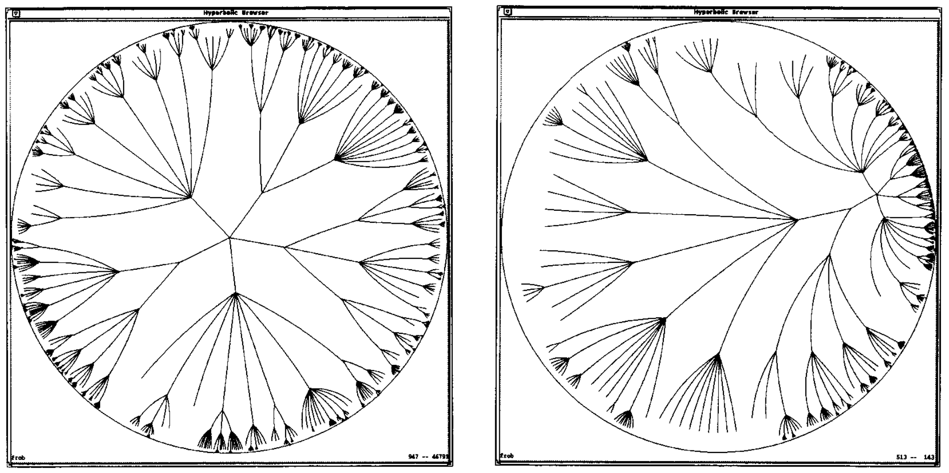

Figure 7. The tree on the left is the same as Figure 5. The focus has moved in the right image to a node that was to the left and slightly below the origin in the left image

contrast, not only does a better job of dividing display space between focus and context, but also preserves angles and local shapes so that structures throughout the display are easier to interpret and compare.

3.3. 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 move toward the edge. Figure 7 shows the same tree as Figure 5 but with a different focus (right hand image). The root has been shifted to the right, putting more focus on the nodes that were toward the left.

Changes of focus are implemented by adjusting the focus of the mapping from the hyperbolic plane to the Euclidean plane. We actually think of this in terms of rigidly moving the hyperbolic plane under the focus, rather than the equivalent motion of the focus over the hyperplane plane. 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. Thus, there is never a need to repeat the layout process. Rather, the original node positions are rigidly transformed and mapped to the Euclidean plane during display.

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. Furthermore, since we only need the mapped positions of the nodes that will be displayed, the transformation is only 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. The implementation of translation can be fairly efficient, requiring about 20 floating point operations to translate a point and map it to the Euclidean plane, comparable to the cost of rendering a node on the screen.

3.4. 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. This can be used to identify screen space for displaying node information. We can compute a circle in the hyperbolic plane around each node that is guaranteed not to intersect with 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 representation of the node. The display regions can be used in conjunction with a facility that selects different representations for each node depending on the amount of space available. This could be used to implement a 'zoom and bloom' space similar to that of the Pad systems [1, 14].

While the circle approach is very efficient, it does not make full use of the screen space because significant parts of the display are not covered by any circle, especially when nodes have many children. A somewhat more expensive technique that does a better job of identifying screen space for displaying node information notes is based on calculating, during layout, the midway points in hyperbolic space between a node, its parent, its nearest siblings and one of its children. These points are then mapped to the display space, and used to identify an elliptical display region for the node (shown in Figure 8). For convenience of the node display routines, the ellipses are

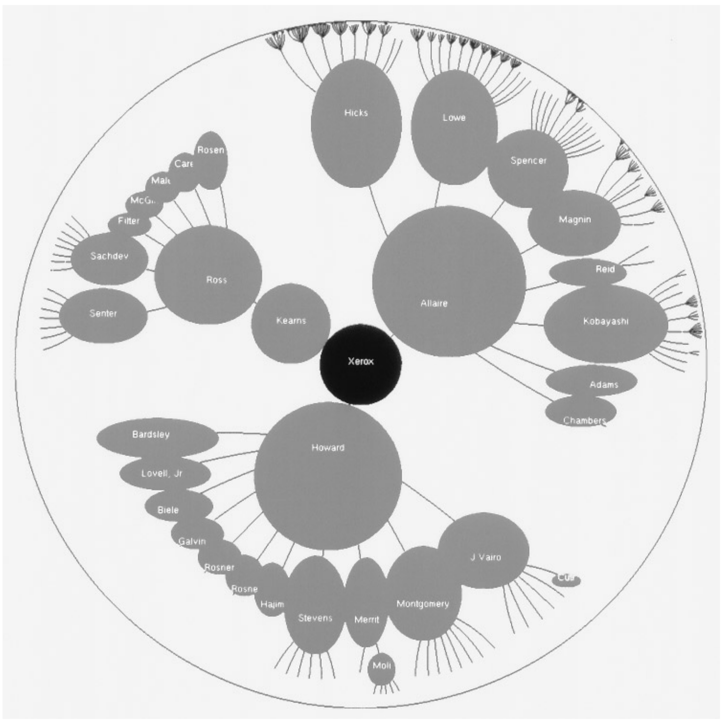

Figure 8. The regions available to nodes for displaying information

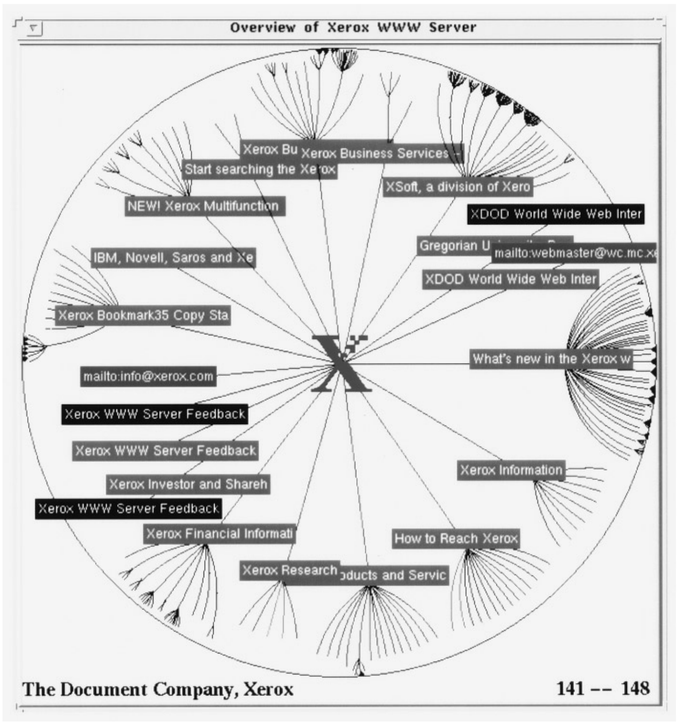
Overview of Xerox WWW Server
Xerox Bu
Xerox Business Services
Start searching the Xerox
XSoft, a division of Xero
NEW! Xerox Multifunction
XDOD World Wide Web Inter
Gregorian U
mailto:webmaster@wc.mc.x
IBM, Novell, Saros and Xe
XDOD World Wide Web Inter
Xerox Bookmark35 Copy Sta
What's new in the Xerox w
mailto:info@xerox.com
Xerox WWW Server Feedback
Xerox WWW Server Feedback
Xerox Information
Xerox Investor and Shareh
Xerox WWW Server Feedback
Xerox Financial Informati
How to Reach Xerox
Xerox Research
oducts and Servic
The Document Company, Xerox
141 -- 148

Figure 9. Long text mode with up to 25 characters displayed for each node that would normally display at least 2 characters

always aligned with the axes and given a slight horizontal bias. This computation is not exact since it can lead to modest overlapping of display regions. However, it will typically be effective in practice because it provides more node inforation without interfering with understanding of the structure.

Some applications can tolerate even greater amounts of overlap. For example, the browser supports a 'long text' mode in which all nodes beyond an allocated space threshold disregard their boundaries and display up to some maximum number of characters. Despite the overlapping of the text, this leads to more text being visible and discernible on the screen at once (see Figure 9).

4. Preserving Orientation

The use of hyperbolic space presented the challenging design problem of preserving a user's sense of orientation. Difficulties arise on the hyperbolic plane because objects tend to get rotated as they are moved. For example, most nodes rotate on the display during a pure translation. There is a line that does not 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 first frame, ends up with its children fanning out to the right in the last 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 source and the destination of the drag, the display will do a counter-intuitive pirouette about the point being dragged.

This effect is caused by a fundamental property of hyperbolic geometry. In the usual Euclidean plane, if some graphical object is dragged around, but not rotated, then it 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 in which the loop was traversed.) This leads to the counter-intuitive behaviour that a user who moves the focus around the hierarchy can experience a different orientation each time they revisit a node, even though all they did was translations.

We address both of these problems by interpreting the user's manipulation 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. The key is to use the additional rotation to establish some property that the user can easily understand. From the user's perspective, drags and clicks move the point that the user is manipulating where they expect, while preserving some other intuitive property.

We have found two promising properties for guiding the added 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 transformation in Figure 3 works this way. It seems to give an intuitive behaviour both for individual drags and for the cumulative effect of drags. In this approach, the user is typically not aware that rotation is being added.

The other approach does not attempt to preserve node orientation. Instead, when a node is brought to the focus, the display is rotated to have its children fan out in a canonical direction e.g. to the right. This is illustrated in Figure 10 and also in the animation sequence in Figure 11. This approach works best when the children of the root node are all laid out on one side, as also true in the two figures, so that the children of the root also fan out in the canonical direction when it is in focus.

5. Animated Transitions

Animated transitions between different views of a structure maintain object constancy and help the user assimilate 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 an 'nth-root' of a transition transformation, i.e. the rigid transformation that applied n times will have the same effect as the original. Succesive applications of the 'nth-root' generate the intermediate frames. The sequences in Figure 3 and Figure 11 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 display quality during motion. These compromises provide options for use in a system that automatically adjusts rendering quality

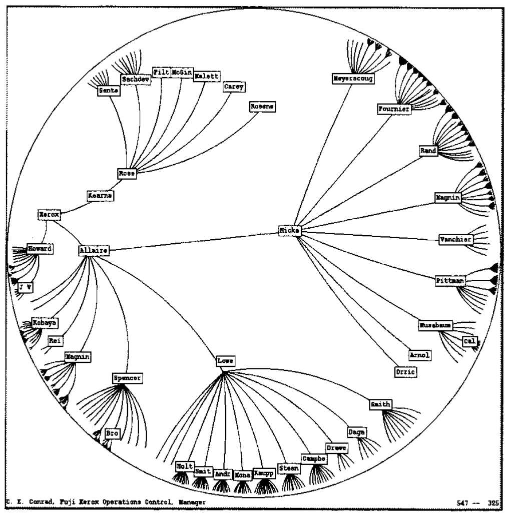

Figure 10. In right orientation mode, the children of the root are laid out only to its right, and the structure is rotated to display children of the focus node to its right

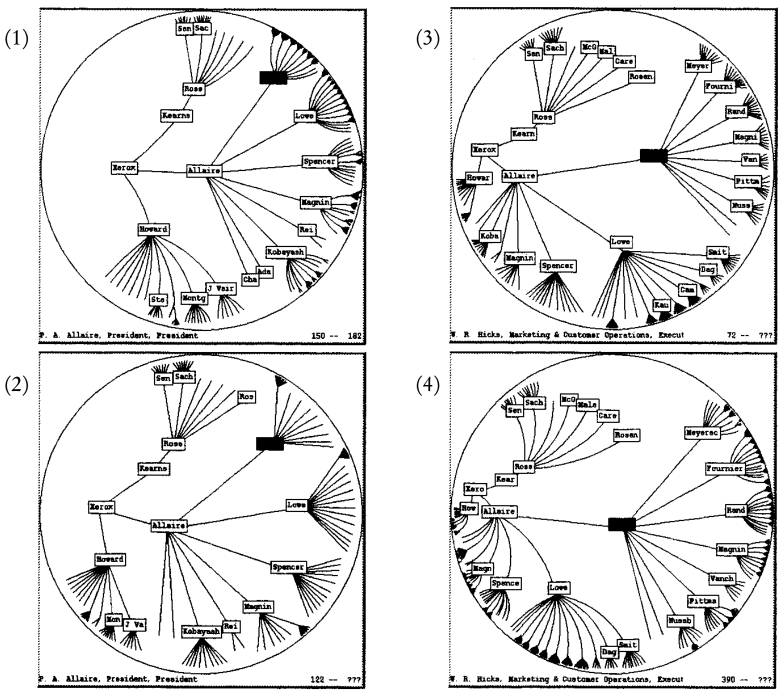
(1)
(2)
(3)
(4)
P. A. Allaire, President, President
150 -- 182
P. A. Allaire, President, President
122 -- ???
W. R. Hicks, Marketing & Customer Operations, Execut
72 -- ???
W. R. Hicks, Marketing & Customer Operations, Execut
390 -- ???

Figure 11. Animated transition with compromised rendering

during animation, e.g. the Information Visualizer governor [16] or Pacers [23]. Fortunately, there are compromises that do not significantly affect the sense of coherence. Figure 11 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 aggressively increases display performance since the vast majority of nodes are very close to the edge. But it does not significantly degrade perceptual quality for a moving display because those nodes occupy only a small fraction of the display and not the part that the user is typically focusing on.

Another 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 are not as important during animation. This appears to be true for two reasons. The difference between arcs and lines is not as apparent during motion. Furthermore, again particularly during motion, the user's attention 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.

6. Implementation

In this section, we present the details of the mathematics necessary to implement the hyperbolic browser. It is primarily of interest only to those interested in implementing our technique.

Representations Our implementation relies on representation for the hyperbolic plane, rigid transformations of the plane and mappings from the plane to the unit disk. We represent a point in hyperbolic space by the corresponding point in the unit disk under the Poincaré mapping. Our representation thus directly encodes the mapping from the plane to the unit disk[b]. Points in the unit disk are represented as floating point complex numbers of magnitude less than 1.

Rigid transformations of the hyperbolic plane are represented by circle preserving transformations of the unit disk. Any such transformation can be expressed as a complex function of z of the form

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

[b] On graphics hardware that has fast support for 3×3 matrix multiplication, it might be faster to use the Klein model for the representation of the hyperbolic plane, as done in [5], because rigid transformation can then be expressed in terms of linear operations on homogeneous coordinates. Mapping to the 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 $r_p = r_k/(1 + \sqrt{1 - r_k^2})$.

Where P and θ are complex numbers, $|P| < 1$ and $|\theta| = 1$, and $\bar{P}$ 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).

Given a transformation, $\langle P, \theta \rangle$ the inverse transformation (which is needed to map from display coordinates back into the hyperbolic plane) can be computed by:

$$P' = -\bar{\theta} P \qquad \theta' = \bar{\theta}$$

The composition of a transformation $\langle P_1, \theta_1 \rangle$ followed by $\langle P_2, \theta_2 \rangle$, is given by:

$$P = \frac{\theta_2 P_1 + P_2}{\theta_2 P_1 \bar{P}_2 + 1} \qquad \theta = \frac{\theta_1 \theta_2 + \theta_1 \bar{P}_1 P_2}{\theta_2 P_1 \bar{P}_2 + 1}$$

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.

Layout The layout routine is structured as a recursion that takes a node and a wedge in which to lay out the node and its children. It places the node at the vertex of the wedge, computes a wedge for each child and recursively calls itself on each child. The wedge is represented by the point at its vertex, the endpoint of its midline, and the angle from the midline to either edge of the wedge. The vertex and endpoint are represented by complex numbers. Since the endpoint is a point at infinity, its complex number has magnitude 1. For convenience of calculations, the endpoint is represented relative to the vertex, in the sense that the representation corresponds to where the endpoint would end up if the vertex were shifted to the origin.

The layout routine records the position of the wedge as the position of the node. Then, if there are children, a simple procedure is to divide the angle of the wedge by the number of children, n, and subdivide the wedge into n equal sized wedges, each spanning that angle. The slightly more complicated procedure actually used in the figures gives different children different fractions of the wedge depending logarithmically on the number of children and grandchildren of each child.

The children are placed in the middle of their subwedges at a distance computed by the formula

$$d = \sqrt{\left(\frac{(1 - s^2)\sin(a)}{2s}\right)^2 + 1} - \frac{(1 - s^2)\sin(a)}{2s}$$

where a is the angle between midline and edge of the subwedge and s is the desired distance between a child and the edge of its subwedge; we typically use a value of about 0·12 for s (the values used in Figure 6 are 0·06, 0·12 and 0·18, respectively). The result, d, is the necessary distance from parent to child. If the calculation of d results in a value less the s, we set d to s to maintain a minimum spacing between parent and child. Both s and d are represented as the hyperbolic tangent of the distance in the hyperbolic plane. This form facilitates later operations in the Poincaré map, because it has the convenient property that a line segment on the unit disk with one end on the origin and extending the given amount represents a segment extending the represented distance in the hyperbolic plane.

Given a subwedge for a child and the distance, d, to the child, the next step is to calculate a wedge inside the subwedge, with its vertex at the child, to use for the recursive call. Given the vertex, p, midline endpoint, m, and angle, a, of the subwedge, the corresponding parameters of the contained wedge that results from moving d into the subwedge can be calculated using the transformation apparatus:

$$
\begin{aligned}
p' &= \mathrm{Trans}\,(dm, \langle p, 1\rangle)\\
m' &= \mathrm{Trans}\,(\mathrm{Trans}\,(m, \langle p, 1\rangle), \langle -p', 1\rangle)\\
a' &= \mathrm{im}\,(\log\,(\mathrm{Trans}\,(e^{ia}, \langle -d, 1\rangle)))
\end{aligned}
$$

where Trans is the transformation function described above, which takes a point and a transformation specification and returns the transformed point. The $\mathrm{im}\,(\log\,(\cdots))$ in the formula for a' returns the angle corresponding to the complex number, doing the inverse of the conversion from angle to complex number done by the e^{ia} (these functions can be implemented using cos, sin and arc tangent and a complex number constructors and selector instead).

Display Node display involves recursing on the node structure. Starting from the root, the procedure draws a node's incoming link, recurses on its children, and finally draws the node. This procedure utilizes a 'current transformation' which maps from the hyperbolic plane to the unit disk positions according to the current focus. The coordinates for drawing links and nodes are obtained by transforming the recorded positions and spacings of nodes by the current transformation and scaling the resulting unit disk coordinates to the actual display window size.

Links between nodes are drawn as arcs (corresponding to straight lines in the hyperbolic plane) to convey a sense of warping of the space and to preserve the angle near the node centers. The center of curvature of the arc that links two points represented by complex numbers a and b in the unit circle is given by

$$
\begin{aligned}
d &= \mathrm{re}\,(a)\,\mathrm{im}\,(b) - \mathrm{re}\,(b)\,\mathrm{im}\,(a)\\
c &= \frac{i}{2}\,\frac{(a(1+|b|^2) - b(1+|a|^2))}{d}
\end{aligned}
$$

In addition, if the quantity d is positive, then the arc from a to b goes clockwise around the circle. Otherwise it goes counterclockwise.

Interaction After layout, the recorded positions of the nodes are not changed; instead a current transformation is maintained for use during display. Initially, the current transformation is set to the identity transformation, $\langle 0, 1\rangle$. When the user clicks on a new position to be the focus, we compute a transformation that maps the indicated point on the hyperbolic plane to the origin. Similarly, when the user drags from one point to another, we compute a transformation that maps from the first indicated point to the second. In both cases, in orientation preserving mode, the orientation of the origin is preserved. The desired origin-preserving transformation can be calculated with:

$$
\begin{aligned}
a &= \mathrm{Trans}\,(s, \langle -p, 1\rangle)\\
b &= \frac{\mathrm{re}\,((e-a)(1+\overline{ae})) + \mathrm{im}\,((e-a)(1-\overline{ae}))i}{1-|ae|^2}\\
T &= \mathrm{Compose}\,(\langle -p, 1\rangle, \langle h, 1\rangle)
\end{aligned}
$$

where s is the starting point where the user first clicked, e is the endpoint point (either the origin or where the user ended their drag), and p is the point whose orientation is to be preserved (we use the image of the origin under the current transformation). Compose is transformation composition, defined earlier. T is the resulting transformation. This transformation is composed with the current transformation to get the new current transformation.

To provide a smooth animation between the current transformation and a new current transformation, a series of frames with transformations between the two are generated. Linear interpolation in disk coordinates can be used to compute the intermediate transformations, but we use a more sophisticated procedure involving calculation of the '*n*th-root' of the transition transformation between the current and new transformations. The computation of the nth root of a transformation, $\langle P, \theta\rangle$ is somewhat involved, with three cases. Different formulas apply in the different cases but, in addition, some formulas are converted to equivalent forms to improve numerical stability. The procedure first computes:

$$d = 4\,|P|^2 - |\theta - 1|^2$$

Then depending on the value of d, one of following three intermediate computations is performed:

if $d > 0$

$$t = \frac{\sqrt{d}}{|\theta + 1|}$$

$$r = \frac{\tanh\left(\dfrac{\operatorname{argtanh}(t)}{n}\right)}{t}$$

$$a = r^2 \frac{|\theta - 1|^2}{|\theta + 1|^2}$$

$$b = r\sqrt{\frac{1 + \dfrac{|\theta - 1|^2}{|\theta + 1|^2}}{1 + a}}$$

if $d = 0$

$$a = \frac{1}{n^2}\frac{|\theta - 1|^2}{|\theta + 1|^2}$$

$$b = \frac{1}{n}\sqrt{\frac{1 + \dfrac{|\theta - 1|^2}{|\theta + 1|^2}}{1 + a}}$$

if $d < 0$

$$t = \frac{\sqrt{-d}}{|\theta + 1|}$$

$$a = \frac{\tan^2\left(\dfrac{\arctan(t)}{n}\right)|\theta - 1|^2}{-d}$$

$$b = \frac{1}{|\theta - 1|}\sqrt{\frac{4a}{(1 + a)}}$$

Finally, a transformation is calculated:

$$m = \text{if im}\,(\theta) > 0 \text{ then } 1 \text{ else } -1$$

$$\theta' = \frac{1-a}{1+a} + \frac{2m\sqrt{a}}{1+a}i$$

$$P' = Pb\sqrt{\theta'/\theta}$$

7. Mappings for Dual Focus and Stretch Factor

While the Poincaré mapping is our basis for getting from the hyperbolic plane to the Euclidean plane, variations are sometimes useful. If we start with the Poincaré mapping to get to the Euclidean plane and then apply any conformal (angle preserving) mapping to the Euclidean plane, the composition will still have the advantages of being a conformal mapping from the hyperbolic plane to the Euclidean plane. But, it can have some properties that the Poincaré map lacks.

One useful mapping is a dual focus mapping, as shown in Figure 12, where there are two foci, one for each of two different points on the hyperbolic plane. We achieve this mapping by applying the transformation

$$z' = z\,\frac{a^2 - 1}{a^2 z^2 - 1}$$

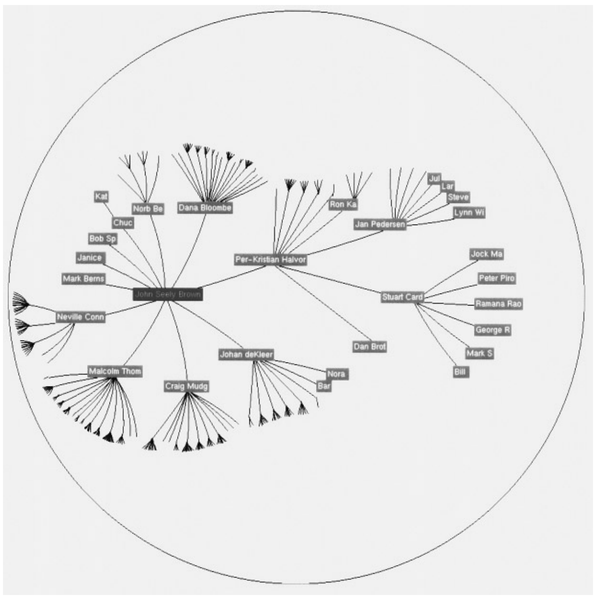

Figure 12. A dual focus mapping allows examining contexts of distant parts of the hierarchy

to the unit disk, represented as complex numbers. The result is that the points that would have mapped to a and $-a$ now become the two foci. If a is small, the display is elongated slightly, while if it is close to 1, the display looks more like a binocular view.

One use of the dual focus mapping is to put a node and one of its ancestors at the two foci to better visualize their relationship, as done in Figure 12. This is done by first finding a circular mapping that puts those nodes an equal distance from the center, on either side, and then applying the dual focus mapping. A more prosaic use of the dual focus is to change the aspect ratio of the visualization (by up to about 50%) to better fit a window, while keeping a conformal mapping.

Another variant mapping is deliberately non-conformal. As mentioned above, there is a layout parameter for how close together to place sibling nodes (as shown in Figure 6). Often, a user might want to alter this parameter interactively so as to put more or fewer nodes in the focus region. Fortunately, a non-conformal circumference-reducing transformation can achieve almost the same effect without having to redo the layout. Figure 13 shows a side-by-side comparison of shrinking the sibling spacing in the layout by 50% and doing a 50% circumference reduction.

The idea is to move nodes radially further from or closer to the root node in the hyperbolic plane. The transformation we use adjusts the radial distance so that the circumference at that new distance is some ratio of the circumference at the original distance. If positions in the hyperbolic plane are represented by their Poincaré map with the root centered, then the new radial distance, d', is related to the old distance, d, by

$$s\frac{d}{d^2-1}=\frac{d'}{d'^2-1}$$

where s is the scaling factor. This has approximately the same effect as a new layout with the spacing between siblings being adjusted by the same ratio.

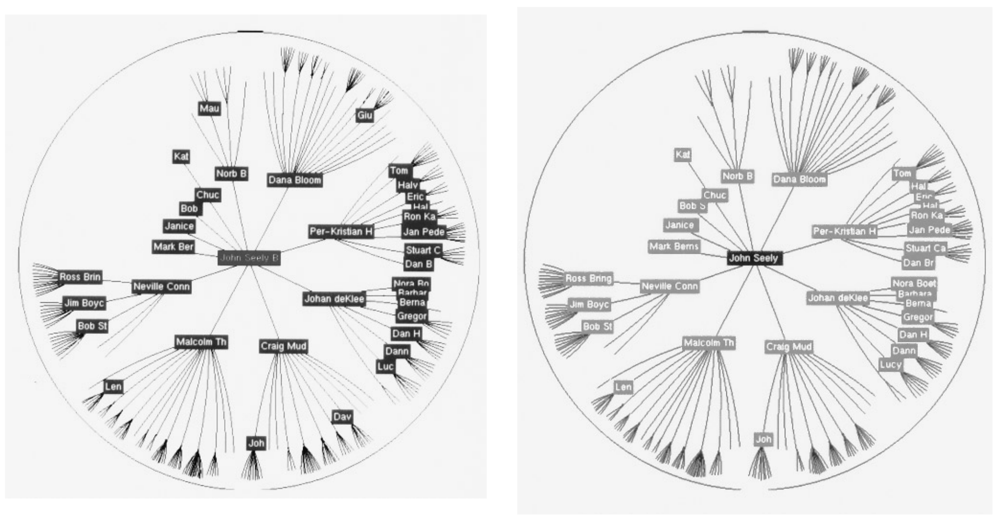

Figure 13. Using layout-time parameter for sibling spacing (left) vs. a display-time non-conformal circumference reduction mapping (right)

8. Visualizing Graphs

A natural question is whether the hyperbolic browser can be extended to visualize structures other than trees, for example, more general graph structures. The properties of the hyperbolic plane exploited for hierarchy layout apply similarly to laying out general graph structures: its spaciousness overcomes the metric problem of laying out structures in Euclidean space (i.e. the space is too confined). However, laying out general graph structures also presents a topological problem: dealing with crossing links. Since the hyperbolic plane has the same topological structure as the Euclidean plane, it does not overcome this problem.

The spaciousness of the hyperbolic plane does, however, allow a finesse that would be more difficult using conventional layout techniques. This approach involves converting a graph to a tree and then using the hyperbolic browser to visualize the resulting tree. This approach can be quite effective when the graphs are 'almost' trees. For example, a directory structure may be a tree, except for some symbolic links that introduce non tree-like links. Similarly, the links in a World Wide Web structure are often mostly tree-like, with additional cross-references.

We convert a graph as a tree by making a copy of each graph node for each incoming edge so that each edge goes to its own copy. Then one copy of a node is chosen as the main one, for example, the one (or the one of several) closest to the root. The children of a node are attached only to the main copy. This transformation is straightforwardly implemented using a breadth-first traversal of the graph. When

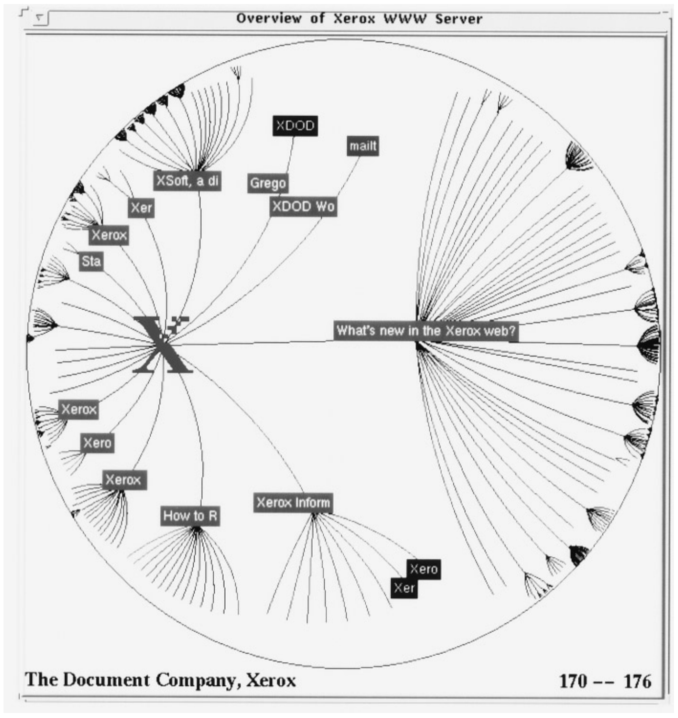

Figure 14. The link structure of the Xerox WWW Server shown as a tree. The X represents the Xerox home page and its children represent the pages pointed to by that page, and so on

the tree is displayed, the nodes that are copies can be visually distinguished. For example, the link structure of documents served by the Xerox WWW server starting from its root document is shown in this manner in Figure 14. The repeated nodes are shown with blue backgrounds.

This presents the problem of finding the children of repeated nodes. One solution is to maintain the original layout and provide a mechanism for automatically navigating to the main copy. So, for example, when one of the repeated nodes is selected by the user, an animated transition can move the main copy of the node into focus so that its descendants are visible. Understanding of the transition can be facilitated by using a path through the nearest common ancestor at a pace controlled by the length of the path.

An alternative approach for turning a graph into a tree gives all node repeats a copy of the descendent hierarchy; thus, all node repeats are equally valid. This expands the graph out into a (possibly infinite) tree, with one tree node for each rooted path in the graph. Since layout can be done locally on the hyperbolic plane and thus incrementally, the entire logical structure need not be laid out initially. Rather only the part of the structure visible at pixel resolution needs to be laid out, with additional layout done as the user moves the focus. For efficiency, only one set of descendant hierarchy needs to be maintained since transformations can be calculated which map each unique descendant hierarchy (i.e. one per graph node) to each of its locations. This is possible because of the roominess of the hyperbolic plane and the associated viability of using uniform layout at all locations.

9. Conclusion

We believe that the hyperbolic browser offers a promising new addition to the suite of available focus + context techniques. Hyperbolic geometry provides an elegant solution to the problem of visualizing large hierarchies. The hyperbolic plane has the room to layout large hierarchies, and the Poincaré map provides a natural, continuously graded, fisheye 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.

A preliminary experimental study, described in [9], though inconclusive, did ascertain that all of its four subjects preferred the hyperbolic browser over a conventional browser in both 'getting a sense of the overall tree structure' and 'finding specific nodes by their titles', as well as 'overall'. 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.

We have developed and explored a number of enhancements and variations of the core hyperbolic browser that address common needs. The alternative mappings for dual focus and interactive focus stretching, respectively, provide mechanisms for examining distal parts of hierarchy simultaneously and for controlling the tradeoff between the number of nodes visible and the amount of space available for displaying node information. Our initial work on graph visualization suggests that the hyperbolic browser can be applied to graphs of the kind common in many applications.

Many of the conventional techniques of information visualization would increase the value of the hyperbolic browser for navigating and learning hierarchies. For example, landmarks can be created in the space by utilizing color and other graphical elements (e.g. the prominent red X Xerox logo is an effective root marker). Other possibilities include providing a visual indication of where there are nodes that are invisible because of the resolution limit, using line thickness to convey depth or other information, and using a ladder of multiscale graphical representations in node display regions as done in 'zoom and bloom' interfaces. The effective use of these types of variations are likely to be application or task dependent and thus best explored in such a design context.

Acknowledgements

The focused work described here would not have ever materialized except for the rich context of information visualization work at PARC as initiated by Stuart Card, Jock Mackinlay and George Robertson. A number of other colleagues have made valuable suggestions during the development of our prototypes. We would particularly like to thank Peter Pirolli for his collaboration on the experimental study described in a previous paper. We are also grateful for Barbara Gable's assistance in developing datasets and in obtaining copyright clearances for the Escher figure. The reviewers of this paper and its ancestors have steadily improved its quality. Xerox Corporation is seeking patent protection for technology described in this paper.

References

1. B. B. Bederson & J. D. Hollan (1994) Pad++: A zooming graphical interface for exploring alternate interface physics. In: *Proceedings UIST'94* ACM Press, New York, pp. 17–26.
2. J. Bertini (1983) *Semiology of Graphics* University of Wisconsin Press, Wisconsin.
3. H. S. M. Coxeter (1965) *Non-Euclidean Geometry* University of Toronto Press, Toronto.
4. G. W. Furnas (1986) Generalized fisheye views. In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* Addison-Wesley, Reading, MA, pp. 16–23.
5. C. Gunn (1991) Visualizing hyperbolic space. In: *Computer Graphics and Mathematics* Springer-Verlag, Berlin, pp. 299–31.
6. Don Hopkins (1989) The shape of psiber space. http://hello.kaleida.com/u/hopkins/psiber/psiber:html, 1989.
7. B. Johnson & B. Shnedierman (1991) Tree-maps: A space-filling approach to the visualization of hierarchical information. In: *Proceedings of Visualization 1991.* IEEE Society Press, Los Alamitos, CA, pp. 284–291.
9. J. Lamping, R. Rao & P. Pirolli (1995) A focus + context technique based on hyperbolic geometry for visualizing large hierarchies. In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* Addison-Wesley, Reading, MA.
10. Y. K. Leung & M. D. Apperley (1994) A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction* **1**, 126–160.
11. J. D. Mackinlay, G. G. Robertson & S. K. Card (1991) The perspective wall: Detail and context smoothly integrated. In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* Addison-Wesley, Reading, MA. pp. 173–179.
12. J. Mackinlay, G. Robertson & R. Deline (1994). Developing calendar visualizers for the information visualizer. In: *Proceedings of the ACM Symposium on User Interface Software and Technology* ACM Press, New York, pp. 109–118.
13. E. E. Moise (1974) *Elementary Geometry from an Advanced Standpoint* Addison-Wesley, New York.

14. K. Perlin & D. Fox (1993) Pad: An alternative approach to the computer interface. In: *Proceedings SIGGRAPH'93* Addison-Wesley, Reading MA, pp. 57–64.
15. R. Rao & S. K. Card (1994) The table lens: Merging graphical and symbolic representation in an interactive focus + context visualization for tabular information. In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* Addison-Wesley, Reading, MA, pp. 318–322 and 481–482.
16. G. G. Robertson, S. K. Card & J. D. Mackinlay (1989) The cognitive coprocessor architecture for interactive user interfaces. In: *Proceedings of the ACM SIGGRAHP Symposium on User Interface Software and Technology* ACM Press, New York, pp. 10–18.
17. G. G. Robertson, S. K. Card & J. D. Mackinlay (1993) Information visualization using 3d interactive animation. *Communications of the ACM* **36**, 56–71.
18. G. G. Robertson, J. D. Mackinlay & S. K. Card (1991) Cone trees: Animated 3d visualizations of hierarchical information. In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* Addison-Wesley, Reading, MA, pp. 189–194.
19. G. G. Robertson & J. D. Mackinlay (1993) The document lens. In: *Proceedings of the ACM Symposium on User Interface Software and Technology* ACM Press, New York, pp. 101–108.
20. M. Sarkar & M. H. Brown. Graphical fisheye views of graphs. In: *Proceedings of the ACM SIGCHI Conferences on Human Factors in Computing Systems* Addison-Wesley, Reading, MA, pp. 83–91.
21. M. Sarkar & M. H. Brown (1994) Graphical fisheye views. *Communications of the ACM* **37**, 73–84.
22. M. Sarkar, S. Snibbe & S. Reiss (1993) 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, New York, pp. 81–92.
23. S. H. Tang & M. A. Linton (1993) Pacers: Time-elastic objects. In: *Proceedings of the ACM Symposium on User Interface Software and Technology* ACM Press, New York, pp. 35–44.