11-16-2004

# 1-Hyperbolic projection for user interfaces

Alexander Kolliopoulos
*Trinity University*

**The 1-Hyperbolic Projection for User Interfaces**

Alexander Kolliopoulos

A departmental senior thesis submitted to the

Department of Computer Science at Trinity University

in partial fulfillment of the requirements for Graduation.

April 21, 2003

Thesis Advisor

Department Chair

Associate Vice President

for

Academic Affairs

# The 1-Hyperbolic Projection for User Interfaces

Alexander Kolliopoulos

## Abstract

The problem of dealing with representations of information that does not fit conveniently within allotted screen space is pervasive in graphical interfaces. While there are techniques for dealing with this problem in various ways, some properties of such existing techniques are not satisfying. For example, global structure of information may be lost in favor of local focus, or information may not be mapped into a rectangular area. The 1-hyperbolic interface is proposed to deal with some of these deficiencies, and the mathematics involved in display and interaction are derived. The calculations necessary for this interface are easy to implement, and can run reasonably even on slow devices. A fully functional prototype for displaying tree structures has been developed to compare the effects of this new interface to those of a standard interface. The results of usability experiments conducted with this prototype are also presented and analyzed.

## Acknowledgments

I wish to express sincere gratitude to Dr. John Howland for his constant support and encouragement. This work could not have happened without him.

I would also like to thank Dr. Glenn Meyer, whose experience and guidance in human interface experiments and data analysis was immensely helpful.

Dr. Berna Massingill's help with proper LaTeX formatting is greatly appreciated. Without her time, this thesis would have had to happen with a much less capable typesetting program.

Furthermore, thanks goes to Dr. Mark Lewis for putting in an afternoon to develop an interesting tool for distorting interfaces in Java.

For his assistance in statistics, I owe Dr. Richard Cooper my gratitude.

Finally, I would like to thank the entire Trinity University Computer Science and Mathematics departments. The faculties of both of these departments have been instrumental in helping me to develop this work. I cannot possibly thank everyone who has contributed here, and for that I apologize.

# The 1-Hyperbolic Projection for User Interfaces

Alexander Kolliopoulos

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 The Focus+Context Problem

Nearly anyone who has had to work with a computer system has been forced to deal with the problem of displaying a large amount of information with a limited presentation space. Modern computing has introduced great power in a relatively short period of time, but the ways in which people interact with data has changed little since the introduction of the graphical interface. Incremental and evolutionary changes certainly have made the experience easier over time. Translucent menus in Apple's Mac OS X and KDE may help minimize the problem of clutter by allowing one to see the information behind them. Mouse-based gestures can aid in performing repetitive tasks, with implementations in the Mozilla and Opera web browsers [12, 13]. Pie menus have been designed to minimize pointer motion and maximize available screen space by only appearing over the cursor location and positioning options around the pointer in a circular fashion when needed [6]. Despite small innovations such as these that are often limited in the available applications implementing them, the graphical interface is largely unchanged from the early days of the Xerox Star.

However, this has not slowed the growth of information that more powerful computers are capable of handling. Nor has it stopped displays from shrinking for applications such as portable devices.

Information can take on many and varied forms, including hierarchical tree structures, linear data, two-dimensional data, complex graph constructions, and so on. Hierarchies can take the form of a file system tree; menus, such as a hierarchical bookmark list in a web browser; class hierarchy visualizations; and just about any other type of information that can be organized into categories, like contact lists or Usenet newsgroups. Tree structures are so common in computing that a number of techniques to visualize them have been proposed, including tree maps [16], cone trees [15], information slices [1], and the hyperbolic geometry of the Poincaré disk [10]. Linear data can take the form of long documents, a list of objects such as file names or images, or a signal. Two-dimensional data can become quite large in applications such as photo processing; and complex graph structures are less common in interfaces, but they can be useful in visualizing networks. Although these types of information can vary dramatically in their presentation, the interfaces for dealing with more information than screen space allows are fairly standard.

Typically, when one deals with large sets of data, the introduction of a scroll pane with one or two scrollbars tends to be the norm. This has a number of desirable features. For instance, there is no distortion in the information space. The scroll pane simply limits the user's view to some constrained area, while offering scrollbars to allow the user to freely navigate the space. This interface paradigm is advantageous for a number of reasons. For one, the scroll pane interface offers a simple mapping into the information space. The position of each knob on its scrollbar shows exactly where in the information space the current view is located, while the size of the knobs in relation to the scrollbars indicates how much information is not being displayed at the moment. This interface is ubiquitous.

Nearly all software that involves working with information that is either large or dynamic implements scrollbars in some way. There are notable exceptions, such as some 3D rendering packages, but in these cases, such an interface is often not appropriate to the application. However, due to the fact that scroll pane interfaces may be found everywhere, they are familiar to even novice computer users. This may be the greatest asset when considering the application of scroll panes. For being so common, there is little danger of confusing or alienating users by relying on them.

For all its advantages, the scroll pane strategy is not without problems. One objection to the use of scrollbars is that they compound the very problem which they are introduced to deal with, that of limited screen real estate. Every pixel that a scrollbar occupies cannot be filled with the data that they are designed to allow one to navigate. While this may be dealt with by making the scrollbars excessively thin, new problems arise in usability. The less area a scrollbar has, the more difficult it is to locate with a pointing device. This problem is especially evident on handheld computers, where screen resolutions of 320x320 are not uncommon. Thus the scrollbar designer is faced with the problem of finding a balance between information content and usability. Beyond this, use of the scroll pane can lead to excessive navigation for the user, especially when information spaces become very large. An example of this problem is common in the standard file system browser common to most graphical operating systems. When a node with many children is expanded, this forces a user to scroll up and down to find a parent node or some specific child. Such navigation may waste time and force the user to expend energy trying to keep track of the global structure in his or her mind.

The scroll pane is not the only common technique introduced to deal with large amounts of information. Simultaneous views, sometimes called overview-detail displays, allow one to display both local detail and global structure of information at the same time. In this model, rather than relying on a single view of the data in question, two displays are utilized, where one display shows some interesting focus, and the other gives a broad overview. This interface can be found in popular online roadmap guides, such as Yahoo Maps and Mapquest. While this is well suited to continuous, two-dimensional information such as maps, an analogue to more linear data such as documents may also apply. An example can be found in Adobe's Acrobat Reader product, in which two scroll panes may be utilized. In the main scroll pane, a readable view of a document is presented, while another scroll pane gives an overview of pages before or after the current document with smaller thumbnails. While this helps a user orient his or herself locally, it does not succeed in giving a view of the full context of a large document.

The simultaneous views approach is desirable when there is a large amount of information and only a small portion of it is particularly interesting, as with roadmap directions, where most information is concentrated around the starting and ending points, with long, uninteresting stretches over highways. Like the scroll pane model, simultaneous views are often easy to conceptualize for the user. However, navigation can become cumbersome, now that a user may need to worry about zooming and scrolling multiple views of the same data. Also, it is not always clear where in the overview display the detail display is supposed to be focused. This may be dealt with by drawing some sort of rectangle around the focused area, as in Acrobat Reader, but in many cases no such care is taken to orient the user in the visual space.

As there are many forms in which the focus+context problem presents itself, many solutions have been proposed over the last few decades. In the next section, some of these solutions are considered, as the various approaches relate to the topic presented in this work.

## 1.2 Previous work

Some place the beginning of work on the focus+context problem with Kadmon's and Shlomi's 1978 work in polyfocal displays for presentation of statistical data on cartographic maps [7]. However, Spence and Apperley did the first work addressing focus+context in regards to computer displays in 1982, with their proposal of the bifocal display [14]. It is important to understand this, as it can be viewed as a simplification of the tools that will be developed later in this work. In the bifocal display, the viewing area is divided up into three vertical sections. The middle section displays information at its normal size, while two smaller strips on either side give a compressed view of information that is logically to the left or right of what currently has focus. This allows more information to be displayed at once, although objects in the compressed area must somehow be scrolled into the main panel to be usable. It was not until 1995, with the help of Leung, that the bifocal display system was modified to give compressed views on the sides as well as the top and bottom of the display, creating nine different rectangular areas with different scales. While this further shrinks the main usable area, it may be useful for information that requires a large two-dimensional display rather than a primarily linear display.

In 1991, Mackinlay, Robertson, and Card developed a three-dimensional extension of the bifocal display, which they call the perspective wall [11]. The perspective wall acts something like a strip of rubber, with information mapped onto one side. The strip is then pulled across two vertical bars some distance apart to create a readable area between them, with the rest of the strip being pulled back into the distance. While this is a three-dimensional projection, the use of the space is fairly static, as the viewpoint never moves, and the only indication of the third dimension is in the perspective introduced by the strip's being pulled away from the viewpoint. In short, this behaves like a bifocal display with perspective and the ability to stretch the information space to be whatever resolution is appropriate. Obstacles to this interface include such issues as navigation and readability. Simple tasks such as moving the focus or stretching the perspective wall certainly would not be clear to a novice, and mapping information directly onto the surface of the perspective wall would require a reasonable resolution to display text clearly.

Four years later, Lamping, Rao, and Pirolli introduced the hyperbolic tree browser, inspired by an Escher woodcut [10]. This display is based on the hyperbolic geometry of the Poincaré disk. A hyperbolic geometry differs from familiar Euclidean geometry in that given a line $l$ and a point $P$ not on $l$, there are infinitely many lines through $P$ that do not intersect $l$. This is achieved in the Poincaré disk by defining geodesics, or straight lines, to be either lines through the center of the open disk or arcs of circles which are perpendicular where they intersect the boundary of the disk. The measure of distance changes as a point gets further from the center of the disk, so that an infinite distance can be drawn in any direction, but the disk fits within a finite area. This has the effect of giving objects near the center of the disk more room, while those near the edge appear closer together.

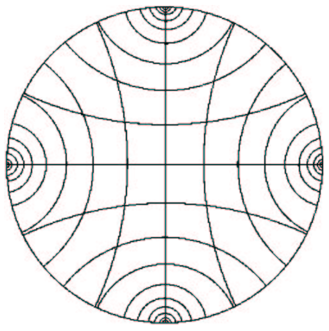

Figure 1.1: Hyperbolic geometry of the Poincaré disk

While the hyperbolic behavior of the Poincaré disk does not seem to be critical to the operation of the tree browser, it does cause lines between points near the edges to curve in an aesthetically pleasing way. However, this is sacrificed during interactive dragging, when lines between points are drawn straight in the Euclidean sense to keep the interface responsive. In fact, the Beltrami/Klein model of hyperbolic space may be used in place of the Poincaré disk, as Lamping, Rao, and Pirolloi mention in their paper. In this model, geodesics appear as straight lines, at the sacrifice of proper Euclidean angle measures between lines, which is not particularly important in the tree browser [5]. This allows the use of 3x3 matrices instead of complex arithmetic, but it tends to push structure to the edge of the disk too quickly, creating an area of great focus, with little detail in global context. However, there are other ways to map an infinite two-dimensional space to a disk, and even a rectangular area, that do not suffer from this problem.

## 1.3 Terminology

Some terms that are used throughout this work are defined here. While many models of hyperbolic geometry exist in mathematics, a *hyperbolic interface* or *hyperbolic browser* will refer only to one utilizing the Poincaré disk as described. This is not to be confused with the terms 1-hyperbolic, 2-hyperbolic, or hyperboloidal, which are described in the next chapter. Note that to be more accurate in their definitions, some of these terms do not have the same meaning as in [8]. *Focus+context* describes the general problem of presenting more information than can be reasonably displayed in a given viewing area; this is sometimes called the detail-in-context problem. *Information space* refers to some structure depicting information that may be rendered to a two-dimensional display. *Presentation space* refers to a rendered display of some information.

# Chapter 2

# The 1-Hyperbolic Projection

## 2.1 Hyperboloidal Geometry

The hyperbolic tree browser has some interesting properties, in that objects near the center are focused, while an infinite amount of space can be fit within an arbitrarily small disk. If a continuous display was available instead of discrete pixels, then any amount of information could be visible at once. Granted, it might be difficult to perceive the small detail near the edge of the disk; but this property seems to be satisfying in that there never has to be any navigational compensation for extremely large amounts of information. Scrollbars are unnecessary, as one can navigate the space by literally dragging it with a pointing device. While the dragging approach is possible with a scroll pane, it becomes unmanageable when the information space grows too large. In contrast, dragging nearer the edge in the hyperbolic case causes the space to move more at the center, since distance is not constant in the Poincaré disk. One question to consider might be whether there exist any other mappings that provide this sort of behavior. The answer is to be found in hyperboloidal projections.

A hyperboloid of two sheets is a surface of revolution in Euclidean three-space, obtained by rotating a hyperbola about its transverse axis, that is the line through its foci. For a hyperboloidal projection, only one of the sheets generated by this revolution is necessary. This sheet will necessarily be asymptotic to a cone of some angle centered on the transverse axis. One may obtain a method of projecting the entire surface of the hyperboloid onto an open circular disk by using this asymptotic cone as the bounds of a projection onto a plane orthogonal to the transverse axis and centered on the cone point. What's more, the distance between points near the center of the disk is much smaller than that of points near the edge, not unlike the Poincaré disk.

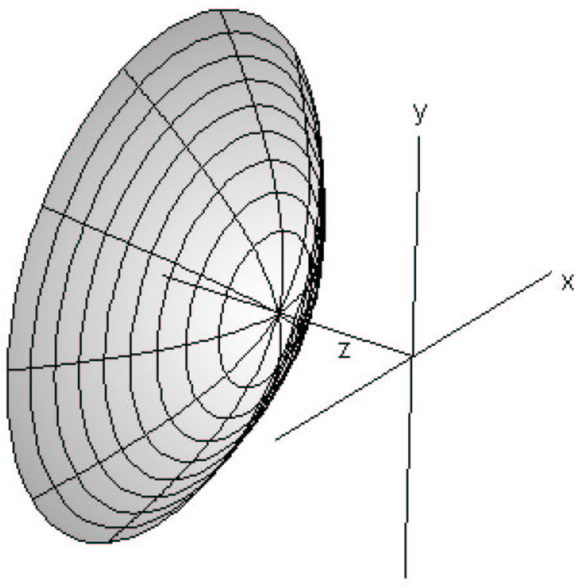

Figure 2.1: Half of a hyperboloid of two sheets

Rather than attempting to evenly map information onto the hyperboloid, information may be projected from a Euclidean plane onto the hyperboloidal surface and then projected onto the viewing plane. Straight lines of this hyperboloidal geometry are shown in

Figure 2.2.

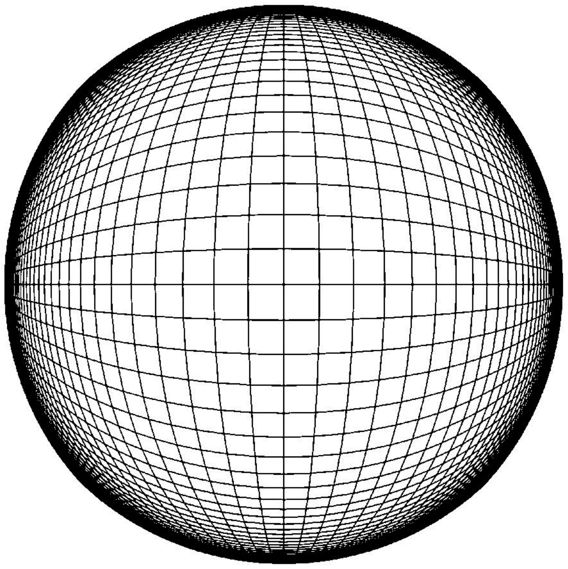

Figure 2.2: A hyperboloidal coordinate system

Clearly this space is not isometric to the Poincaré disk, nor can it be. It is a projection of the Euclidean plane, so given a line and a point not on the line, there is a unique parallel line though that point. However, this geometry shares many of the features that motivate the implementation of the hyperbolic interface.

## 2.2 A Simplified Approach

The hyperboloidal projection is a convenient Euclidean analogue to the Poincaré disk for hyperbolic geometry. This means it shares some of the faults that one might find in the hyperbolic interface. For instance, the hyperbolic interface must be embedded in a circle. To attempt to map this display to fill a rectangle would introduce unacceptable distortions that defeat the elegance of the hyperbolic model. Hence, anyone attempting to design an interface that incorporates a hyperbolic browser as part of a larger system must deal with the inconvenience of a circular component. While it is trivial to inscribe the disk within a square, this has multiple faults. For one, $(4-\pi)r^2$ units of display area are wasted by the unused dead space between the border of the disk and the edge of the bounding square. Also, an arbitrary rectangular bound is preferable, as this allows much more flexibility and freedom in designing a user interface. While the determined designer may successfully use the space around the disk with other interface components, such as oddly shaped buttons not unlike those used in the curious designs of some popular media playing applications, one cannot expect many to go so far out of their way to include a hyperbolic or hyperboloidal interface in an application. It seems that one must choose between traditional, easy to implement strategies such as scroll panes, or put much effort into effectively using some disk-based focus+context solution.

There is a third option available, if one is willing to give up the "infinite view" in one dimension of the hyperboloidal interface. In the projective model, rather than rotating a hyperbola about its transverse axis, one may translate the hyperbola along a vector orthogonal to the plane it rests in [8]. The surface this generates is depicted in Figure 2.3.

Rather than having a singular viewpoint, points may be projected onto a line segment that passes through the viewpoint and is parallel to the vector of translation, to further simplify the projection. This is called a 1-hyperbolic projection.

The effect this has is to give detail-in-context in one dimension, while space in the perpendicular dimension is unchanged, as the projection only changes distance measure in one direction. For simplicity, this work assumes that the projection always occurs in the vertical direction, so that objects on the same horizontal line have the same amount of focus, but objects on the same vertical line receive more focus as they approach the center. With this approach, an interface may naturally be given a rectangular area, unlike in the

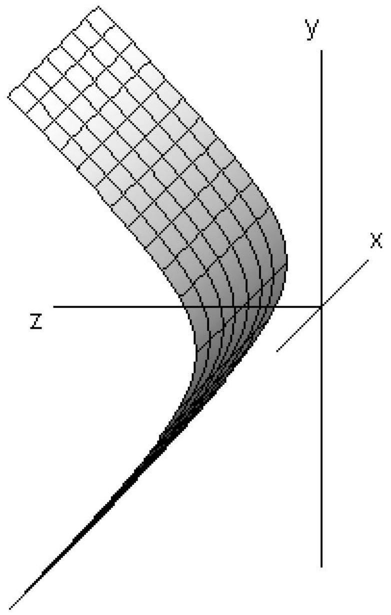

Figure 2.3: A 1-hyperbolic surface

hyperboloidal case. The downside to this is that the projection only influences presentation in one dimension, meaning that if there is two much information to display in the other dimension, a scrollbar might be necessary.

It might be helpful to consider more familiar geometries to fully appreciate the properties of hyperboloidal and 1-hyperbolic space. One can imagine that very roughly, the hyperboloid approximates a sphere. When one views a sphere, it will always appear to be a circle in two dimensions. Details on the surface of a sphere appear bigger toward the closest point of the sphere to the viewer. The sphere can then be rotated to bring some other information into focus. However, unlike the hyperboloid, a sphere is compact. Hence, only so much information can be wrapped around the surface of the sphere without scaling down the information. Unless details are made smaller as the required area grows, which will make even the focused region unintelligible at some point, eventually some detail will have to be mapped onto the back of the sphere, out of view. This situation never occurs on the hyperboloid, because of its asymptotic behavior to the cone that forms the boundary of the projection. Similarly, a cylinder set on a surface so that it can roll may be thought of as an approximation to the 1-hyperbolic projection. Rolling the cylinder brings structure up or down and into focus, but information on the surface of the cylinder may stretch indefinitely to the left or right, and cannot all be viewed at once. The 1-hyperbolic interface exhibits this sort of behavior, only detail can never wrap around out of view vertically, since the surface is asymptotic to two planes that make up boundaries of projection.

With a conceptual model of the 1-hyperbolic interface layed out, the next section develops the projection more concretely.

## 2.3 The Mathematics

Projecting points onto the viewing plane from a 1-hyperboic sheet is fairly straightforward, since the projection only affects position in one dimension. Let the center of the viewing line that points will be projected onto be at the zero of the coordinate system, with the $x$-axis being horizontal, the $y$-axis vertical, and the $z$-axis positive in the direction of the viewing angle. Let the bounds of projection be the planes $y = z$ and $y = -z$, and set the viewing plane to be $z = 1$. This geometry is depicted in Figure 2.4.

The generating hyperbola must have the $z$-axis as its transverse axis while being asymptotic to $y = z$ and $y = -z$. The equation $z^2 - y^2 = a^2$ defines a hyperbola that satisfies these conditions, where $a$ is any positive real number. Notice that as $z$ becomes very large, $a$ has a relatively small effect on the value of $y$, so that the equation grows to approximate $z^2 - y^2 = 0$ for large $z$ and $y$. Hence, it is not difficult to see that this hyperbola is

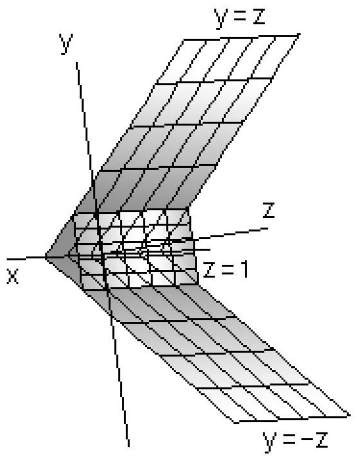

Figure 2.4: Setting up the geometry for the projection

asymptotic to the specified lines. Generating a surface of translation from this hyperbola by sweeping it across a line parallel to the $x$-axis results in the desired 1-hyperbolic surface. This, of course, creates a surface that is hyperbolic vertically; it is trivial to modify this construction to generate a surface that is hyperbolic horizontally instead.

With an equation for the 1-hyperbolic sheet, it is necessary to find a function that takes a point $(x, y)$ in the Euclidean plane to a point $(x_p, y_p)$ in the projective viewing plane. First, points must be projected from the Euclidean plane onto the 1-hyperbolic surface. Initially, represent each point in Euclidean 3-space by setting the $z$-value to zero, so $(x, y)$ becomes $(x, y, 0)$. All points on this plane will simply be projected into the 1-hyperbolic sheet along the line through them that is orthogonal to the plane $z = 0$. This does not affect the $x$ or $y$ values of the point; it only determines the depth of that point on the 1-hyperbolic surface. To determine the value of $z$ for any point in the Euclidean information space, only the $y$ coordinate is necessary, since the equation for the hyperbola, $z^2 - y^2 = a^2$, is in terms of $y$

and $z$. Solving this equation for $z$ yields $z = \sqrt{y^2 + a^2}$. Thus, a point $(x, y)$ is projected onto the hyperbola by $(x, y, \sqrt{y^2 + a^2})$.

Now points on the 1-hyperbolic surface are projected through the viewing plane onto the line of projection, in this case, the $x$-axis. By projecting points onto this line rather than a point, any modification of the $x$-value of a point is avoided, because points are projected along lines that are in some plane orthogonal to the $x$-axis. Thus, $x_p$, the projected value of $x$ in the viewing plane, is the same as the original value of $x$, as was originally intended. Figure 2.5 shows how some point at $(x, y, z)$ is projected onto the viewing plane.

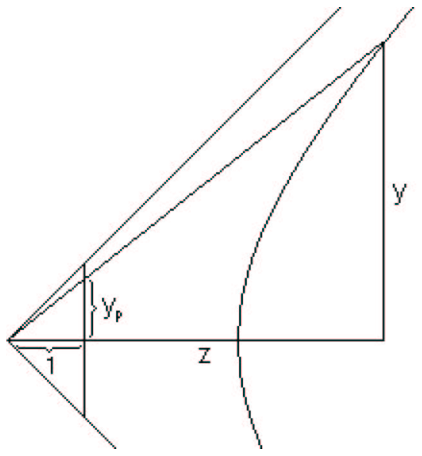

Figure 2.5: Projecting points with the 1-hyperbolic sheet

This point and the viewing plane generate two right triangles. Since they share all interior angles, the triangles are similar; and the ratio of the lengths of corresponding sides is the same. The bases of the triangles have a ratio of $z$ to 1, so $y = zy_p$. Hence, $y_p = \frac{y}{z} = \frac{y}{\sqrt{y^2+a^2}}$. Therefore, a point $(x, y)$ is mapped to $(x_p, y_p) = (x, \frac{y}{\sqrt{y^2+a^2}})$.

This mapping confirms the behavior that is expected of the 1-hyperbolic projection. As $y$ gets very large, $\frac{y}{\sqrt{y^2+a^2}}$ approaches 1. However, $\frac{y}{\sqrt{y^2+a^2}}$ is always less than 1, because

$a^2$ is positive, which forces the denominator to be larger than the numerator. Similarly, for negative $y$, it is the case that $y_p$ is greater than $-1$. Thus, $y$-values are mapped into $(-1, 1)$, no matter how large or small they are. For small $y$, the value of $a$ dominates in the denominator, forcing points to be close to the center. This is consistent with the notion of central focus that is expected of the projection.

It is interesting to note that the calculations here are sufficient for finding the mapping for a hyperboloidal projection. While the hyperboloidal projection affects both $x$ and $y$ coordinates, if one considers only the plane that shares a point on the hyperboloid and the transverse axis of the hyperboloidal sheet, then the projection in this plane is identical to that in the 1-hyperbolic case. One may rotate a point on the hyperboloid to be in the plane $x = 0$, project it using the 1-hyperbolic technique, and then rotate the point back to its original angle relative to the transverse axis. Rather than actually rotating the point twice, an equivalent transformation is to find the distance to the point from the transverse axis and then scale this length with the hyperbolic equation. That is, given a point $(x, y)$, its distance from the center is $d = \sqrt{x^2 + y^2}$, so the hyperboloidal distance should be $d_h = \frac{d}{\sqrt{d^2+a^2}}$. Then multiplying $(x, y)$ by $\frac{d_h}{d}$ results in the point being mapped into the viewing plane for the hyperboloidal projection. Thus, points in Euclidean space are taken into the unit disk using this projection.

## 2.4 Varying Curvature and Distance

So far, the value of $a$ has not yet been considered. For a hyperbola, this value determines the distance from the vertex to the intersection of the asymptotes. In the case of the 1-hyperbolic surface, that intersection is the $x$-axis. Hence, $a$ should be non-negative, since it is not helpful to think of a negative distance, especially when $a$ is squared anyway. It certainly does not help to let $a = 0$ either. This results in all values of $y$ being mapped to 1 or $-1$, unless $y = 0$, in which case a divide by zero would occur. Therefore, $a$ should always be positive.

Varying $a$ can have a dramatic affect on the appearance of the presentation space generated by a 1-hyperbolic projection. Not only does a change in $a$ affect the distance to the hyperbola, but the curvature must change as well, because changes in $a$ do not change the asymptotes of a hyperbola. When $a$ is small, the focus tends to be on a much smaller area of the information space, while the focus takes up more area in the presentation space. For large $a$, a larger area of the information space has focus, but the focus is not as pronounced as it is with smaller $a$. In fact, as $a$ approaches infinity, the projection approaches a state where the entire information space is given the same amount of focus in the presentation space. In this context, focus means the relative distance between points in the presentation space. So a point that is mapped into presentation space with a minimum of 10 units between it and points 1 unit away in the information space would be said to have a greater focus than a point that is 2 units away from the closest point in the presentation space that has 1 unit distance in the information space.

Figure 2.6 shows how different values of $a$ can affect the appearance of the projected presentation space.

## 2.5 Interacting with 1-Hyperbolic Space

The details of displaying space with a 1-hyperbolic projection are fairly clear, but some consideration must be given to how a user will interact with such a presentation. While the entire vertical information space may be viewed at once, because this projection maps any value into $(-1, 1)$, it is not practical or useful to expect the display to be static. To

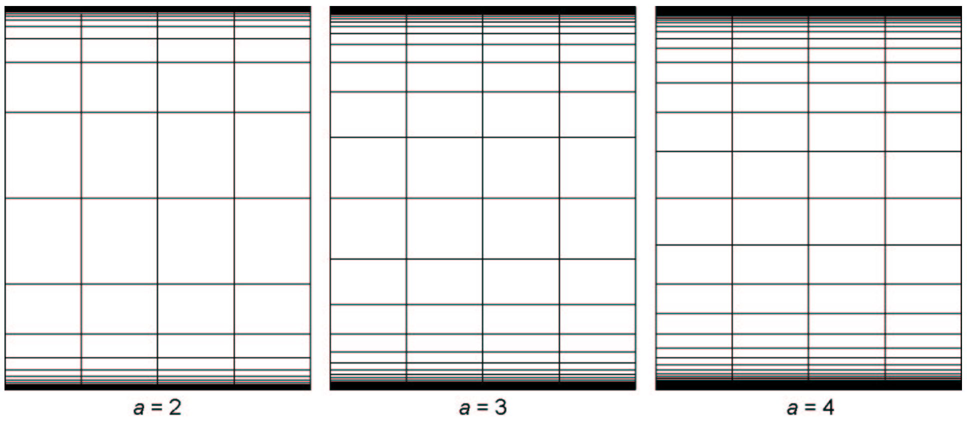

Figure 2.6: Varying distance to the 1-hyperbolic surface

scroll the information space up or down, bringing different parts of the information space into focus, all that is necessary is to rigidly translate the information space up or down and project it again. Now the question becomes how the user is to translate the information space up or down. A vertical scrollbar should be avoided, for reasons that have already been considered. Since there is no transformation of horizontal space, it is not unreasonable to use a horizontal scrollbar if it is necessary. For navigating vertical space, an approach similar to that of the hyperbolic tree browser seems most promising, where the user simply drags the space with the pointer [10]. Unlike non-Euclidean hyperbolic space, no rotations may be introduced by translations in the 1-hyperbolic projection, so dragging the space is not likely to introduce any confusion with the 1-hyperbolic projection. Allowing this one-to-one interaction with the interface may be more natural to the novice, compared to a scroll pane approach to displaying information. Dragging up causes information to be translated up, and dragging down translates information down. With a scroll pane, dragging the scrollbar's knob up actually moves the information down. This seems natural once one understands the conceptual model of the scroll pane, but without this understanding such behavior might be unexpected, if the novice even figures out that the interaction with the scrollbar is necessary to translate information up or down. While dragging is also possible in a flat presentation of information, it can become tiresome when the information space is very large. Since the 1-hyperbolic projection maps all information within a finite area, dragging any point into focus may be accomplished with a single motion of the pointer.

Now the problem is to find the amount of translation of the information space given some translation of the pointer during dragging by the user. Since the user is interacting with space that has been projected with the 1-hyperbolic surface, it is necessary to find the inverse projection that takes these points into the Euclidean plane that the information space exists in. Since the $x$-value of any projected point remains constant, it is only necessary to find the inverse projection for $y$. Solving the projection equation for $y$ requires only simple algebra and proceeds as follows:

$$\begin{aligned}
y_p &= \frac{y}{\sqrt{y^2+a^2}}, \\
y_p^2 &= \frac{y^2}{y^2+a^2}, \\
y_p^2 &= \frac{y^2+a^2-a^2}{y^2+a^2}, \\
y_p^2 &= 1-\frac{a^2}{y^2+a^2}, \\
y_p^2-1 &= -\frac{a^2}{y^2+a^2}, \\
1-y_p^2 &= \frac{a^2}{y^2+a^2}.
\end{aligned}$$

Since $y_p \neq 1$ and $y_p \neq -1$, it is the case that $1-y_p^2 \neq 0$. Also, $a$ is defined to be positive, so it is never 0. Thus, taking the reciprocal of both sides of the last equation is valid.

Continuing the derivation,

$$\begin{aligned}
\frac{1}{1-y_p^2} &= \frac{y^2+a^2}{a^2}, \\
\frac{a^2}{1-y_p^2} &= y^2+a^2, \\
\frac{a^2}{1-y_p^2}-a^2 &= y^2, \\
\sqrt{\frac{a^2}{1-y_p^2}-a^2} &= y^+.
\end{aligned}$$

This is an equation for $y$ in terms of $y_p$, but it requires a test for the sign of $y_p$ to determine whether $y$ should be positive or negative, which is the reason for the notation of $y^+$. This may be corrected, and more simplification is possible in order to reduce the number of operations:

$$\begin{aligned}
y^+ &= \sqrt{\frac{a^2-(a^2-a^2y_p^2)}{1-y_p^2}}, \\
y^+ &= \sqrt{\frac{a^2y_p^2}{1-y_p^2}}, \\
y &= ay_p\sqrt{\frac{1}{1-y_p^2}}.
\end{aligned}$$

Normally, the final step of pulling $y_p$ out of the radical would only be valid for $y_p \geq 0$, but this has the benefit of returning the sign of $y_p$ that is lost in the first step of the derivation, where both sides are squared.

When a user clicks on a draggable area of the presentation space, the pointer location may be projected back into the information space. As the pointer moves, the new point's location may be determined, and the information space may be translated by the difference.

Thus, when a user drags near the edge of the projection, content moves much further than when the user drags near the center. However, the dragging still appears natural, since objects next to the pointer remain next to it throughout dragging.

## 2.6 Considering Perspective

Up to this point, points on the 1-hyperbolic surface have been projected through the viewing plane onto the $x$-axis, to produce a pseudo-orthographic effect. There is no reason why these points cannot be projected onto the origin. This would make use of the three-dimensional space, so that objects approach the center horizontally as they get further from the center of the hyperbola. This has the effect of allowing more information to be displayed further from the center, but it is not without some problems.

The horizontal relationship between information can be lost using this model. With the standard 1-hyperbolic projection, points in the information space always have the same horizontal relationship with points as they do in the presentation space. This is not true with the perspective model. A point that is on one side of another point in the information space may appear to be on the other side when projected to the viewpoint. Points that lie on the same vertical line will not lie on a vertical line when displayed unless they happen to be in the center of the space. Also, the horizontal scrollbar becomes a problem, as the space is no longer flat horizontally. While using a scrollbar is still possible, its behavior might not be as predictable to a user as it would be otherwise. Dragging is possible, but if the space extends for a great distance horizontally, this can become tedious.

Despite some of the problems the perspective 1-hyperbolic interface can produce, there have been favorable comments about it when compared to a standard 1-hyperbolic interface. Even when the information fits within the presentation space horizontally, some have preferred the three-dimensional effect of the perspective technique for its aesthetics.

To implement a perspective 1-hyperbolic interface, only the $x$-coordinate needs to be transformed. It is not difficult to show that two points on the same horizontal line on the 1-hyperbolic sheet will be projected onto the same horizontal line of the viewing plane, using a triangle with the two points and the origin as vertices. This triangle must intersect the viewing plane at two points that are on the same horizontal line, as in Figure 2.7.

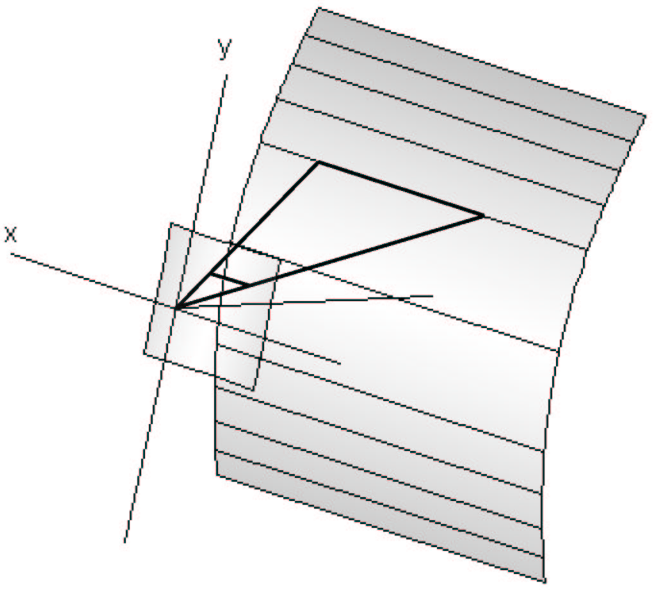

Figure 2.7: Horizontal lines are not distorted by perspective

Thus, the value of $y_p$ is unchanged from the standard 1-hyperbolic projection. Finding the value of $x_p$ for the perspective 1-hyperbolic projection again requires similar triangles. The triangles of figure 2.7 may be projected to the plane $y = 0$ which gives the same triangles as those of Figure 2.5 except that the side with length $y$ is replaced with $x$ and $y_p$ is replaced with $x_p$. Hence, $x_p = \frac{x}{z} = \frac{x}{\sqrt{y^2+a^2}}$. This behaves as is expected; points in the information space that are further from the $x$-axis have larger denominators, so they appear to be closer to the $y$-axis in the presentation space when projected to the viewpoint.

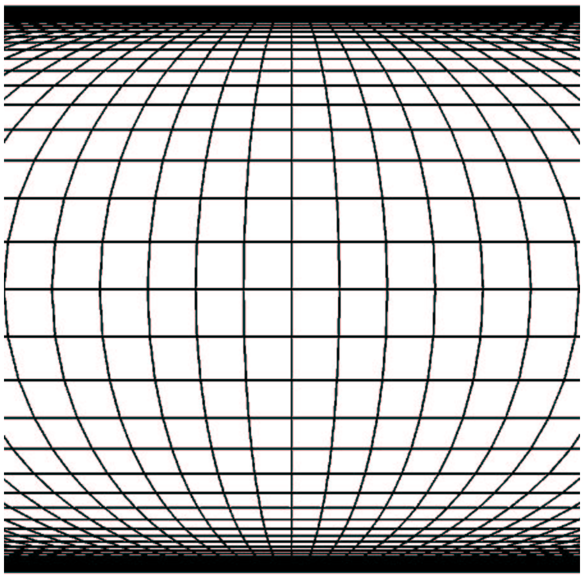

Figure 2.8: A 1-hyperbolic projection with perspective

## 2.7 The 2-Hyperbolic Projection

One final variation of the 1-hyperbolic projection to consider creates a display that shows an entire 2-dimensional Euclidean space within a finite area. While the hyperbolic tree browser and the similar hyperboloidal projection are capable of this, they both suffer the deficiency of displaying in a unit disk. The 2-hyperbolic projection, however, maps Euclidean space into a rectangle. The principle is similar to the extension of the bifocal display that compresses space in two dimensions. In the 2-hyperbolic projection, a standard 1-hyperbolic projection is applied to the information space vertically and horizontally simultaneously. Therefore, a point $(x, y)$ is projected to $(\frac{x}{\sqrt{x^2+a^2}}, \frac{y}{\sqrt{y^2+a^2}})$. The effect of this projection is demonstrated in Figure 2.9.

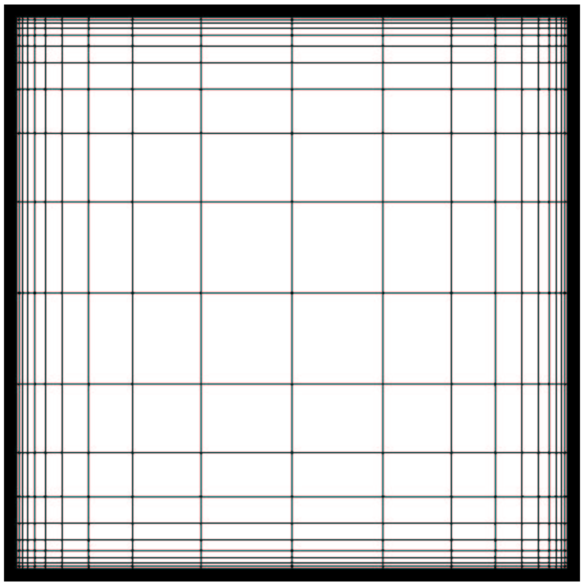

Figure 2.9: A 2-hyperbolic projection

The advantages of this interface echo those of the 1-hyperbolic interface, only the advantages are applied to both dimensions of some 2-dimensional information. No scrollbars are necessary at all, and the entire context of any information is visible at once. Problems can present themselves, depending on the application of the 2-hyperbolic interface. For example, it is usually not helpful to map text into a display using this technique, as this makes text near the edges difficult to read. If the 2-hyperbolic interface is only used to position text to be displayed flat, then text can spill over the edge of the borders of the viewport, requiring scrollbars again.

However, this interface might be useful for navigating primarily graphical information, such as a roadmap. Flat text labels can then be displayed only when they are close enough to the center so that the entire label will be visible. It is not unreasonable to imagine a treemap interface that makes use of a 2-hyperbolic display [16]. Treemaps can be useful tools for visualizing trees, especially when they have some visual component, but treemaps require a great deal of screen space, even for fairly small trees. With a 2-hyperbolic interface, treemaps might show more detail on each node without sacrificing much in navigability, since all context is visible at once with the 2-hyperbolic interface.

# Chapter 3

# Developing a Prototype

## 3.1 Specification

With the equations necessary for laying out and interacting with points in 1-hyperbolic projected space, creating a prototype is fairly straightforward. The only additional step in rendering is to scale from the range $(-1, 1)$ to $(0, h-1)$, where $h$ is the height of the viewport in pixels. Such a scaling is trivial to implement. Now it is necessary to determine how a prototype of a 1-hyperbolic interface should behave.

First, it must be decided what type of data structure to represent in the prototype and how it will be displayed in a flat Euclidean plane before mapping it to the presentation space. The interface is best suited to information that may be represented in a primarily linear fashion; but strictly linear data, such as a list, does not fully exploit the 1-hyperbolic's ability to show global context. A tree, on the other hand, may be more appropriate for the prototype. Using the tree representation common to many file system browsers seems ideal. With this model, all children of a node are aligned vertically under the parent, while being indented to set them apart. Nodes may be expanded or collapsed, to show or hide their children, respectively. This representation is favorable because only a small indent is necessary to set off children, so a tree must be very deep to require much horizontal area. On the other hand, vertical area can be consumed quickly, since every displayed node requires its own vertical space. The indenting of nodes has the benefit of allowing the 1-hyperbolic interface to show more structure for nodes at the periphery of the display. While the nodes may overlap, it is still clear where child and parent nodes are in the tree. Also, a list can be thought of as a limited tree, so implementing a prototype to display trees also allows one to see how linear lists appear in a 1-hyperbolic display. Hence, the prototype should be capable of displaying these vertically-oriented trees with collapsible nodes.

Some of the details of user interaction have already been considered. One of the advantages of a 1-hyperbolic interface is that at least one scrollbar is unnecessary—in theory, any information displayed in the viewport may be dragged into focus with one motion of the pointing device. So the user should be able to click in any area of the display and drag the pointer, with information following the vertical movements. However, the prototype must be able to differentiate between dragging and selecting of elements in the information space. A tree browser would be uninteresting if a user could only view a tree and not interact with it in any other way. Thus, the prototype should keep track of the distance of pointer movement during dragging. When the user stops dragging, if the distance moved by the pointer is less than some $\varepsilon$ and the pointer is over some node of the tree, then that node should be selected, giving the user some visual feedback.

The prototype must be responsive, whether dealing with small or large trees. It is not acceptable to have any delay in rendering while the user is dragging. Most users would quickly tire of waiting on an interface to catch up with them. Additionally, if movement isn't rendered at a decent frame rate, the smooth affect of the 1-hyperbolic interface will be lost; and it may not be clear how the interface behaves to the user.

## 3.2 Implementation

The time given to develop the prototype was very short, so it was created with JavaScript and Cascading Style Sheets for the Mozilla web browser. One of the primary motivations for this choice is that images and fonts are trivial to handle. Mozilla has excellent support for the PNG graphics format, which includes an option for an alpha channel in images. Using this feature, some interface elements may be made translucent. Thus, when one of these images overlaps some other item, the item in the background will not be completely obscured. Drop shadows may also be added to icons to add a sense of depth and distinctness to overlapping icons. Changing font sizes is trivial, so as nodes of the tree lose focus, their font may be made smaller to avoid overlapping text, which would result in an unreadable mess. Interaction is handled easily in JavaScript with event handlers that may be assigned to specific objects. When a user selects a node, the background of that node may have its color changed to indicate that it is selected. While JavaScript is not as flexible as many other languages, it offers all the features necessary for this prototype, in an environment that encourages quick development.

One nagging consideration was whether an interpreted language like JavaScript is capable of being as responsive as is necessary for the prototype. While the mathematics of the 1-hyperbolic interface are very simple, there is a problem in that even the slightest movements require that every single node of the tree be rendered again. To get the nodes to overlap, they must all be contained within their own style sheets, each assigned a $z$-index so that nodes that are closer to the viewing plane appear on top of those that are further away, as they should. Hence, Mozilla will be handling all rendering of nodes, with translucencies and all. This is favorable from an ease of development standpoint, but it depends on the speed of Mozilla's JavaScript and layer rendering implementation. Rather than deal with this potential problem from the beginning, it was decided to go ahead with development assuming all rendering will be "fast enough." Luckily this assumption was not unreasonable for small trees; thanks to Mozilla's relatively mature JavaScript implementation.

## 3.3 Results

The source code for the main JavaScript file may be found in Appendix A. This file is included in some HTML document that also includes a JavaScript file to set up a tree, by creating a `Tree` object and adding children to nodes with the `addChild` method of the `Node` class. A similar prototype for the 1-hyperbolic projection with perspective was also implemented, which could use the same tree constructing JavaScript files.

The first implementation, which satisfied the original specifications, worked reasonably well; but their were some important usability problems that have been addressed in the version of the interface reflected in Appendix A. The most serious issue is that of displaying large trees quickly. While rendering is decently fast for small trees, larger trees can introduce noticeable lag when many nodes are expanded. The situation is far from hopeless, however. When there are many nodes to be displayed, quite a few will be very close together near the edges of display. Not all of these nodes are necessary to get a feel for the structure of the tree. Indeed, when nodes overlap so much, it is difficult to tell what they are supposed to represent in the first place. The solution to this is to display nodes near the edges only if they give some important details of the structure of the tree. In the solution implemented here, a few decisions were made about what exactly is "important." In the `display` method of the `Node` class, there are a number of conditions that must be met to display a node that is considered too close to the last node rendered. If the node in question is not on the same level of the tree as the last node, then it will be rendered. This ensures that expanded parents show at least one child. Additionally, any node with children is displayed. This may cause problems in trees for which many nodes have children, but it was reasonable with the trees made for the prototype. Finally, the selected node is displayed at all times. This information reduction does cause some nodes to disappear when they approach the edge, but the effect is not too jarring, since these nodes tend to be difficult to differentiate from nearby nodes at that point. These conditions help to reduce the number of nodes displayed for large trees, and they tend to greatly help display speed.

Another problem is that there are no limits on how far a user may drag the space up or down. There is nothing keeping a user from dragging the entire tree up to the topmost few pixels, making the interface far from helpful. This also introduces a problem of getting the tree back into focus if a user accidentally drags it too far up or down. Dragging the pointer near the center of the display has very little effect on structure near the edges, and dragging near the edge can cause structure to move so far that the tree completely overshoots the area of focus and ends up clustered near the other edge. To deal with this, the prototype counts the number of displayed nodes whenever it renders the tree. Then the user is not allowed to drag the space too far past the first or last element. This is analogous to not allowing a user to use scrollbars to scroll ridiculously far past the displayed information, but it does introduce another potential annoyance. Suppose a user drags the space as far up as possible and then collapses some parent nodes. This reduces the number of nodes displayed, so that the bottommost node is higher than the limit should be. In the original solution, this meant that any dragging would cause the information to snap back down to the limit that it is supposed to be at, which is not predictable or desirable behavior. This was addressed by allowing the user to pull the space back down, but not pull it up while it is still above the limit. While it does seem a bit strange to be able to drag space one way, but not back to where it just was, this solution is much smoother and less disturbing.

One final unanticipated issue was the initial display of the tree. In the original design, the root node was placed in the center of the display initially. This only makes use of half of the available screen space, since all children are placed under the root. To better make use of the display area, the improved version of the interface counts the number of nodes to display, and focuses on the nodes closest to the middle of the list. While this can put the root node behind overlapping nodes, it makes optimal use of the available screen area for the default starting state.

The following figures show the appearance of the 1-hyperbolic interface prototype. Figure 3.1 shows the display with different values of $a$, the distance from the $x$-axis to the hyperbolic surface. From left to right, the values of $a$ are 4, 9, and 14 units. Figure 3.2 compares a standard 1-hyperbolic projection with one using perspective, on the left and right respectively. The perspective interface has a slightly smaller value of $a$ in this case to emphasize the curving effect.

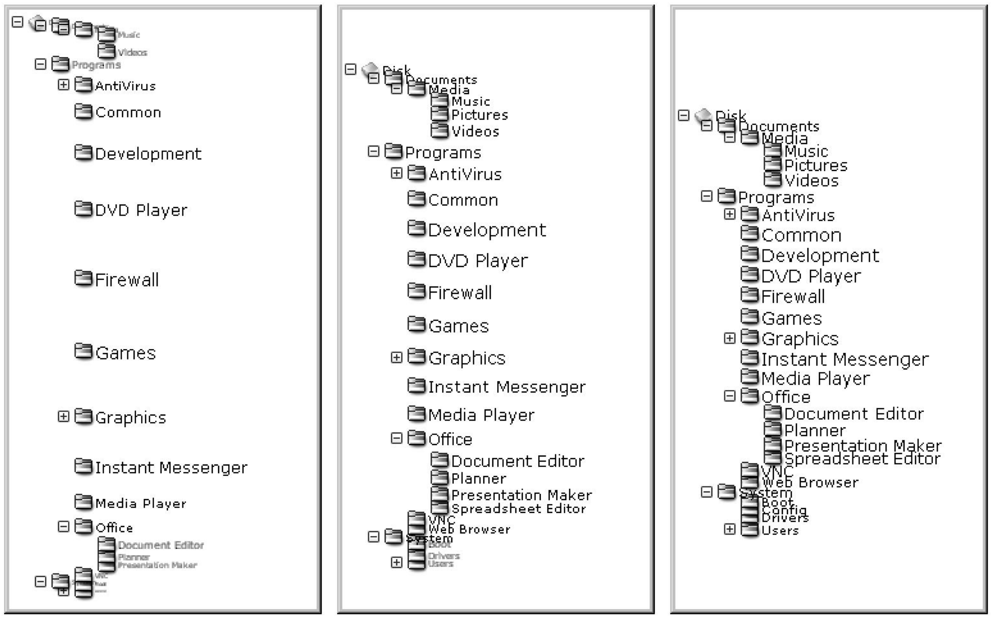

Figure 3.1: Comparing distance using the prototype

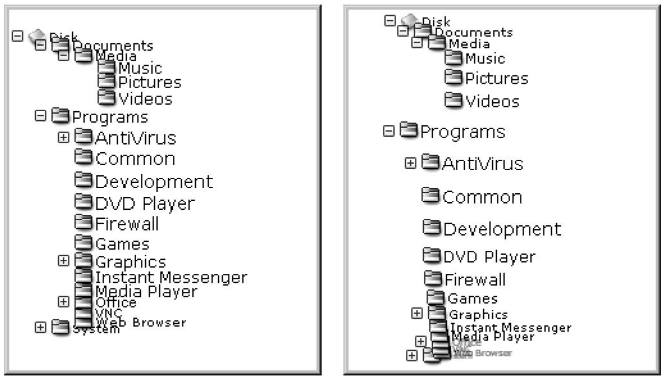

Figure 3.2: Comparing a standard 1-hyperbolic display with perspective

# Chapter 4

# Usability Studies

## 4.1 Overview

Many proposed solutions to the focus+context problem are accompanied by no laboratory experiments to evaluate the effectiveness of the solution [1, 3, 4, 9]. In cases where experiments are conducted, small numbers of subjects are typical [10] or details of the experiments are unclear [2]. Some of these works describe very general frameworks for information visualization, but the 1-hyperbolic projection is specific enough to merit a study comparing it to a standard scrollable interface. With an implementation of a flat, scrollable tree browser in JavaScript prepared as a control, it is possible to test the prototype on a group of subjects.

## 4.2 Experiment Design

The task given to the participants is to select a node in a tree, as this is something that a user might do with a tree often. When a user navigates a tree, many things happen—parents are expanded or collapsed, nodes are selected, scrolling occurs, and so on. The goal of the user is to select the correct node as quickly and painlessly as possible. Hence, only three variables are measured: time, accuracy, and preference. Time is measured as the number of milliseconds that passes from when a user is given a tree with instructions to a target node to when the user clicks a button indicating that the user believes the correct node is selected. Accuracy is a Boolean measurement; either the user selected the target node, or something went wrong. Preference is more subjective, as this is measured by responses to a survey at the end of the trials. While other things could be measured, such as the number of clicks the user makes, these three should suffice to evaluate the effectiveness of the 1-hyperbolic interface.

Each user is given a series of eight tasks to carry out using either the experimental interface or the control interface. Once all of the tasks are completed with one interface, they are repeated with the other. There is a serious problem in that the control interface is familiar to nearly anyone who has used a computer, while the 1-hyperbolic interface is new. This familiarity can cause problems in measurements, since the subjects are used to interacting with a scrollable tree interface and may be faster with it as a result. To reduce this bias, the experiment is modified in two ways. First, the experiment requires that subjects use a stylus and tablet as the pointing device. Since most computer users interact with a mouse, this requires the subjects to essentially learn how to interact with both interfaces as a beginner. Additionally, interacting with a stylus more closely resembles a handheld or tablet PC interface, both of which could benefit greatly from new interface techniques. Even with the stylus as a pointing device, subjects are still familiar with the behavior of a scrolling interface, while they will have to learn how dragging works in the 1-hyperbolic case. To deal with this bias, subjects are required to work through four unmeasured tutorials before the timed trials for both types of interfaces that train them in dragging or scrolling, selecting items, and expanding folders. These are intended to give the subjects some time to get used to the behavior of the interfaces and the tablet before the measured trials. Additionally, subjects are randomly selected to either perform the tasks with the control interface first, followed by the experimental interface, or they use the experimental interface first, followed by the control interface.

The purpose of each of the trials is given in Table 4.1.

| | |
|---|---|
| **Trial 1** | Expanding a node to select a child |
| **Trial 2** | Selecting a node that shares its name with a node under a different parent |
| **Trial 3** | Selecting a node near the middle of a long, linear list |
| **Trial 4** | Selecting a node near the top of a long, linear list |
| **Trial 5** | Selecting a node near the bottom of a long, linear list |
| **Trial 6** | Scrolling and expanding folders in a large tree |
| **Trial 7** | Finding a node deep and near the bottom in a large tree |
| **Trial 8** | Finding a node near the top of a large tree |

Table 4.1: Purposes of each experimental trial

The experiment was conducted on 41 undergraduates, who nearly all reported that they use computers regularly and are comfortable using them.

## 4.3 Analyzing the Results

With 41 participants performing 16 tasks each, there is a sizable amount of information to analyze. First, consider the accuracy results, given in Table 4.2. Each row shows the statistics for one trial, where E and H stand for Euclidean (scrollable control) and 1-hyperbolic, respectively.

The accuracy results are similar for the Euclidean and 1-hyperbolic cases. It is interesting that so many subjects did poorly with trial 7. It seems that in this case, several of the participants had trouble following the directions. The only other trial where more than one subject performed incorrectly for a particular interface is trial 2. This is not unexpected,

| **Trial** | **Correct selections** | **Incorrect selections** | **Percent correct** |
|---|---|---|---|
| E 1 | 40 | 1 | 98% |
| H 1 | 40 | 1 | 98% |
| E 2 | 37 | 4 | 90% |
| H 2 | 39 | 2 | 95% |
| E 3 | 41 | 0 | 100% |
| H 3 | 41 | 0 | 100% |
| E 4 | 41 | 0 | 100% |
| H 4 | 40 | 1 | 98% |
| E 5 | 41 | 0 | 100% |
| H 5 | 40 | 1 | 98% |
| E 6 | 41 | 0 | 100% |
| H 6 | 41 | 0 | 100% |
| E 7 | 30 | 11 | 73% |
| H 7 | 27 | 14 | 66% |
| E 8 | 41 | 0 | 100% |
| H 8 | 41 | 0 | 100% |

Table 4.2: Statistics for accuracy results

since the task here requires one to pay attention to the parent node of the node that is to be selected.

The question now is whether there is any significant difference in the accuracy of the two interfaces. The null hypothesis is that there is no difference in the subjects' accuracy using the two interfaces, and the alternative hypothesis is that there is a significant difference in accuracy between the two interfaces. Rather than treating each trial separately, it is necessary to sum the accuracy results (1 for correct, 0 for incorrect) of the trials for each subject so that means may be calculated for the paired samples $t$-test. This results in a mean of 7.61 for the control interface and 7.54 for the experimental interface, and the standard deviations are .628 and .636, respectively. However, with $t_{(40)} = .595$, the significance is .555 for the two-tailed test, well over the .05 significance level. Therefore, there may be no conclusions drawn about any difference in the results for selection accuracy.

There is much more information to consider when looking at the timed performance of each trial. Table 4.3 gives an overview of some statistics for the time measurements. All times are in seconds, and those trials where the subject selected an incorrect node are thrown out.

| **Trial** | **Average** | **Median** | **Minimum** | **Maximum** | **Standard Deviation** |
|---|---|---|---|---|---|
| E 1 | 16.6 | 15.0 | 9.5 | 34.5 | 5.6 |
| H 1 | 21.6 | 18.5 | 9.1 | 90.2 | 13.3 |
| E 2 | 7.3 | 6.8 | 3.4 | 15.4 | 2.5 |
| H 2 | 8.7 | 7.1 | 4.5 | 23.1 | 4.2 |
| E 3 | 10.0 | 9.5 | 6.1 | 26.2 | 3.4 |
| H 3 | 7.4 | 6.9 | 4.2 | 19.5 | 3.0 |
| E 4 | 6.2 | 5.6 | 3.6 | 22.3 | 3.0 |
| H 4 | 13.7 | 12.1 | 5.9 | 38.9 | 6.7 |
| E 5 | 9.1 | 8.0 | 5.1 | 20.5 | 3.4 |
| H 5 | 12.8 | 11.4 | 5.2 | 39.1 | 6.6 |
| E 6 | 15.9 | 15.4 | 6.7 | 27.1 | 4.8 |
| H 6 | 21.8 | 19.0 | 9.0 | 54.2 | 9.5 |
| E 7 | 30.3 | 28.8 | 14.0 | 62.6 | 10.4 |
| H 7 | 27.6 | 24.6 | 17.2 | 46.1 | 8.0 |
| E 8 | 12.0 | 10.8 | 5.3 | 23.2 | 3.9 |
| H 8 | 15.9 | 14.6 | 7.9 | 47.2 | 7.5 |

Table 4.3: Statistics for time results

The only trials that seem to favor the 1-hyperbolic interface are the third and seventh. Since the 1-hyperbolic prototype initializes the display so that elements near the middle of a structure are in focus, the results of trial 3 make sense, given that the task here is to select a node near the middle of a long, linear list. Trial 7 required quite a bit of scrolling compared to the other trials, so this may have favored the 1-hyperbolic interface as well.

Analysis of the time proceeds much as that for the accuracy, using the paired *t*-test, only with trials where the incorrect item was selected removed. In this case, the means for the control and experimental interfaces are 12.5 and 15.8 seconds, with standard deviations of 3.45 and 4.95, respectively. Unlike the results for accuracy, the time results are quite significant. With $t_{(40)} = 4.232$, the results have a less than .001 probability of being due to chance. So with the 1-hyperbolic interface, it typically takes about 25% longer to complete a task.

This in no way means that the 1-hyperbolic interface is a lost cause. As has already been noted, the subjects are all familiar with the control interface. While measures were taken to minimize the effects of this, it cannot be expected that using a stylus and a few untimed trials is enough to level one's experience with the two interfaces. It might be that these recorded times are partly the result of inexperience with the new interface. The survey responses also give a different feel for how the subjects perceived the difference in the interfaces. Table 4.4 shows the frequency of responses to some of the more interesting questions.

| | |
|---|---|
| *Which did you find to be easier to use?* | |
| Euclidean | 20 |
| No preference | 7 |
| 1-hyperbolic | 14 |
| *Which did it seem quicker to complete tasks with?* | |
| Euclidean | 20 |
| No preference | 1 |
| 1-hyperbolic | 20 |
| *Which would you rather use on a daily basis?* | |
| Euclidean | 21 |
| No preference | 3 |
| 1-hyperbolic | 17 |

Table 4.4: Survey responses

It is curious that the subjects were split so evenly on the question of which interface seemed to allow for the quickest completion of tasks, even when this does not entirely reflect reality. Perhaps as strange is that so few subjects responded with no preference, selecting one of the interfaces instead. Apparently, most of the subjects perceived some difference in their ability to work in the two interfaces. The comments given by participants may offer some insight into these perceptions.

One subject commented that “The rolling browser seemed to work a lot better, because you can fairly easily see what is above and below where you currently are.” Another subject noted that “The rolling browser very quickly lets the user see the contents of a directory. This is especially useful for directories containing large numbers of items. It is very easy to use.” Not all responses were positive, though. Some participants expressed frustration with the tablet and stylus input system. One subject wrote that “The pen thingy was more irritating than either type of browser,” and another stated quite simply, “the pen is cinfusing [sic].” It might have been preferable to use a computer where the stylus can be used directly on the screen, such as with a tablet PC, to accommodate these users; but unfortunately, no such systems were available to the experimenter. A few subjects noted their displeasure with the 1-hyperbolic browser. One comment was “I don’t like the scrolling feature on the rolling browser. It just gives me a headache.” Still others speculated that they might get faster with the 1-hyperbolic browser if they worked with it more regularly. Given the enthusiasm and openness some showed for the experimental interface, it would be difficult to say that there is no use for it in real applications.

# Chapter 5

# Conclusions

## 5.1 Contributions

The 1-hyperbolic interface shows promise as a complement to the scrollable interfaces that most computer users know. Unlike some proposed focus+context solutions, the 1-hyperbolic projection is specific and practical enough to implement in interfaces that a normal desktop user might encounter, rather than being confined to research applications. With the great processing power in even modest desktop computers, visual effects such as translucency and dynamic icon and font sizing can enhance the appearance of a 1-hyperbolic browser in ways that make its behavior more clear to the user. With even a rather naïve approach to information reduction, very large trees may be displayed and interactively manipulated, as has been shown with an implementation in an interpreted scripted language that runs reasonably well on a 500 MHz machine with no hardware graphics acceleration.

## 5.2 Future Directions

The usability experiments conducted were an excellent resource for examining the utility of the 1-hyperbolic interface, but they were still quite limited in scope. Future experiments might involve different input systems, such as tablet PC's, handheld organizers, and the familiar mouse. In the author's experience, a stylus interface feels more natural than a mouse with the 1-hyperbolic interface, but this can be a barrier for those unaccustomed to using a tablet. It might also be helpful to examine the experiences of complete novices with 1-hyperbolic and scrollable interfaces.

There might be other useful ways of extending or modifying the behavior of the 1-hyperbolic interface that have not been considered here. For example, it might be helpful to allow the user to dynamically alter the distance to the hyperbolic sheet to change the curvature of the surface and the amount of focus on elements near the center. However, it is not desirable to have some control displayed all the time for a function that would likely go unused for long periods of time, but hiding such functionality away in some menu is probably not a good solution either. It might be useful to have keyboard shortcuts to modify this distance on the fly. Another approach could be to allow a key combination to activate a temporary scrollbar at the pointer position to allow the user to finely manipulate the value.

## 5.3 Concluding Remarks

While the 1-hyperbolic interface may not be completely revolutionary, there is some merit in this. To depart completely from familiar techniques of interaction can leave users confused and unreceptive to new experiences. The principles required to implement a 1-hyperbolic browser are not too far removed from those of other interfacing elements, making it more likely that developers will create 1-hyperbolic components for desktop software systems. This is especially true in the open source case, where projects such as KDE and Mozilla have shown a responsiveness to new technologies. It is the author's hope that eventually the 1-hyperbolic interface will be as pervasive as similar scroll pane interfaces, at least as an option.

# Bibliography

[1] Keith Andrews and Helmut Heidegger. Information Slices: Visualising and Exploring Large Hierarchies Using Cascading, Semi-Circular Discs. In *IEEE Symposium on Information Visualization 1998*. IEEE Computer Society, October 1998.

[2] Lyn Bartram, Albert Ho, John Dill, and Frank Henigman. The Continuous Zoom: A Constrained Fisheye Technique for Viewing and Navigating Large Information Spaces. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 207–215. ACM Press, November 1995.

[3] Cava and Freitas. Visualizing Hierarchies Using a Modified Focus+Context Technique, 2001. `http://www.inf.ufrgs.br/~carla/papers/CavaFreitas-Infovis.pdf`.

[4] G. W. Furnas. Generalized Fisheye Views. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23. ACM Press, 1986.

[5] David W. Henderson. *Experiencing Geometry in Euclidean, Spherical, and Hyperbolic Spaces*. Prentice Hall, 2001.

[6] Don Hopkins. Pie Menu Central, 2003. `http://www.piemenus.com/`.

[7] N. Kadmon and E. Shlomi. A Polyfocal Projection for Statistical Surfaces. *The Cartographic Journal*, 15:36–41, June 1978.

[8] Alexander Kolliopoulos. Visualizing Trees with a Hyperbolic Projection in one Dimension. *Journal of Computing Sciences in Colleges*, 18(4):133–138, April 2003.

[9] Matthias Kreusler and Norma López Heidrun Schumann. A Scalable Framework for Information Visualization. In *IEEE Symposium on Information Visualization 2000*, pages 27–36. IEEE Computer Society, October 2000.

[10] J. Lamping, R. Rao, and P. Pirolli. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. In *Conference proceedings on Human factors in computing systems*, pages 401–408. ACM Press, May 1995.

[11] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The Perspective Wall: Detail and Context Smoothly Integrated. In *ACM Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, pages 173–179. ACM Press, 1991.

[12] Mozilla Mouse Gestures, 2003. `http://optimoz.mozdev.org/gestures/`.

[13] Mouse Gestures In Opera, 2003. `http://www.opera.com/features/mouse/`.

[14] R. R. Spence and Apperley. Data Base Navigation: An Office Environment for the Professional. *Behaviour and Information Technology*, 1:43–54, 1982.

[15] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone Trees: Animated 3D Visualizations of Hierarchical Information. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, pages 184–194. ACM Press, 1991.

[16] Ben Shneiderman. Treemaps for Space-Constrained Visualization of Hierarchies, 2002. `http://www.cs.umd.edu/hcil/treemap-history/index.shtml`.

# Appendix A

# 1-Hyperbolic Interface Prototype

```
/////////////
// Globals //
/////////////

var img_plus = new Image();     img_plus.src = 'plus.png';
var img_minus = new Image();    img_minus.src = 'minus.png';
var IMG_PLUS_MINUS_WIDTH = 13;
var IMG_PLUS_MINUS_HEIGHT = 11;
var IMG_ICON_WIDTH = 18;
var IMG_ICON_HEIGHT = 16;
var INDENT_SIZE = IMG_ICON_WIDTH;
var LINE_HEIGHT = 18;
var FONT_SIZE = 14;
var VIEW_HEIGHT = 330;
var HYPERBOLIC_DISTANCE = 9;
var MAX_NODES = 1024;

/////////////
// Classes //
/////////////

// Tree class
function Tree(img_icon, str_label, str_id, expand) {
        if (expand == undefined)
                expand = true;
        this.nodes = new Array();
        this.root = new Node(img_icon, str_label, str_id, this, expand);
        this.nodes[str_id] = this.root;
        this.selected = null;
        this.focus = 1;
        // Used for counting number of displayed nodes
        this.position = 0;
}

Tree.prototype.select = function(select_id) {
        if (this.selected != null) {
                document.getElementById('table_' + this.selected)
                        .style.backgroundColor = 'transparent';
        }

        this.selected = select_id;
        document.getElementById('table_' + select_id)
                .style.backgroundColor = '#88ccff';
}

Tree.prototype.redisplay = function() {
        this.draw_state = (this.draw_state + 1) % 256;

        // Draw root
        this.position = 0;
        this.root.display(0);

        // Draw children of root; if expanded,
        // draw their children recursively
        this.position = 1;
        if (this.root.expanded)
                this.root.displayChildren(1);
}

Tree.prototype.center = function() {
        // Get the number of displayed nodes in this.position
        this.redisplay();
        this.focus = Math.floor(this.position / 2);
        this.redisplay();
}

// Tree node class
function Node(img_icon, str_label, str_id, tree_obj, expand) {
        this.tree = tree_obj;
        this.parent_node = null;
        this.children = new Array();
        this.icon = img_icon;
        this.label = str_label;
        if (expand != undefined)
                this.expanded = expand;
        else
                this.expanded = false;
        this.id = str_id;
}

Node.prototype.displayChildren = function(depth) {
        window.prev_top_y = -10;
        window.prev_depth = -10;

        for (var i = 0; i < this.children.length; ++i) {
                // Draw child
                this.children[i].display(depth);

                ++this.tree.position;

                // Draw its children if necessary
                if (this.children[i].expanded)
                        this.children[i].displayChildren(depth + 1);
        }
}

Node.prototype.display = function(depth) {
        var original_y = this.tree.position - this.tree.focus;
        // Hyperbolic function returns center of line,
        // so must correct to get top
        var top_y = Math.round(hyperbolic(original_y)
                               - (LINE_HEIGHT / 2) + 1);

        var indent = depth * INDENT_SIZE;
        if (this.children.length == 0)
                indent += IMG_PLUS_MINUS_WIDTH;

        // Information reduction for top and bottom
        if (Math.abs(window.prev_top_y - top_y) < 6
            && window.prev_depth == depth && this.children.length == 0
            && this.tree.selected != this.id) {
                // Don’t show nodes that are too close to a sibling
                document.getElementById(’node_div_’ + this.id)
                        .style.display = ’none’;
        } else {
                window.prev_top_y = top_y;
                window.prev_depth = depth;

                var font_size = HYPERBOLIC_DISTANCE * FONT_SIZE
                                 / zDistance(original_y);

                document.getElementById('node_div_' + this.id).style.top =
                        top_y + 'px';
                document.getElementById('node_div_' + this.id).style.left =
                        indent + 'px';
                document.getElementById('table_' + this.id)
                        .style.fontSize = font_size + 'px';
                document.getElementById('node_div_' + this.id)
                        .style.zIndex = MAX_NODES
                        - parseInt(Math.abs(this.tree.position
                                            - this.tree.focus));
                // Ensure visible nodes are redrawn when expanding
                document.getElementById('node_div_' + this.id)
                        .style.display = 'block';
        }
}

Node.prototype.addChild = function(img_icon, str_label, str_id, expand) {
        if (expand == undefined)
                expand = false;
        var current_node = new Node(img_icon, str_label, str_id,
                                  this.tree,expand);
        current_node.parent_node = this;
        this.children.push(current_node);
        this.tree.nodes[str_id] = current_node;
        return current_node;
}

Node.prototype.create = function(tree_name) {
        var str = '<table border="0" cellpadding="0"'
                   + ' cellspacing="0" id="table_'
                   + this.id + '"><tbody><tr>';

        if (this.children.length > 0) {
                str += '<td width="' + IMG_PLUS_MINUS_WIDTH + '">';
                str += '<a href="#" onMouseUp="if (window.no_drag) ';
                str += tree_name + '.nodes[\'' + this.id;
                str += '\'].toggleExpand(); return false;">';
                str += '<img src="' + (this.expanded ? img_minus.src
                                                     : img_plus.src);
                str += '" width="' + IMG_PLUS_MINUS_WIDTH;
                str += '" height="' + IMG_PLUS_MINUS_HEIGHT;
                str += '" border="0" id="expand_toggle_';
                str += this.id + '"></a></td>';
        }

        str += '<td width="' + IMG_ICON_WIDTH + '">';
        str += '<a href="#" onMouseUp="if (window.no_drag) ';
        str += tree_name + '.select(\'';
        str += + this.id + '\'); return false;">';
        str += '<img src="' + this.icon + '" width="' + IMG_ICON_WIDTH;
        str += '" height="' + IMG_ICON_HEIGHT + '" border="0"></a></td>';

        str += '<td nowrap="nowrap" class="treelabel">';
        str += '<a href="#" onMouseUp="if (window.no_drag) ';
        str += tree_name + '.select(\'' + this.id;
        str += '\'); return false;">';
        str += this.label + '</a></td>';

        str += '</tr></tbody></table>';

        var node_div = document.createElement('div');
        document.body.appendChild(node_div);
        node_div.setAttribute('id', 'node_div_' + this.id);
        node_div.style.position = 'absolute';
        node_div.style.display = 'none';
        node_div.innerHTML = str;

        for (var i = 0; i < this.children.length; ++i)
                this.children[i].create(tree_name);
}

Node.prototype.toggleExpand = function() {
        if (this.expanded) {    // Collapse
                // Re-render all nodes
                for (var node_id in this.tree.nodes) {
                        document.getElementById('node_div_' + node_id)
                                .style.display = 'none';
                }

                this.expanded = false;
                document.images['expand_toggle_' + this.id].src
                        = img_plus.src;
                this.tree.redisplay();
        } else {        // Expand
                this.expanded = true;
                document.images['expand_toggle_' + this.id].src
                        = img_minus.src;
                this.tree.redisplay();
        }
}

///////////////
// Functions //
///////////////

function init() {
        // Display tree
        tree.root.create('tree');
        tree.center();

        // Handle dragging
        window.addEventListener('mousedown', startDrag, false);
}

function startDrag(e) {
        e.preventDefault();
        window.last_client_y = e.clientY;
        window.first_client_y = e.clientY;
        window.last_y = inverseHyperbolic(e.clientY);
        window.onmousemove = handleDrag;
        window.onmouseup = endDrag;
        window.no_drag = true;
}

function handleDrag(e) {
        if (window.no_drag
            && Math.abs(window.first_client_y - e.clientY) > 4) {
                window.no_drag = false;
        }

        FOCUS_MIN = -2;
        FOCUS_MAX = tree.position + 1;

        if (e.clientY != window.last_client_y) {
                if ((tree.focus == FOCUS_MIN
                     && e.clientY > window.last_client_y)
                    || (tree.focus == FOCUS_MAX
                     && e.clientY < window.last_client_y)) {
                        return;
                }
                var current_y = inverseHyperbolic(e.clientY);
                var delta_y = window.last_y - current_y;

                // Keep user from scrolling ridiculously far up or down
                if (tree.focus + delta_y < FOCUS_MIN
                    && e.clientY > window.last_client_y) {
                        tree.focus = FOCUS_MIN;
                        tree.redisplay();
                } else if (tree.focus + delta_y > FOCUS_MAX
                           && e.clientY < window.last_client_y) {
                        // When collapsing nodes puts us past upper limit,
                        // don't snap down
                        if (tree.focus <= FOCUS_MAX) {
                                tree.focus = FOCUS_MAX;
                                tree.redisplay();
                        }
                } else {
                        tree.focus += delta_y;
                        tree.redisplay();
                        window.last_y = current_y;
                        window.last_client_y = e.clientY;
                }
        }
}

function endDrag(e) {
        window.onmousemove = null;
}

function hyperbolic(original_y) {
        // Hyperbolic map to range (-1, 1)
        var y = original_y / Math.sqrt(original_y * original_y
                             + HYPERBOLIC_DISTANCE * HYPERBOLIC_DISTANCE);

        // Stretch to fit viewport coordinates,
        // taking into account line height
        // (-1, 1) => (LINE_HEIGHT / 2, VIEW_HEIGHT - LINE_HEIGHT / 2)
        return (y * (VIEW_HEIGHT - LINE_HEIGHT + 1)
                + (VIEW_HEIGHT - 1)) / 2;
}

function inverseHyperbolic(viewport_y) {
        viewport_y = forceValidY(viewport_y);
        // Stretch from viewport coordinates to (-1, 1)
        // (LINE_HEIGHT / 2, VIEW_HEIGHT - LINE_HEIGHT / 2) => (-1, 1)
        var y = (2 * viewport_y - VIEW_HEIGHT + 1)
                / (VIEW_HEIGHT - LINE_HEIGHT + 1);

        // Inverse hyperbolic map to information space coordinates
        return y * HYPERBOLIC_DISTANCE * Math.sqrt(1 / (1 - y * y));
}

function zDistance(y) {
        return Math.sqrt(y * y
               + HYPERBOLIC_DISTANCE * HYPERBOLIC_DISTANCE);
}

function forceValidY(viewport_y) {
        if (viewport_y < LINE_HEIGHT / 2)
                return LINE_HEIGHT / 2;
        if (viewport_y > VIEW_HEIGHT - LINE_HEIGHT / 2 - 1)
                return parseInt(VIEW_HEIGHT - LINE_HEIGHT / 2 - 1);
        return viewport_y;
}
```