

University of Nevada, Reno

# **Interactive Internet Visualization from the Inside of a Sphere**

A thesis submitted in partial fulfillment of the  
Requirements for the degree of Master of Science  
with a major in Computer Science

by

David Smith Shelley

Dr. Mehmet Gunes - Thesis Advisor

December, 2010



THE GRADUATE SCHOOL

We recommend that the thesis  
prepared under our supervision by

**DAVID SMITH SHELLEY**

entitled

**Interactive Internet Visualization from the Inside of a Sphere**

be accepted in partial fulfillment of the  
requirements for the degree of

**MASTER OF SCIENCE**

Mehmet H. Gunes, Ph.D., Advisor

Frederick C. Harris, Jr., Ph.D., Committee Member

Cansin Y. Evrenosoglu, Ph.D., Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

December, 2010

# Abstract

The rapid expansion of the Internet has made understanding its intricate connections difficult. Hence, advanced techniques are needed to understand the underlying network. This thesis introduces a novel way to address the issue of visualizing the large-scale topology of the Internet. The proposed Inner Sphere visualization method projects the network topology on the inside of a sphere. User navigation around the network is accomplished through moving the sphere around the user's point of view. This approach to the visualization of large-scale networks builds upon previous work in the visualization and navigation of large data sets. Since previous research has shown that the spatial cognition ability in humans greatly affects the usefulness of a user interface, two empirical experiments were performed that test the usefulness of viewing topologies from the inside of a sphere compared to a flat surface. The results of our study show that network navigation on a sphere is faster but can also be confusing. Our Inner Sphere visualization method is implemented in a new tool for interactive network visualization called GerbilSphere. GerbilSphere takes into account the results of many user studies in order to create a more intuitive user interface.

# Acknowledgments

I would like to thank my advisor Dr. Mehmet Gunes for the support and guidance in this endeavor. I would also like to thank Dr. Frederick Harris, Jr. and Dr. Cansin Yaman Evrenosoglu for serving on my committee.

I also would like to thank my wife Melissa for her support. Without her I would never have been able to stay in school to finish this advanced degree.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>ii</b>
<b>List of Tables</b>	<b>vi</b>
<b>List of Figures</b>	<b>vii</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
<b>Chapter 2 Background</b>	<b>3</b>
2.1 Tree Network Layouts . . . . .	4
2.1.1 2D Trees . . . . .	4
2.1.2 3D Cone Tree Layouts . . . . .	6
2.2 Force-Directed Network Layouts . . . . .	7
2.2.1 Spring Force Model . . . . .	8
2.2.2 Graph Theoretic Force Model . . . . .	12
2.2.3 Multilevel Decomposition Model . . . . .	13
2.2.4 3D and Non-Euclidean Force-Directed Layouts . . . . .	13
2.3 Interactive Networks . . . . .	14

	iv
2.3.1	Zoom + Pan . . . . . 15
2.3.2	Focus + Context . . . . . 15
2.3.3	Node and Edge Selection . . . . . 16
2.3.4	Multi-level Visualization . . . . . 17
2.4	2D vs 3D Interfaces . . . . . 17
2.4.1	2D vs 3D Document Management Systems . . . . . 18
2.4.2	2D vs 3D Hierarchical Data Structures . . . . . 20
2.5	Related Internet Visualizations . . . . . 20
<b>Chapter 3</b>	<b>GerbilSphere: Inner Sphere Graph Visualization Tool 24</b>
3.1	Graph File Formats . . . . . 24
3.2	Graph Memory Storage . . . . . 26
3.3	Layout Algorithm . . . . . 26
3.4	Spatial Indexing . . . . . 31
3.5	User Interaction . . . . . 32
3.5.1	Node Selection and Menus . . . . . 33
3.5.2	Interactive Zoom + Pan . . . . . 35
3.5.3	Labeling . . . . . 36
<b>Chapter 4</b>	<b>Experiments 38</b>
4.1	User Study Overview . . . . . 38
4.1.1	Single-Axis Topology Overview . . . . . 40
4.1.2	Multi-Axis Topology Overview . . . . . 43
4.2	User Study . . . . . 45
4.2.1	Subjects . . . . . 45
4.2.2	Equipment . . . . . 45

	v
4.2.3 Task . . . . .	45
4.2.4 Procedure . . . . .	46
4.3 Results . . . . .	48
<b>Chapter 5 Internet Visualization Implementation</b>	<b>51</b>
5.1 Data Source . . . . .	52
5.2 Internet Layout . . . . .	52
5.3 Display Environments . . . . .	52
5.4 Visual and Overall Assessment . . . . .	54
<b>Chapter 6 Conclusion and Future Work</b>	<b>56</b>
6.1 Conclusions . . . . .	56
6.2 Future Work . . . . .	57
<b>Bibliography</b>	<b>59</b>

## List of Tables

4.1	Singe-Axis Paired Samples Statistics . . . . .	48
4.2	Single-Axis Paired Samples Test . . . . .	49
4.3	Multi-Axis Paired Samples Statistics . . . . .	49
4.4	Multi-Axis Paired Samples Test . . . . .	50

# List of Figures

2.1	2D Tree. . . . .	5
2.2	Hyperbolic Browser Transitioning of Focus [43] . . . . .	6
2.3	3D Cone Tree. . . . .	7
2.4	Progression of a Force-Directed Layout. . . . .	8
2.5	Frutcherman and Reingold Grid Based Approach. . . . .	11
2.6	Non-Euclidean Force-Directed Layout . . . . .	14
2.7	3D vs 2D Spatial Memory Test . . . . .	19
2.8	Internet Backbones . . . . .	21
2.9	Minimum Spanning Tree of the Internet [13] . . . . .	22
2.10	3D Hyperbolic Space . . . . .	23
2.11	A Single ISP Router Network [23] . . . . .	23
3.1	GerbilSphere, Visualization on the Inside of a Sphere. . . . .	25
3.2	Nodes Moving to the Tangent Plane of Node $\mathbf{v}$ . . . . .	29
3.3	How Node $\mathbf{A}$ Travels from its Tangent Plane back to the Sphere at $\mathbf{C}$ . . . . .	30
3.4	Volume Grid Spatial Data Structure . . . . .	32
3.5	GerbilSphere's 2D Perspective of a Network . . . . .	33
3.6	Node Selection and Menus. . . . .	34

	viii
3.7 Zoom + Pan on a Tether. . . . .	35
3.8 Font Construction Consists of Primitives . . . . .	36
3.9 Node and Edge Labeling Types. . . . .	37
4.1 Single-axis Topology on a Spherical and Flat Surface. . . . .	39
4.2 Multi-axis Topology on a Spherical and Flat Surface. . . . .	39
4.3 Single-axis Start and End Positions on Flat and Sphere Surfaces . . .	41
4.4 Multi-axis Start and End Positions on Flat and Sphere Surfaces . . .	44
4.5 Topology of the Network Displayed During the One Minute Training.	47
5.1 Desktop Version. . . . .	53
5.2 Multi-User VR Version. . . . .	54
5.3 The AS Internet Sphere . . . . .	55

# Chapter 1

## Introduction

The rapid expansion of the Internet has made understanding its intricate connections difficult. With advancements in measurement techniques, the topology of the Internet can be constructed with more accuracy than before [29]. The topology of the Internet describes its connections and consists of a large data set that can be further understood through visualization. Over the last few years there has been quite a number of research works on the visualization of large data sets. Much of the work done in the area of Information Visualization has been with the application of interactive graphics and animation technology. By combining visualization techniques with user interaction, it has become practical to visualize larger information sets that otherwise would not be possible [58].

Visualizing the Internet allows you to explore network data visually and interactively and it helps with the detection of underlying patterns and structures of the network. Internet visualization also helps improve the topology, workload, performance, and routing configurations. The primary issue with visualizing the Internet is occlusion. The Internet consists of thousands of connected entities and showing them

all at once on a normal size screen would cause many objects to occupying the same space at the same time. Many techniques have been created to lessen the impact of occlusion. There have been several visualizations that show different elements of the Internet. Often these visualizations provide users with 3D maps to guide their browsing of the Internet. It was believed that these 3D interfaces were capable of displaying more information than a single 2D display, and that users would learn more about the Internet by flying through a 3D space [8]. Often new techniques to visualization improve the aesthetics without providing empirical user studies that show the usefulness of the techniques [15]. Further empirical studies on 2D vs 3D user interfaces have shown that 2D is often better. In this paper, we present GerbilSphere which is an Inner Sphere 2D system for the visualization of the Internet topology. This paper also contributes an empirical user study that tests the usefulness of the Inner Sphere method against a traditional flat method for network topology visualization and navigation.

The rest of the thesis is organized as follows: We start with background information on network visualization in Chapter 2 and give information about some of the popular layout algorithms used to visualize networks. We specifically look at force-directed layout algorithms with their strengths and drawbacks. We discuss the construction of the GerbilSphere visualization software, which uses the Inner Sphere method, and the reasons behind our algorithms in Chapter 3. We conduct a user study on how Inner Sphere topology navigation compares to flat layout topology navigation and present the results of our experiments in Chapter 4. We describe the Internet visualization using GerbilSphere and the improvements made based on the results from the user study in Chapter 5. Finally, we summarize our work and discuss future work in Chapter 6.

# Chapter 2

## Background

A graph is a network and can be defined as  $G = (V, E)$  where  $V$  is a set of vertices and  $E$  is the set of edges. In this paper, we will be looking at graphs of computer networks and the terms network and graph will be used synonymously. Since data can often be transformed into a graph, automatic layout techniques to display graphs are useful for visualizing data. Depending on the structure of the graph being visualized, different approaches can greatly improve the visual result of a layout. In this chapter, we will be looking at tree layouts as well as force-directed layouts. Both types of layouts strive to represent a network in a meaningful way. We will also look at various Internet visualizations that use these layout algorithms. Some of the desired results of layout algorithms include the even distribution of vertices, the minimization of edge crossings and uniform edge lengths. Other considerations are the time complexity of the layout and the memory requirements.

## 2.1 Tree Network Layouts

Tree layouts can be used to represent graphs that are hierarchical and binary in nature. Trees are a special case of graphs where you have no cycles in the graph. Tree layouts are very popular because they are intuitive to the user. They can be created quickly and they are easily implemented. In addition to a fast creation and easy implementation, converting relational graphs into trees can reduce the amount of edges and results in less occlusion in large graphs. Every graph can be converted into a tree structure by calculating a spanning tree of the graph. Computer networks, such as the Internet, do not form a tree but can be visualized as a tree when a spanning tree is derived from it. This strategy of creating a tree to view the Internet topology was used by Munzner [49]. Trees can be visualized in 2D or 3D.

### 2.1.1 2D Trees

Tree layouts can look very nice and easily portray the structure of a network (Figure 2.1). There are usually certain aesthetic characteristics that are sought when creating a 2D tree layout. There are three aesthetic properties that have been labeled as requirements for a beautiful tree [70]. First, the nodes that occupy the same level of the tree should be drawn in a straight line, with all drawn levels parallel to each other. Second, a left child should be positioned on the left side of its parent and a right child on the right. Third, the parent node should be centered over its children nodes. The value of tree layout algorithms is based on their fulfillment of the aesthetic points as well as their computational efficiency. Tree layout algorithms have been studied over several decades. Early algorithms by Wetherell and Shannon [70] successfully laid out trees fulfilling various aesthetic requirements. Reingold and Tilford [56] improved on

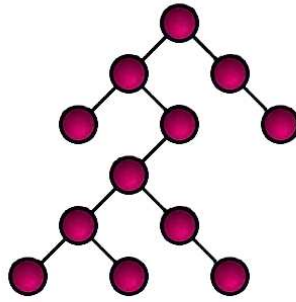


Figure 2.1: 2D Tree.

the algorithm by making the tree narrower and still aesthetically pleasing. This was done with a post-order traversal to visit all the nodes and position the subtree in the proper space. Further development on the rooted tree algorithm was done by Walker [67] and the running time of his algorithm was improved by Buchheim *et al.* [11].

Hyperbolic trees present a distorted view of a tree in such a manner as to bring greater detail to a node of interest and less detail to the surrounding nodes. This technique can be considered a focus+context strategy and is discussed in Section 2.3.2. Early examples of this technique can be found in the work by Lamping *et al.* in their Hyperbolic Browser [43, 44]. The Hyperbolic Browser seen in Figure 2.2 shows the focused node in the center moving down as a new node transitions into focus. Although tree algorithms are fast and generate aesthetic layouts, converting general graphs into tree graphs is not always desired as it may cause valuable edges to be lost. Moreover, many structural characteristics such as clustering of general graphs are left out with the conversion.

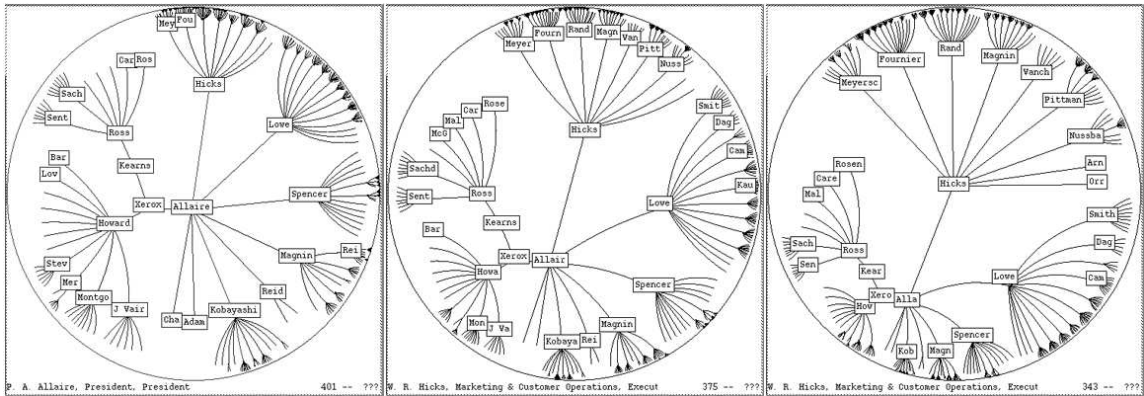


Figure 2.2: Hyperbolic Browser Transitioning of Focus [43]

### 2.1.2 3D Cone Tree Layouts

The Cone Tree is a visualization technique which is used for the visualizing of hierarchical information as opposed to arbitrary graphs. It requires interactive animation of its branches in order for the user to select branches that are occluded. Robertson *et al.* [61] indicated that the Cone Tree maximized the available screen space and gave an overall view of the hierarchical structure. The creation of a Cone Tree involves placing a node at the top level of the hierarchy on the ceiling of the 3D room. The next level of nodes in the hierarchy is drawn in a circle below the top node with the plane of the circle parallel to the ground. Each successive level is then drawn below its parent node, and the end result is a structure that is a Cone Tree (Figure 2.3). Navigation to the occluded nodes is accomplished by rotating a cone until the desired node comes into view. The rotation is animated in hopes of allowing the user to maintain the relationships between the structures. Carriere and Kazman [12] noted that the traditional Cone Tree could only handle roughly 1,000 nodes before the visual clutter hindered its usefulness. They improved the Cone Tree by adding additional graphical and interaction techniques and were able to view 5,000 nodes.

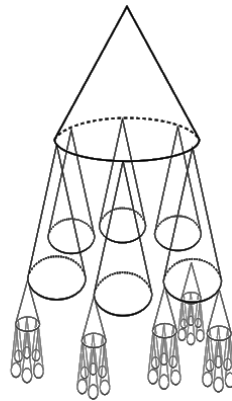


Figure 2.3: 3D Cone Tree.

## 2.2 Force-Directed Network Layouts

Most graphs do not contain explicit information on the best way in which to lay them out. Through the use of forces, a graph can be laid out in an aesthetically pleasing manner based on its structure alone. The basic idea with force-directed approaches is to apply an attractive force between nodes that are considered connected and repulsive forces between all nodes. The application of these forces must be repeated several times in order for a graph to arrive at an aesthetically pleasing state. The value of force-directed layouts were claimed to be of great worth before a formal study investigating their worth was done. Purchase *et al.* [54] conducted a study to determine which attributes were most important for graph visualization. It was found that symmetry, minimal crossings and minimal bends in edges increases user response times when interacting with a graph. The use of force-directed layouts can result in a layout with symmetry and minimal crossing edges. We present two models of applying forces, namely *Spring Force* and *Graph Theoretic*.

### 2.2.1 Spring Force Model

In order to determine forces, Eades [18] converted graphs to a system of springs. Each edge in the graph could analogously be replaced with a spring and each vertex with a steel ring. By constructing the graph with connected steel rings and springs, the system of springs could be released and allowed to move into a more stable state, hopefully arriving at a more pleasing graph layout. This algorithm begins with an initial arbitrary positioning of the steel rings as shown in Figure 2.4a. Next, it completes a series of discrete time steps and computes the attractive and repulsion forces between the vertices and updates each vertex position (Figure 2.4b). Several iterations of the algorithm are required to end up with a layout that is visually appealing (Figure 2.4c). The required time complexity for the node repulsion is  $\Theta(|V|^2)$  since

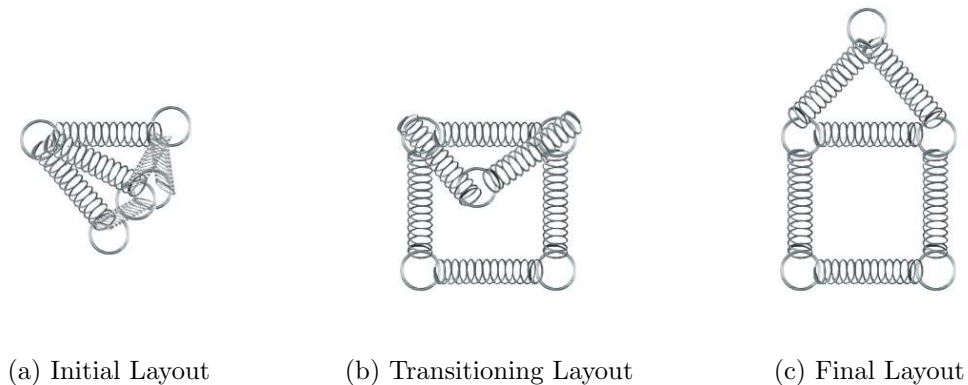


Figure 2.4: Progression of a Force-Directed Layout.

each node is compared against every other node. The required time complexity for the edge attraction is  $\Theta(|E|)$  since each edge represents two neighboring nodes.

Eades used attractive forces between neighboring vertices connected by an edge, but his equations did not follow the normal spring calculations based on Hook's law. Instead, Eades used his own formulas for the force calculations. Eades's algo-

rithm also stopped after a pre-determined amount of iterations. The algorithm could stop before completing enough iterations to produce a pleasing layout. It also could stop after separately connected graph components moved off the visible screen area. On the other hand, the algorithm could reach a stable state long before the iterative steps complete and thus waste time. Even with all its short comings, for small graphs the n-body algorithm of Eades does improve graph readability.

Frutcherman and Reingold [24] set out to improve Eades's algorithm. Similarly, they did not use force equations that are found in the physical world, but their own. They also applied attractive forces between neighboring vertices and repelling forces to all vertices. They also point out that the term *forces* is a misnomer since they use forces to calculate velocity for every time quantum. A more correct term would be *impulse vector*. In Algorithm 1, we present a pseudocode for their grid based algorithm that we implemented. The major improvements with their method was the idea of a global graph temperature (*Updates* section of Algorithm 1) that cooled down as time progressed. Without a graph temperature vertices would move a constant distance at each time quantum. With the temperature applied, the time quantum displacement distance decreases as a function of the current global temperature of the graph. This allowed the displacement to shrink as a function of time, which causes a refinement of motion. Another improvement introduced, was to cage the vertices inside a frame which prevented them from leaving the screen as the forces were applied (*Updates* section of Algorithm 1). A third improvement was to break the frame up into a grid as in Figure 2.5a and apply the repulsion forces (*Repulsion* in Algorithm 1) only to other nodes in the same and surrounding grids as in Figure 2.5b. All nodes in the surrounding grids are then checked to see if they meet a distance requirement (Figure 2.5c). Those nodes that meet the requirement have a repelling effect on the current node.

**Input:** Set of vertices with random positions and the set of edges

**Output:** Set of vertices with new positions and the set of edges

$$k = \sqrt{(\text{frameWidth} * \text{frameHeight})/|V|}$$

**function**  $f_a(x) = \text{begin return } x^2 / k \text{ end;}$

**function**  $f_r(x) = \text{begin return } k^2 / x \text{ end;}$

**//Each vertex has two vectors: .pos and .disp**

**while**  $temperature > 0$  **do**

Repulsions

**foreach**  $Vertex v \in V$  **do**

$tDisp = 0;$

$nearGridsList = \text{getNearGridsList}(v.myGridNum);$

$nearVertsList = \text{getNearVertsListFromGrids}(nearGridsList);$

**foreach**  $Vertex u \in nearVertsList$  **do**

$vDisp = v.pos - u.pos;$

$tDisp = tDisp + vDisp.norm() * f_r(vDisp.len());$

**end**

$v.disp = v.disp + tDisp;$

**end**

**//Edges have reference vectors fNode and tNode  $\in V$**

Attractions

**foreach**  $Edge e \in E$  **do**

$diff = e.fNode.pos - e.tNode.pos;$

$e.fNode.disp = e.fNode.disp - diff.norm() * f_a(diff.len());$

$e.tNode.disp = e.tNode.disp + diff.norm() * f_a(diff.len());$

**end**

Updates

**foreach**  $Vertex v \in V$  **do**

$v.pos = v.pos + v.disp.norm() * \min(v.disp, temperature);$

$v.pos.x = \min(W/2, \max(-W/2, v.pos.x));$

$v.pos.y = \min(L/2, \max(-L/2, v.pos.y));$

$v.disp = 0 ;$

**end**

$temperature = \text{cool}(temperature);$

**end**

**Algorithm 1:** The Frutcherman and Reingold Grid Based algorithm

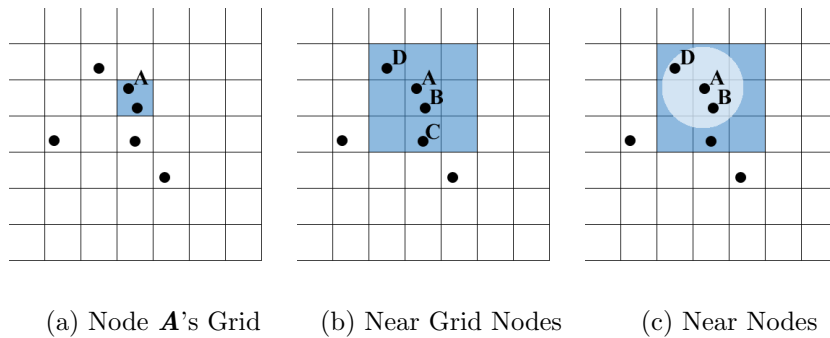


Figure 2.5: Frutcherman and Reingold Grid Based Approach.

The reasoning behind this heuristic is that the force effects that distant nodes have on each other decreases significantly as the distance increases. The complexity of calculating the repulsive forces becomes  $\Theta(|V|)$ , when the vertices are distributed evenly. This changes the original complexity from  $\Theta(|V^2| + |E|)$  to  $\Theta(|V| + |E|)$ . Another benefit to the use of a grid is that separately connected components won't fly apart and smash themselves against the sides of the frame. Both Eades and Frutcherman and Reingold mentioned use of graphs with a vertex size smaller than 100.

Frick *et al.* [22] further improved force-directed layout algorithms. Instead of sequentially selecting a vertex upon which to apply an impulse vector, they randomly select a vertex. This randomness helps the graph to converge faster. Other improvements include the memorization of the last movement (i.e., last impulse of the vertex) of each vertex. This memorization helped reduce fluctuations in movement. Also, a local temperature can be calculated for each node, and the global temperature is the sum of all local temperatures. Further improvements have been done by transforming force-directed layout algorithms to run in parallel. These parallel algorithms have been used over distributed resources [47, 65] and on graphic processing units [23, 28].

## 2.2.2 Graph Theoretic Force Model

In the previously mentioned algorithms, the attraction and repulsion forces are determined by whether a node is connected to another and by the actual physical distance of the vertices. Instead of only using the distance between vertices to calculate the repelling force, Kamada and Kawai [40] propose a graph theoretic distance between the vertices. The graph theoretic distance tells both if the vertices are neighbors and how far apart they are in terms of edges. Kamada and Kawai put a new twist on the way these forces should be derived and explain in their own words:

“We regard, the desirable ‘geometric’ (Euclidean) distance between two vertices in the drawing as the ‘graph theoretic’ distance between them in the corresponding graph.”

This algorithm requires that the graph theoretic distance be computed for all the pairs of vertices. For this computation they used the shortest path algorithm by Floyd [21]. They assumed that the given graph is connected. First, graphs with several connected components are decomposed into connected components, which takes  $\Theta(|V| + |E|)$ . After decomposition, the algorithm is run separately on each connected component, which takes  $\Theta(n^3)$  time. The goal of their algorithm is to reduce the total energy of the system since vertices repel and attract each other. By solving partial differential equations for each vertex, a new location is found for that vertex which reduces the overall energy of the graph. After the energy falls below a set threshold, the repositioning of vertices stops.

### 2.2.3 Multilevel Decomposition Model

Force-directed algorithms are able to handle even larger graphs with multi-scale techniques. Hadany and Harel [30] were the first to use this method for larger graphs and then several other multi-level algorithms were developed [27,31,55,68]. The multilevel model idea operates by grouping vertices into clusters and then using those clusters to form a new graph which is considered a course-scale version. Fine-scale relocations occur at each level considering vertices within each cluster. The operation is repeated with the new graph and continues producing new graphs until a desired threshold is met [68].

### 2.2.4 3D and Non-Euclidean Force-Directed Layouts

Force-directed methods easily transfer to 3D space. The transfer is accomplished through simply calculating the distance formula for 3D points as opposed to 2D points. The drawing of arbitrary graphs in 3D was mentioned by Frutcherman and Reingold [24] and enhancements were done by Brus and Frick [10]. In addition to 3D space, force-directed layout methods can be applied to arbitrary Riemannian geometry. In Euclidean space, the idea of angles and distances is easily computed for the force-directed layouts but non-Euclidean space is a little trickier. Kobourov and Wampler [41] describe algorithms that extend force-directed layouts to hyperbolic and spherical geometries using a tangent space mapping technique. The basic idea behind the mapping is that each point on the sphere uses its tangent plane as in Figure 2.6a. When calculating how a node should interact with other nodes, every other node is embedded onto the tangent plane and the calculations are performed on the plane. The node is then embedded back onto the sphere and its new location on the sphere

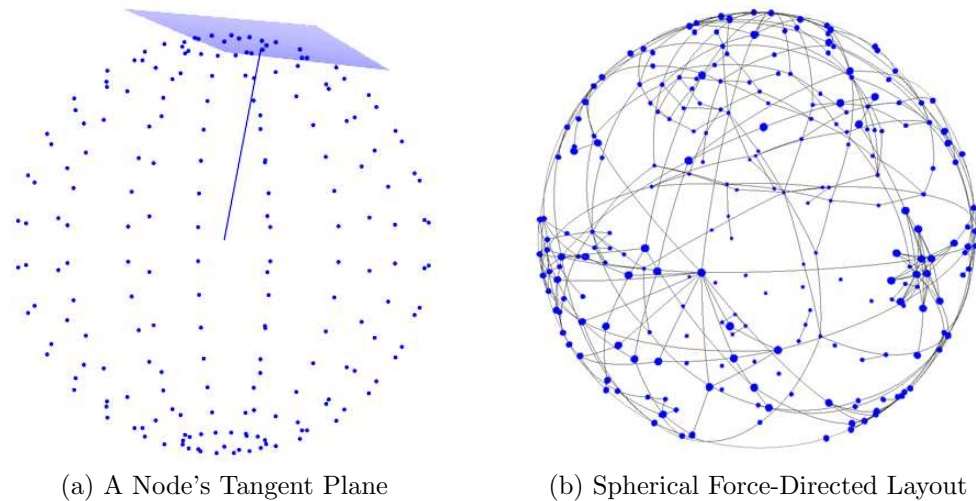


Figure 2.6: Non-Euclidean Force-Directed Layout

reflects its movement on the tangent plane. These steps are repeated for all nodes until the final result is an aesthetically pleasing layout on a sphere (Figure 2.6b). Our layout algorithm, discussed in Chapter 3, is based on the tangent plane method of Kobourov and Wampler.

## 2.3 Interactive Networks

As the size of a network increases, a user's ability to navigate through the data becomes more important. Parts of the network that appear occluded may become visible again as the user navigates their way around. Important aspects of user navigation include the ability to zoom into an area for increased detail and zoom out to decrease the detail but maintain the context of the whole graph. Often with 3D graphs, user interaction allows the navigation of the third dimension.

### **2.3.1 Zoom + Pan**

Zoom + Pan are among the most used interaction techniques employed when visualizing networks. Both of these mechanisms allow the user to see more than would normally be possible. As the amount of information per screen unit increases, the need for zooming and panning increases. Due to the limit of the number of pixels on the screen, huge network graphs cannot be shown because the number of nodes might exceed the number of pixels. Allowing the user to pan essentially simulates having a larger display. The pixels that once were filled with information now have passed that information to the adjacent pixels. The screen border pixels are not able to pass the information so that part of the image is temporarily not visible but this allows another part of the image to become visible. The issue of temporarily losing sight of part of the image also occurs when zooming into a region of the graph. What is gained during a zoom is increased detail of the chosen area but this comes at the expense of losing the context of the whole image. The context of the image can be maintained to a degree by using focus + context techniques.

### **2.3.2 Focus + Context**

Focus + context is a technique that is used to overcome the default behavior of a normal zoom. As a zoom occurs, the focus of the zoom becomes larger and larger and eventually occupies the whole screen. The surrounding image becomes farther and farther away and eventually leaves the screen. In order to keep the context of the overall graph, compression techniques are employed which allow the focused point to enlarge; the surrounding area stays on the screen but compresses into a reduced space. This compression does remove some detail but the overall context is still maintained.

Several focus + context approaches for information visualization, based on fish-eye viewing, have been developed over the past decade. With fish-eye viewing, a balance is sought between viewing a point of interest in detail and keeping the global view at the same time [25].

Other research has been conducted to enhance visualization using a technique called the hyperbolic browser. The hyperbolic browser is used for visualizing and manipulating large hierarchies and uses hyperbolic geometry to accomplish its visualization (see Figure 2.2). The hyperbolic browser has two main properties: first, nodes reduce in size as they distance themselves from the center of the circle, and second, there occurs an exponential growth in the number of components with increasing radius. What makes the hyperbolic browser useful is its “fisheye” distortion and the ability to uniformly embed an exponentially growing structure.

Lamping *et al.* [43] found that the hyperbolic browser supports effective interaction with much larger hierarchies than conventional hierarchy viewers. The hyperbolic browser approach helps keep the context of a hierarchy while allowing the exploration of it. One major drawback to using this technique is that it can’t handle networks with thousands of nodes.

### **2.3.3 Node and Edge Selection**

Node selection is important. The ability to query a node for more information is of great value because extra information may be hidden until the user requests it. This strategy can save huge amounts of screen space. Selecting a node can also allow the user to perform specific tasks on a single entity of the graph being displayed. Edge selection can allow the user to understand the type of relationship between two nodes in the graph.

### 2.3.4 Multi-level Visualization

The ability to zoom into a region of interest allows the user to explore the graph in greater detail. Zooming can compensate for the limited screen space available on displays. A multi-level zoom is a technique by which there are varying levels of detail that can be reached through the zooming mechanism. Each level provides a greater detail than the previous level. A smooth transition between levels is important to help the user maintain a context of where they have come from.

## 2.4 2D vs 3D Interfaces

Early graphical interaction was strictly in 2D. With the improvement in 3D graphic methods, it became possible to interact in real-time with 3D rendered objects. The question is whether or not one should be designing user interfaces in 3D. Current research shows that it all depends on the application and desired task of the user but 2D user tasks are often faster. Some research has compared 2D images on monoscopic displays to 3D images on monoscopic displays or 2D monoscopic displays to 3D stereoscopic displays [62]. Often the results show that user interaction is faster with a 2D version than a 3D version. What is it about a user interface that makes it useful? Many studies have shown that measured performance with user interfaces is higher in people who have higher spatial memory abilities [19, 26, 45, 66]. These findings have been shown with several different types of user interfaces including text editors [19] and computer games [26]. Gagnon [26], in his experiment with video games, showed that higher scores correlated to a subject's spatial memory test and not to their hand eye coordination. These findings suggest that improvements in a user interface can be made through facilitating a user's spatial memory. Early studies on how 2D and

3D interfaces affect spatial memory were in favor of 3D but follow up studies showed 2D and 3D to be the same.

In addition to spatial memory tests, many tests of usefulness have been done with 2D and 3D user interface models. Surprisingly 2D is often more effective than 3D. Sometimes the combination of the two can increase the effectiveness of a tool as shown by Risend *et al.* [57] in their XML3D tool which uses a 3D hyperbolic graph view as well as a traditional 2D list view of data. Ware [69] indicated that the careful combination of 2D and 3D would also enhance a user interface. He advocated that designing should be done mostly in 2D with a 2 1/2 D attitude. It is sometimes the case that new ways to visualize data are generally accepted as a better method based on the visual result alone without performing a sound empirical user study to provide evidence. In the following subsections we review several empirical user studies that have shown that presenting information in 2D is as effective as or better than using 3D.

#### **2.4.1 2D vs 3D Document Management Systems**

It was believed that a 3D perspective offered more than 2D in terms of visual usefulness. Many 2D visualization methods have been transformed into 3D visualization methods [59, 60]. Robertson *et al.* [59] showed that retrieving web pages using their 3D Data Mountain produced lower task times and error rates than the traditional 2D method of Internet Explorer. Cockburn and Mckenzie [16] later proved this was not due to the 3D factor.

Tavanti and Lind [64] did experiments on spatial memory with tasks that involved the memorization of data arranged in a hierarchy. The first hierarchy was in a vertical format like a file system similar to Windows Explorer. The second hierarchy

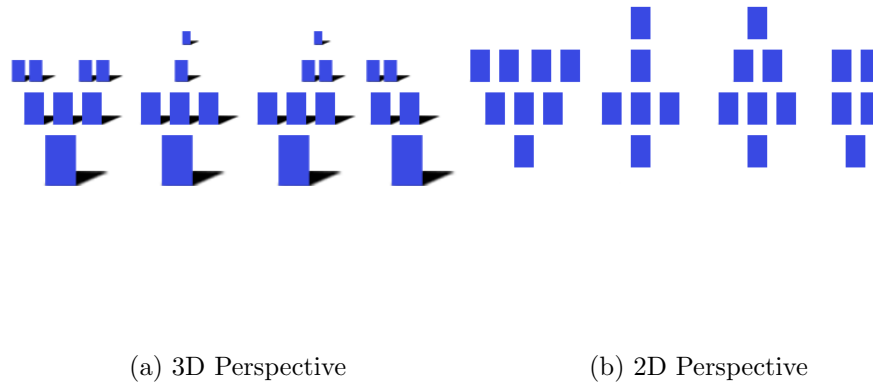


Figure 2.7: 3D vs 2D Spatial Memory Test

was a front to back system with a perspective as seen in Figure 2.7a. Users had to memorize, and recall at a later time, a symbol that was behind each rectangle. After performing these memorization experiments they concluded that 3D helps specific spatial memory tasks. Their results were in favor of 3D, but there were several uncontrolled factors in their experiments. Cockburn [14] set out to redo the experiments of Travanti and Lind and control for the uncontrolled factors in their location recollection experiment. Cockburn compared the scores of memory tasks on the same layout but one had a perspective effect, as seen in Figure 2.7a, and the other had a flat effect (Figure 2.7b). He found that there was no significant difference between the 2D and 3D perspective effects with regard to spatial memory of a static scene presented on a monocular display. Even though there was no significant advantage, participants favored the 3D view over the 2D view.

### 2.4.2 2D vs 3D Hierarchical Data Structures

Cone Trees provide a 3D visualization of hierarchical data. They allow the user to rotate the cones and bring data that was in the back to the front (see Figure 2.3). It was believed that this method to visualization maximized the user's perceptual system and aided in the navigation of the data. Several systems were created that used cone trees in hopes they better exploited the human perceptual system [32,33]. Cockburn and McKenzie [15] compared user navigation of a Cone Tree to user navigation of a normal tree browser that was similar to the file management browser in Windows Explorer. Both systems of visualization were used to explore the same hierarchical data. After their tests were complete they found that when subjects used the normal-tree interface they could follow paths more quickly than when they used the Cone Tree interface. The paper did not factor out the 3D element, it did compare two different types of interfaces on the same task. Even though the times were slower, subjects commented that the Cone Tree interface provided a better sense of the overall structure of information.

## 2.5 Related Internet Visualizations

There have been several visualizations of the Internet. Everything from the multi-cast backbone (Figure 2.8a) to the actual backbone of the Internet (Figure 2.8b). Cheswick *et al.* [13] used traceroute collected information to create a static visualization of the Internet (Figure 2.9). This visualization was simplified using a minimum spanning tree to reduce the amount of edges that cluttered the middle. Munzner [50] also used a minimum spanning tree to visualize the Internet in her H3 hyperbolic browser (Figure 2.10a). This hyperbolic browser does rendering on the inside of

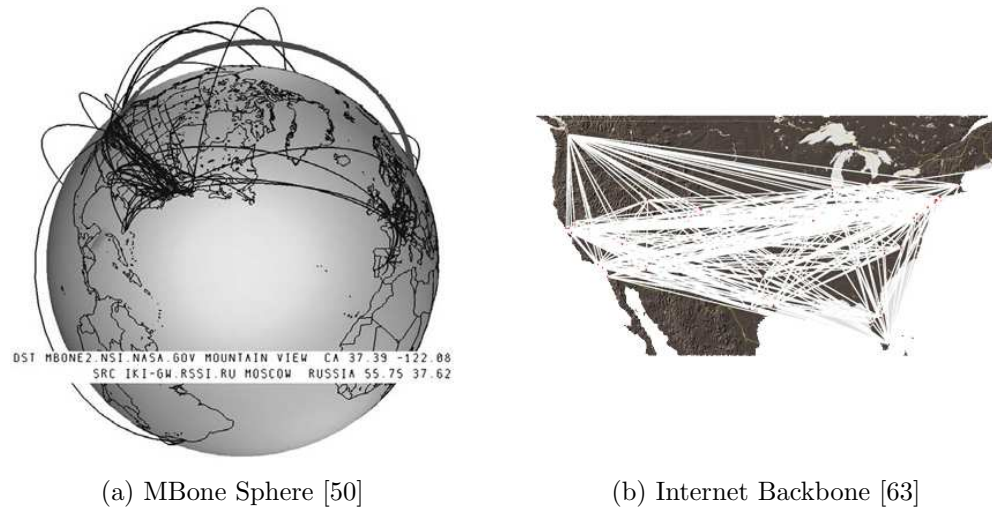


Figure 2.8: Internet Backbones

a sphere. The H3 algorithm is also used in the Walrus [36] visualization tool that has been used for Internet visualization as well (Figure 2.10b). Frishman [23] used a graphics processing unit to visualize an Internet Service Provider network (Figure 2.11). All of these visualizations are impressive, but they do not allow interactive exploration of nested networks. Furthermore, the H3 hyperbolic browser based tools only work with directed graphs and connected graphs with reachable nodes.

Several other tools have been developed to visualize Internet topologies including BGPlay [17], Cylpos [52], DIMES Visualizer [4], InterMapper [1], IPsonar MapViewer [2], LaNet-vi [3], LinkRank [42], Nam [20], Network Weathermap [39], Opte Project [46], Osprey [9], Polyphemus and Hermes [6], Tulip [5], VAST [51].

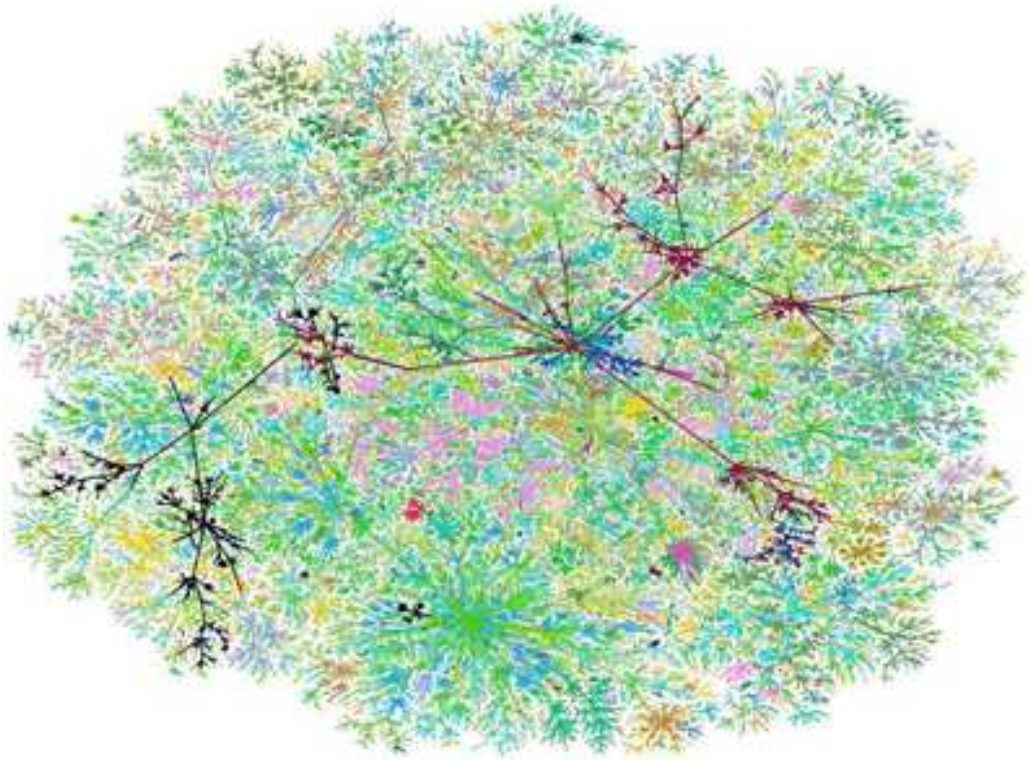


Figure 2.9: Minimum Spanning Tree of the Internet [13]

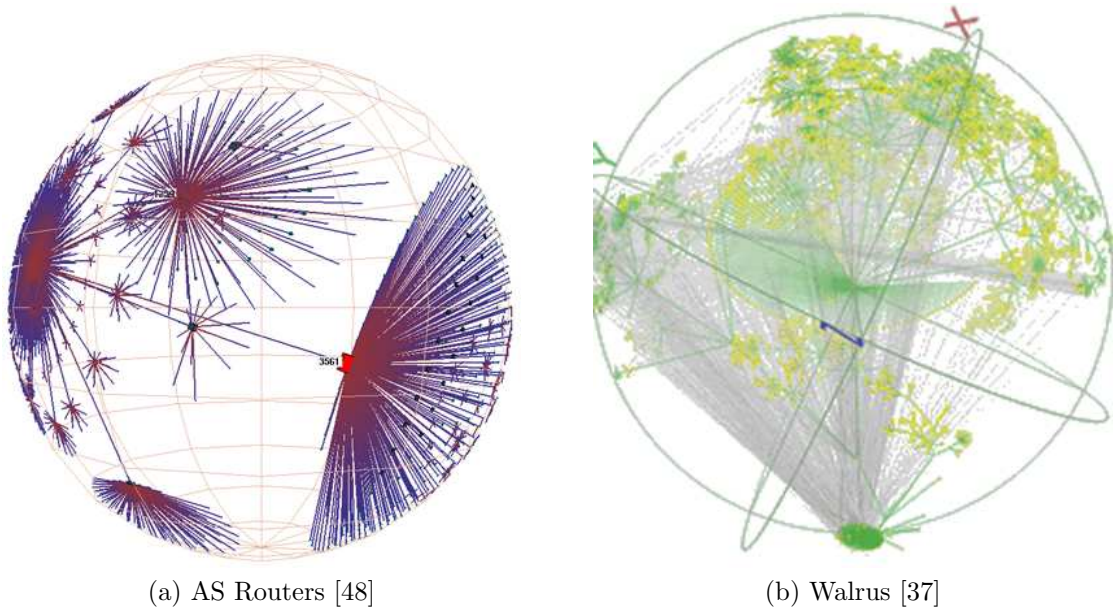


Figure 2.10: 3D Hyperbolic Space

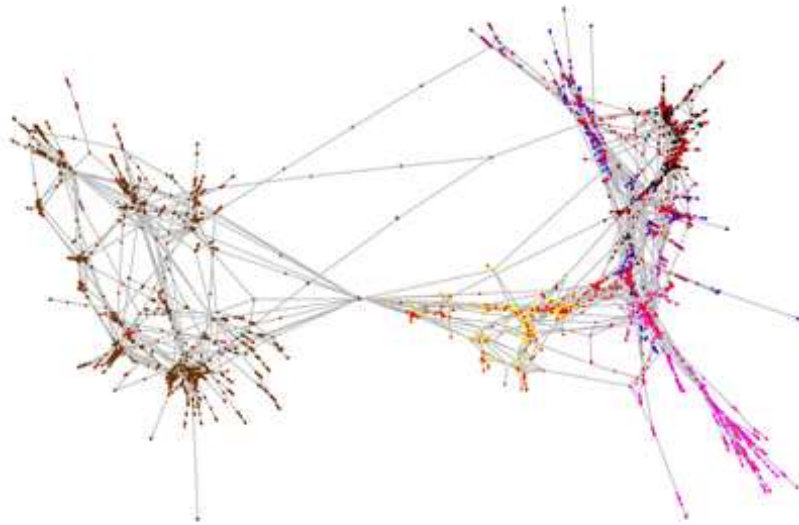


Figure 2.11: A Single ISP Router Network [23]

## Chapter 3

# GerbilSphere: Inner Sphere Graph Visualization Tool

Our method for network visualization differs from earlier approaches in that we place our observer of the network on the inside of a sphere and project the network around them on the sphere. This method can be likened to a patron inside a planetarium, watching the stars surround them. Navigation throughout the network can be likened to a gerbil inside a transparent gerbil ball. As the gerbil moves, its orientation remains fixed and the ball revolves around it (Figure 3.1).

### 3.1 Graph File Formats

Currently, GerbilSphere supports the Pajek [7] and the Extensible Graph Markup and Modeling Language (XGMML) [53] file formats. The Pajek format is a simple text-based format in which the graph is described with a list of nodes followed by a list of edges. The XGMML is based on the Graph Modeling Language (GML) [34]

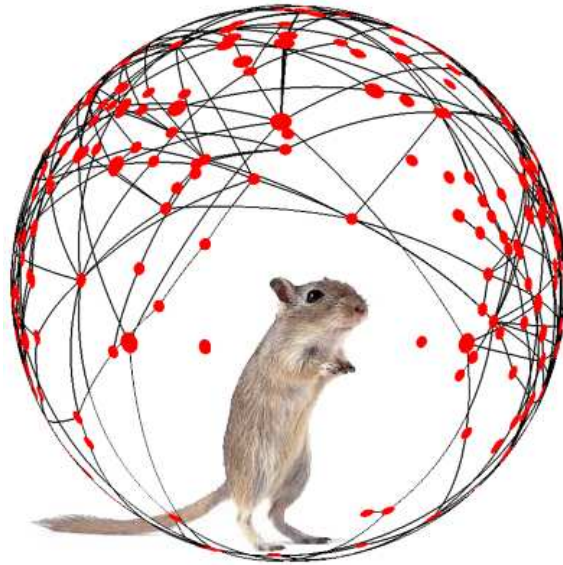


Figure 3.1: GerbilSphere, Visualization on the Inside of a Sphere.

and all the tags used in GML are also available in XGMML. The main difference is that XGMML is an XML 1.0 application language satisfying a list of syntax rules. The file content must adhere to a *Document Type Definition(DTD)* or a *Schema*. The root element of a XGMML file is a *graph* and the *graph* can contain *node*, *edge* and *att* elements. This language is extensible by attaching more information to a *graph*, *node* or *edge* using the *att* element. A subgraph can be created by placing a *graph* element inside of an *att* element. Each element only allows specific other elements inside. For example, the *graph* element only allows *node*, *edge* and *att* elements inside. The *node* element only allows *graphics* and *att* elements. Any additional meta information for a node can be contained inside the *att* element of that node. Like the *node*, the *graphics* and *att* elements are the only allowed elements inside an *edge* element. Through the use of these elements, whole graphs can easily be represented.

## 3.2 Graph Memory Storage

Two representations of a loaded graph are created with one residing in system memory and the other on the graphics card. Once GerbilSphere loads a graph file into memory, the graph is stored as a list of nodes and a list of edges in system memory with every edge pointing to two nodes and every node containing a list of its neighbors. Every node keeps track of what 3D grid spatial data structure it belongs to, as well as its index in that structure's list. This allows for constant time transferring between grids. Once the graph layout becomes *baked* (i.e., finished), the location of the nodes and edges is copied onto the graphics card. This single transfer to the graphics card is considered retained mode rendering and it greatly increases the rendering speed of visualization since CPU-to-GPU communications are slow.

## 3.3 Layout Algorithm

Our program utilizes several layout algorithms that place the nodes and edges on the surface of a sphere. Once a layout is computed, the results can be saved to a file for faster loading in the future. If a graph is opened and does not have any coordinates associated with its nodes, a simple sphere layout is used to give each node an initial starting position. Each node's position on the sphere is then used to calculate which 3D volume grid it belongs to. Once an initial starting position is given, the graph is considered *half-baked*, meaning it needs more adjusting to arrive at a nicer layout. The second algorithm used is the grid based force-directed algorithm of Frutcherman and Reingold (Algorithm 1) which we extended to 3D space using a 3D volume grid. Even in 3D space this algorithm does not work on the surface of a sphere without modification. In order to apply the forces on the sphere, we use a third algorithm

described by Kobourov and Wampler [41] to perform a spherical force-directed layout. To our knowledge, the combination of these algorithms, see Algorithm 2, has never been done before. We modify Kobourov and Wampler’s algorithm to function with the 3D grid based system which improves the running time. The 3D grid modification requires our *getNearGridsList()* function to find at most 18 neighboring volume grids. These grids are found in constant time due to the nature of our spatial data structure which is described in Section 3.4.

Through modifying several parts of a generic force-directed algorithm, Kobourov and Wampler were able to determine how the forces applied to a sphere. As the repulsive and attractive forces are calculated, a new function maps points from the sphere to the tangent plane of a node as seen by the *fromSphereToTangentPlane()* function in the *Repulsion* section of Algorithm 2. Unlike Kobourov and Wampler, we keep the coordinate system strictly in  $R^3$ , instead of switching back and forth between polar and  $R^3$ . Our *fromSphereToTangentPlane()* function gets the same result as Kobourov and Wampler but uses different calculations. We first find the directional vector that points from the center of the sphere to the current node  $v$  (Figure 3.2a) and next apply various scales of that vector to map all other nodes to node  $v$ ’s tangent plane (Figure 3.2b). Both the *Repulsion* section and the *Attraction* section of Algorithm 2 use the *fromSphereToTangentPlane()* function.

**Input:** Set of vertices with random positions and the set of edges

**Output:** Set of vertices with new positions and the set of edges

$$k = \sqrt{(\text{frameWidth} * \text{frameHeight})/|V|}$$

**function**  $f_a(x) = \text{begin return } x^2 / k \text{ end;}$

**function**  $f_r(x) = \text{begin return } k^2 / x \text{ end;}$

**//Each vertex has two vectors: .pos and .disp**

**while**  $\text{temperature} > 0$  **do**

Repulsions

**foreach**  $\text{Vertex } v \in V$  **do**

$tDisp = 0;$

$nearGridsList = \text{getNearGridsList}(v.myGridNum);$

$nearVertsList = \text{getNearVertsListFromGrids}(nearGridsList);$

**foreach**  $\text{Vertex } u \in nearVertsList$  **do**

$vDisp = v.pos - \text{fromSphereToTangentPlane}(u.pos);$

$tDisp = tDisp + vDisp.norm() * f_r(vDisp.len());$

**end**

$v.disp = v.disp + tDisp;$

**end**

**//Edges have reference vectors fNode and tNode  $\in V$**

Attractions

**foreach**  $\text{Edge } e \in E$  **do**

$diff = e.fNode.pos - \text{fromSphereToTangentPlane}(e.tNode.pos);$

$e.fNode.disp = e.fNode.disp - diff.norm() * f_a(diff.len());$

$diff = e.tNode.pos - \text{fromSphereToTangentPlane}(e.fNode.pos);$

$e.tNode.disp = e.tNode.disp - diff.norm() * f_a(diff.len());$

**end**

Updates

**foreach**  $\text{Vertex } v \in V$  **do**

$v.pos = v.pos + v.disp.norm() * \min(v.disp, \text{temperature});$

$v.pos = \text{fromTangentPlaneToSphere}(v.pos);$

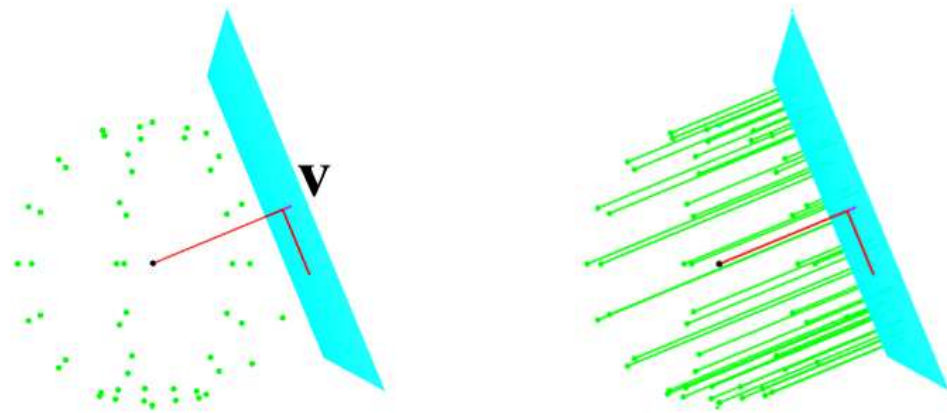
$v.disp = 0 ;$

**end**

$\text{temperature} = \text{cool}(\text{temperature});$

**end**

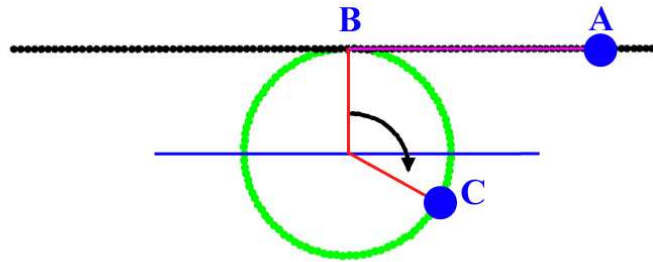
**Algorithm 2:** Spherical Volume Grid Based algorithm

(a) Node  $v$ 's Tangent Plane

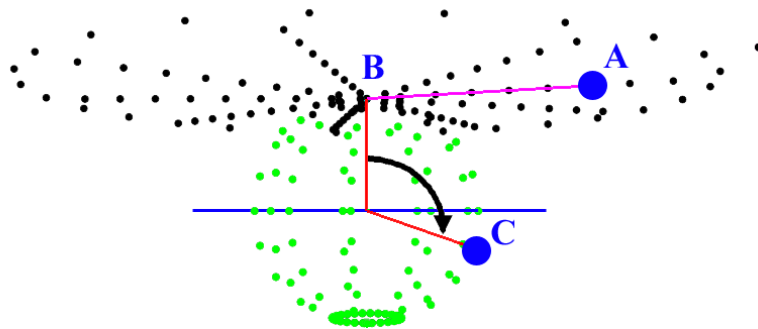
(b) All Nodes on Plane

Figure 3.2: Nodes Moving to the Tangent Plane of Node  $v$ 

The final modification of the generic force-directed algorithm is to the update portion. In the update, each node's displacement vector is defined in terms of its own tangent plane. After adding the displacement vector to the current node, the node is then transferred back onto the sphere maintaining the same angle and distance as its displacement was on its own tangent plane. This is accomplished with the *fromTangentPlaneToSphere()* function as seen in the *Updates* section of Algorithm 2. The final process of moving a node from its tangent plane back to the sphere is illustrated in Figure 3.3. In Figure 3.3, Node **A** is a node on its tangent plane after adding its displacement to itself. The label **B** indicates node **A**'s original position. Next, the distance and angle between **B** and **A** is calculated. Finally, node **A** is moved to position **B** and rotated using the angle and distance between **B** and **A** to end up back on the sphere at point **C**.



(a) Front View



(b) Side View

Figure 3.3: How Node A Travels from its Tangent Plane back to the Sphere at C.

### 3.4 Spatial Indexing

There are several types of spatial data structures that can be used to improve the running time of algorithms that operate on objects in space. These structures subdivide the space into regions and depending on the algorithm, the regions are uniform or non-uniform. We can place the nodes of a graph into regions and save time by only checking the regions that house nodes when performing calculations.

There are several spatial data structures that could be used. The quadtree is based on the divide-and-conquer principle and recursively divides a square into 4 squares. The subdivisions usually occur around points whose location is static and the subdividing of a region stops when a minimum number of nodes in each area is reached. If the points change location then the quadtree must change. An octree is based on the same idea as the quadtree but divides a 3D cube into 8 cubes. At first the octree seemed like the best fit for a spatial data structure for network visualization. However, we did not use the octree for several reasons. First, with the force-directed layout the nodes are constantly changing position in graphs that are not yet baked. Each time a node moves it may require the insertion or deletion of nodes in the octree. Due to the high volume of nodes and the even distribution of nodes around the sphere, the octree would have similar grid sizes as the volume grid. Moreover, queries into the octree would not be as fast as a simple volume grid structure. With these considerations in mind, GerbilSphere uses a simple axis-aligned 3D grid for a spatial indexing data structure as seen in Figure 3.4. The grid can be created in  $\Theta(n)$  where  $n$  is the number of grids. This is faster than other tree algorithms which often have a creation complexity of  $\Theta(n^2)$ . The grid is created by uniform partitions and allows nodes to easily move around without having to recalculate the spatial data

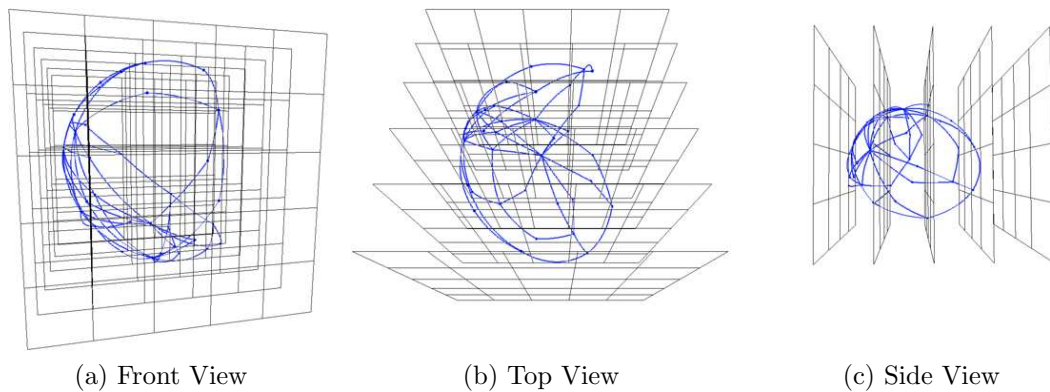


Figure 3.4: Volume Grid Spatial Data Structure

structure. The grid also allows for direct access to neighbor grids when calculating nodes that may be physically near, but not in the same grid. The same static 3D grid volume can be used for several graphs, with each graph having an index into the 3D volume. We can use this for the current graph we are viewing as well as the possible graphs that we zoom into. Even though certain situations may cause some of the 3D volumes to be empty, the required space is very small. Furthermore, the advantages of loading several graphs into the same spatial data structure outweigh the small loss of space. The axis-aligned 3D grid also allows for easy picking of nodes.

### 3.5 User Interaction

In order to improve user interaction of a visualized graph, GerbilSphere takes into account the empirical research done on 2D and 3D perspective found in Chapter 2.4. Since the user’s perspective comes from the inside of the sphere, as seen in Figure 3.5a, the visualization of the network will appear to be on a flat 2D surface when zoomed in as seen in Figure 3.5b. When the user zooms out and widens their lens, our

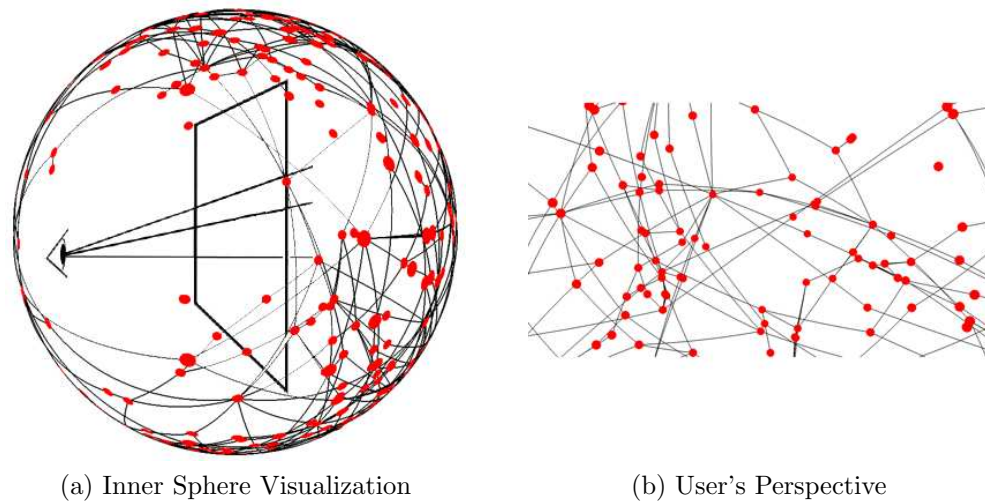


Figure 3.5: GerbilSphere's 2D Perspective of a Network

method takes advantage of the focus+context techniques in which a focal point of the network can be examined while maintaining more of the context of the network structure. Maintaining the context is especially important as the number of nodes in the network increases.

### 3.5.1 Node Selection and Menus

The user can change the focus by clicking on a particular node of interest or clicking and dragging the cursor to scroll the 3D sphere and bring other nodes onto the screen. Picking through ray casting can be a time consuming event unless the number of object computations is reduced. Normally, each click to the screen will generate a single ray that must be checked against all node objects. This takes  $\Theta(n)$  complexity where  $n$  is the number of objects. By placing each node in a spatial data structure, faster ray object intersections will occur since only the nodes in the same grid volume as the ray must be checked. As described in Chapter 3.3, each node is already placed into a grid for the force-directed layout. Since each node is already in the 3D

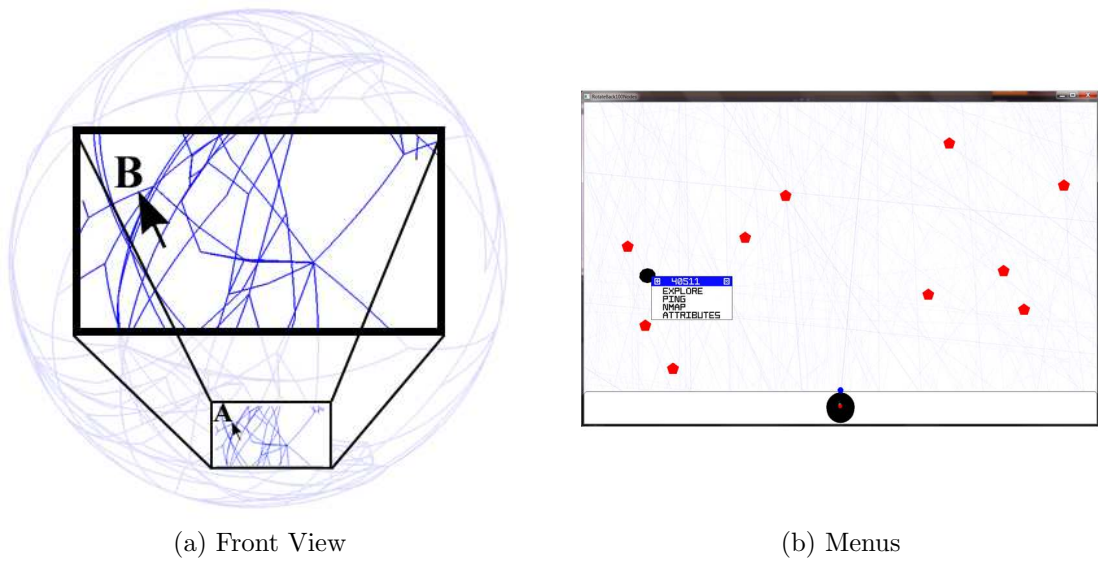
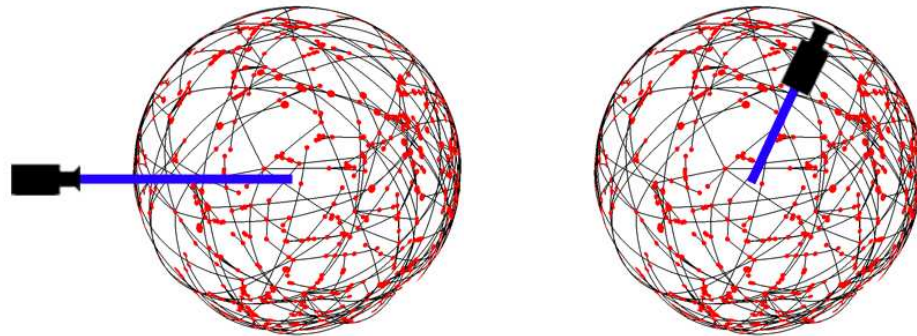


Figure 3.6: Node Selection and Menu.

grid, the grid is also used for picking. Picking is accomplished by first checking if the vector formed where the user clicks intersects with the whole visualization sphere. The whole sphere is used as a top level bounding volume. Next, we calculate where the vector intersection occurs on the sphere. This can be seen in Figure 3.6a where a click on the near plane frustum at point **A** calculates the point **B** on the sphere. Finally, the grid and neighboring grids of the clicked location are searched to find the node that is closest to the click. Nodes are only considered if their distance is less than a set threshold such as the distance of the radius of a node.

GerbilSphere has a custom menu system where each node can have its own menu items as seen in Figure 3.6b. This custom menu system allows for easy management of the nodes of a network. Administrators can easily perform functions on the nodes through their menus and use them to remember points of interest.



(a) Negative tether length

(b) Positive tether length

Figure 3.7: Zoom + Pan on a Tether.

### 3.5.2 Interactive Zoom + Pan

As the network revolves around the user, a particular point of interest can be examined by zooming in on that point. The world camera is attached to one end of a tether which is telescoping, and the other end of the tether is attached to the center of the sphere. The camera can move along the tether to zoom in (Figure 3.7b) and zoom out (Figure 3.7a) completely if desired. The tether length can be positive and negative. The horizontal and vertical panning is inverted when the tether length is negative and larger than the radius of the sphere. This inversion ensures that the sphere will always spin in the direction the user drags their mouse. If the user continues to zoom into a node, eventually the user will be placed inside that node and the same navigation principles will apply.

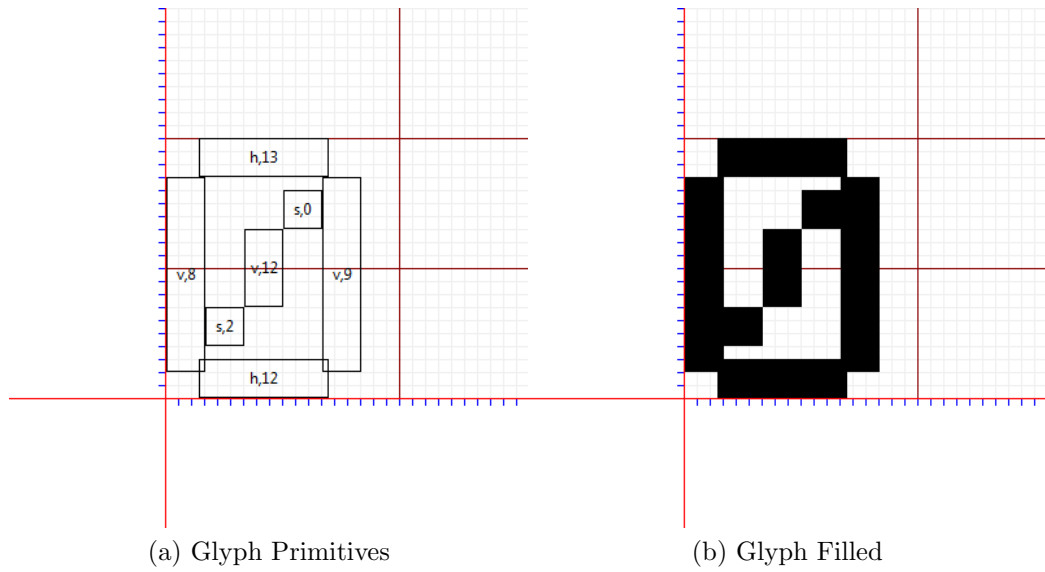


Figure 3.8: Font Construction Consists of Primitives

### 3.5.3 Labeling

In order to increase the speed of labeling huge graphs, GerbilSphere has its own custom vector font based on quad primitives (Figure 3.8). The advantage to this font is that it can be scaled, rotated, twisted, morphed, and extruded. With labels that have a fixed orientation, the data describing each character can be sent from the CPU to the graphics card and reside on the graphics card. This prevents the need for the CPU to send the font data for each frame since it is already in graphics card memory. The labeling of nodes and edges can be always on, or only appear for a selected object.

GerbilSphere currently supports several types of labeling styles for edges (Figure 3.9a) and for nodes (Figure 3.9b). In Figure 3.9a, **A** shows an edge made entirely of labels. **B** shows a label floating above the edge. **C** shows the alternating orientation of labels. **D** shows the label on the edge. **E** shows alternating orientation of labels on the edge. In Figure 3.9b, **A** shows a node's label as an internal horizontal label. **B**

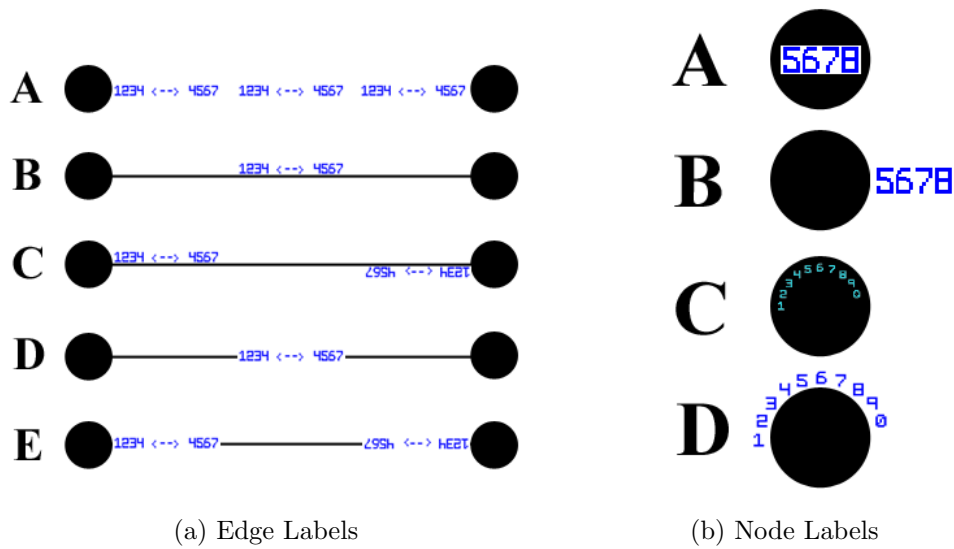


Figure 3.9: Node and Edge Labeling Types.

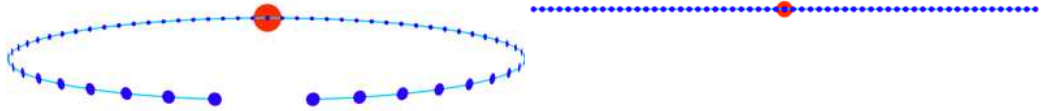
shows a label floating horizontally to the left. **C** shows an internal semi-circle label. **D** shows an external semi-circle label. If all labels are to be shown, then a clipping technique is used as to only show labels that appear in the viewing frustum, thus improving rendering time. Moreover, it may not be practical to show all labels for huge graphs.

# Chapter 4

## Experiments

### 4.1 User Study Overview

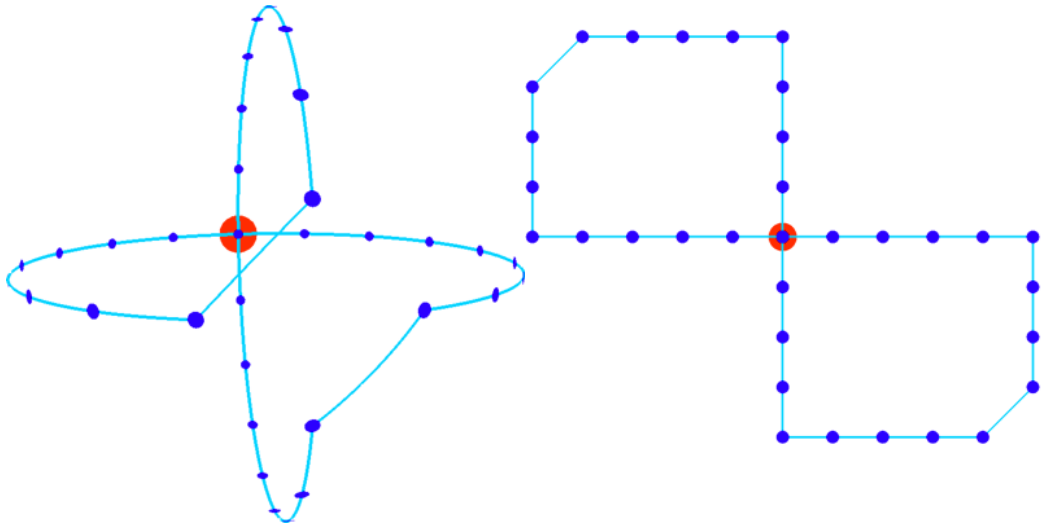
In order to test the usefulness of the GerbilSphere visualization tool, an empirical study was conducted that compared the usability of the Inner Sphere layout method to a 2D flat layout method. Two experiments were done that required path tracing and clicking nodes on two different topologies. These topologies were placed on a flat surface as well as inside of a sphere. The first topology was a series of nodes connected in a line as shown in Figure 4.1. We name this the *single-axis* topology experiment since path tracing the topology only requires traveling on a single axis. The second topology was in a cross shape as seen in Figure 4.2. We call this one the *multi-axis* topology experiment since tracing the connections requires navigation on two axes. Color changes and scaling have been made to the figures to enhance visual aesthetics for printing.



(a) Single-axis Sphere

(b) Single-axis Flat

Figure 4.1: Single-axis Topology on a Spherical and Flat Surface.



(a) Multi-axis Sphere

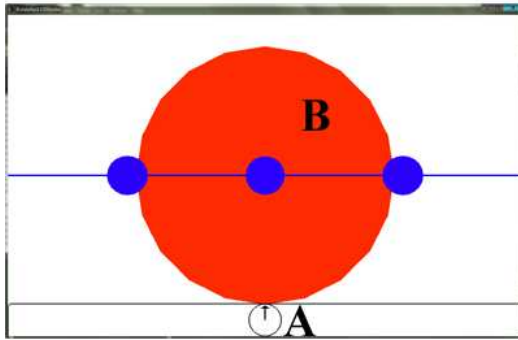
(b) Multi-axis Flat

Figure 4.2: Multi-axis Topology on a Spherical and Flat Surface.

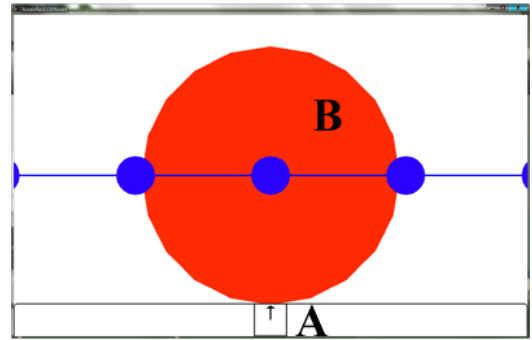
### 4.1.1 Single-Axis Topology Overview

Is the world flat or is it round? Early sea travelers might have asked this question and thought to themselves when leaving home “If the world is flat then I have to turn around and go back the way I came, to get home. If the world is round then I can keep going and I’ll eventually end up back home”. In the *single-axis* experiment the hypothesis formulated was that a user’s spatial perception of a sphere (i.e., its continuous surface) would allow faster path tracing and clicking of nodes than tracing and clicking the same topology on a conventional 2D flat surface. Mathematically, the graph theoretic distance of the *single-axis* topology is the same when it is placed on a sphere or flat surface. The physical distance and angle between connected nodes is also the same for both surfaces. With the visualization camera placed 1,400 pixels from the nodes, the path tracing of the topology on either the spherical or flat surface appears virtually identical. Due to the curvature of the sphere, there is a 12 pixel distortion of the edges of the screen which causes 12 additional pixels to be seen on the flat surface that are not visible on the sphere. This can be seen by comparing Figures 4.3a and 4.3b. We could not control for this 12 pixel difference on the edges of the screen due to the nature of the sphere. This distortion corrects itself as the nodes move towards the center and the correction is visually undetectable. Furthermore, this visual difference gives no indication to subjects about what surface they are navigating.

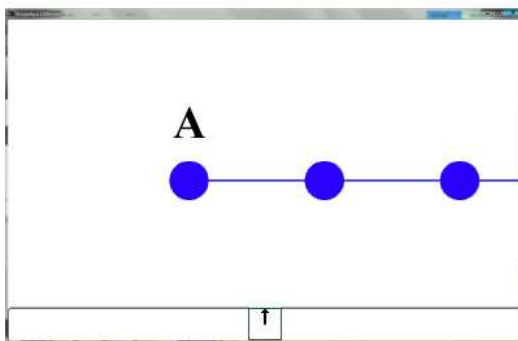
We did want subjects to know what surface they were navigating so a 2D shape was placed around the north pointing arrow (see letter **A** in Figures 4.3a and 4.3b). Apart from these differences the visual result of both surfaces is identical. There is however, a spatial difference between the two surfaces. When the *single-axis* topology is placed on a sphere, the average physical distance between all nodes is smaller than



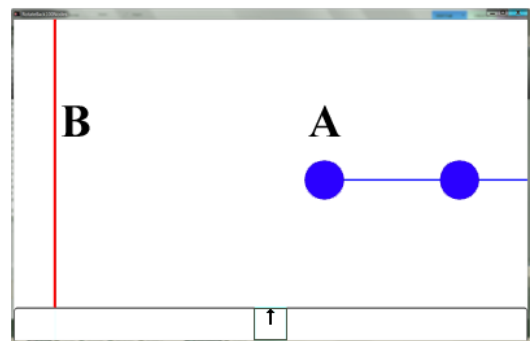
(a) Single-axis on Sphere, Start



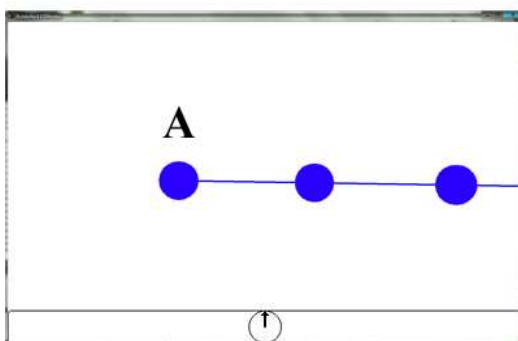
(b) Single-axis on Flat, Start



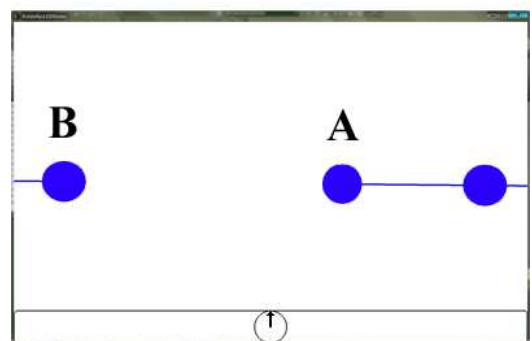
(c) Single-axis on Flat, End Node A



(d) Single-axis on Flat, Boundary B



(e) Single-axis on Sphere, End Node A



(f) Single-axis on Sphere, End Node B

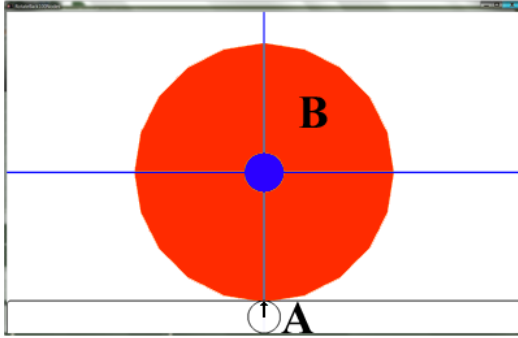
Figure 4.3: Single-axis Start and End Positions on Flat and Sphere Surfaces

the average distance on a flat surface. Therefore, user navigation on the sphere should always be faster when tracing a straight path that almost spans the circumference if the user knows their longitude. If the user does not know their longitude then circumnavigation of the sphere would be in faith that the surface will bring them back to their starting position. The starting screen seen by the subjects is shown in Figure 4.3a and 4.3b. Notice how the subject can either trace the path to the left or the right since this topology has a left and right leg.

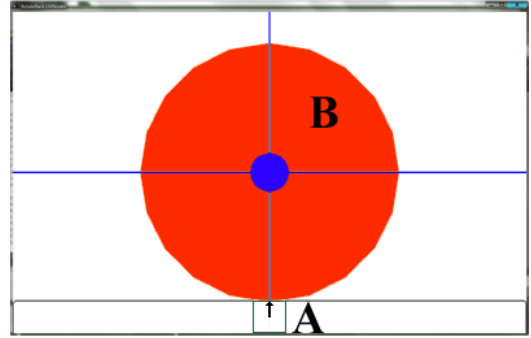
This study tests whether subjects have a concept of the sphere’s continuous surface by seeing if they choose to circumnavigate the sphere when reaching the last node of one of the legs without longitudinal positioning. In order to test this, on the spherical surface, upon reaching the last node in the topology, the other end of the topology, which is not connected, does not come into view unless the user scrolls far enough as to bring the last node past the middle of the screen as seen in Figure 4.3e and 4.3f. We consider this action to be an intention to keep scrolling around the sphere. If the user does not scroll over far enough they will not see the other end and may decide to turn back around and waste time by tracing a path they have already traveled. On the flat surface, if the user scrolls the last node (Figure 4.3c) past the middle of the screen, they will see the boundary of the topology (Figure 4.3d). The original color scheme of the user study consisted of a black background with white nodes and yellow edges connecting the nodes. The large starting circle, seen as red and labeled **B** in Figures 4.3a and 4.3b, was blue. Color changes and scaling have been made to the figures to enhance visual aesthetics for printing. The independent variable was “network layout surface” with two levels (Sphere and Flat). The dependent variable was how many unique nodes subjects could click on the *single-axis* topology. This was a repeated measures design.

### 4.1.2 Multi-Axis Topology Overview

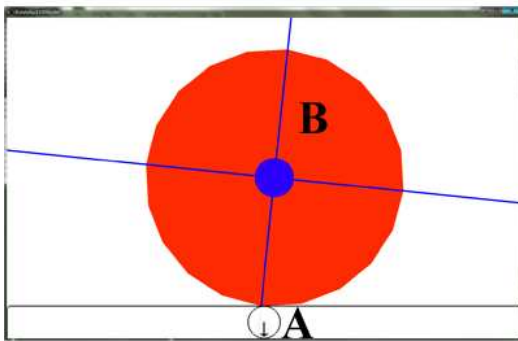
In the *multi-axis* experiment the hypothesis formulated was that a user's spatial orientation during navigation on a sphere is equal to his/her spatial orientation during navigation on a flat surface. Due to the nature of the sphere, when a user follows a straight line that covers 180 degrees, the user will be upside down upon reaching the end of the line. Inversion during navigation can be confusing. To aid the user with orientation, we place a small arrow at the bottom of the screen that points north (see letter **A** in Figures 4.4a, 4.4b, 4.4c and 4.4d). The *multi-axis* topology is designed to invert the user's orientation upon following any of its paths. User's who notice the north pointing arrow change and point down will have to figure out which path they have already traversed since arriving back at the start looks similar even with an inverted orientation (Figure 4.4c and 4.4d). Upon arriving back at the beginning, the pitch of the camera can be a little tilted (Figure 4.4c) on the sphere surface but apart from that and the north pointing arrow, there is no other indication of orientation. Nodes that were clicked before are reset to their original color again when another node is clicked. The original color scheme of the user study consisted of a black background with white nodes and yellow edges connecting the nodes. The large starting circle, seen as red and labeled **B** in Figures 4.4a, 4.4b, 4.4c and 4.4d, was blue. Color changes and scaling have been made to the figures to enhance visual aesthetics for printing. The independent variable was "network layout surface" with two levels (Sphere and Flat). The dependent variable was how many unique nodes subjects could click on the *multi-axis* topology. This was a repeated measures design.



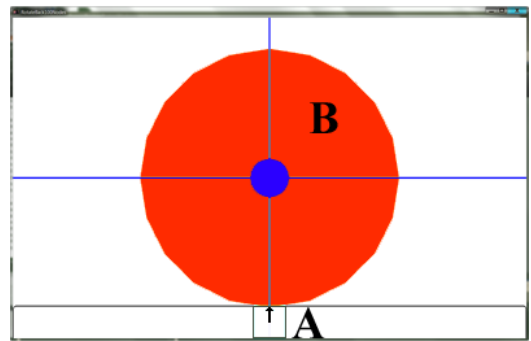
(a) Multi-axis on Sphere, Start



(b) Multi-axis on Flat, Start



(c) Multi-axis on Sphere, Ambiguous



(d) Multi-axis on Flat, Ambiguous

Figure 4.4: Multi-axis Start and End Positions on Flat and Sphere Surfaces

## 4.2 User Study

### 4.2.1 Subjects

We recruited 35 computer science students from University Nevada, Reno to participate in this study. All participants were volunteers and presumably had previous experience with click and drag tasks using a mouse.

### 4.2.2 Equipment

The computer used for the study had an Intel Core 2 Duo 3.16 GHz chip with 4GB of memory. A 19-inch monitor 1200x900 and normal QWERTY keyboard and optical mouse were connected to the computer. The GerbilSphere visualization engine was used to present subjects with spherical and flat network topologies.

### 4.2.3 Task

The subject's task was to click on as many unique nodes as possible within a minute. Nodes could be found on the topology by tracing the path formed by edges. Subjects were asked to click on as many white dots (nodes) as possible within the allotted time. The subjects were free to navigate the topology and click on the dots in any order. This required the memorization of dots previously clicked on since all dots reset themselves to white after another dot is clicked. A penalty was not given for clicking on the same dot twice, but this would be a waste of time. The following cues were set up to aid memorization of the dots already clicked:

- 1) A single large dot was placed at the center of the topology to inform the subject of where they started (large dot labeled **B** in Figures 4.4a, 4.4b, 4.4c and 4.4d).
- 2) Lines (edges) connected the dots and created a connection pattern.
- 3) A 2D compass was placed at the bottom of the screen to indicate which direction was North (see label **A** in Figures 4.4a, 4.4b, 4.4c and 4.4d). This compass always points up on the flat surface.
- 4) A square surrounded the compass to indicate the flat surface was being used (label **A** in Figures 4.4b and 4.4d).
- 5) A circle surrounded the compass to indicate the spherical surface was being used (label **A** in Figures 4.4a and 4.4c).

These cues were important because no other visual cues, such as latitude or longitude, were given to the subjects to inform them of what surface they were navigating.

#### **4.2.4 Procedure**

Participants were asked to sit at a computer and complete the experiments they were presented with. The first task was a training program that ensured the students were comfortable with clicking and dragging the screen. This was given to reduce any learning effect that might occur. During the training program, subjects were

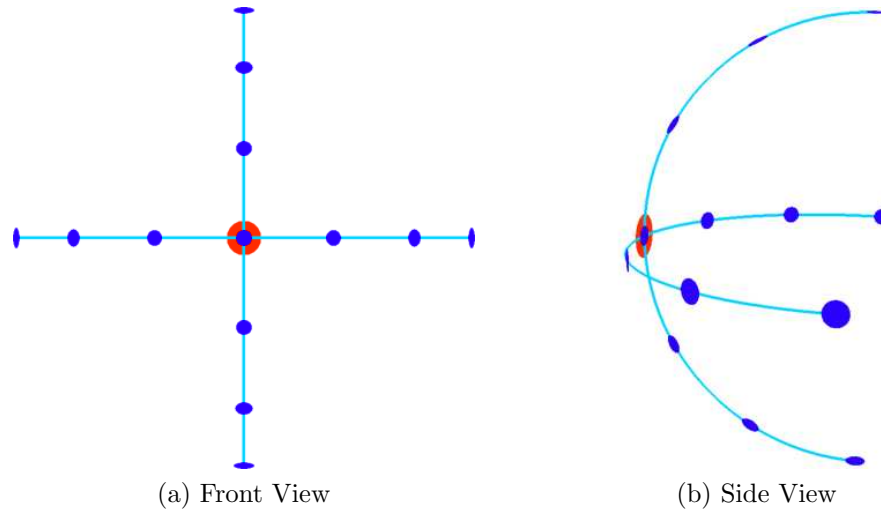


Figure 4.5: Topology of the Network Displayed During the One Minute Training.

told about the actual experiments that followed their training. They were told that they would be navigating network topologies and that the topologies would be on a flat surface or a spherical surface viewed from the inside of the sphere. They were also informed about the compass and shape that surrounded the compass and its meaning. They were also told that the only way to distinguish between what surface they were on and their orientation, was the compass and shape, as seen as label **A** in Figure 4.4a. The training program was on a spherical surface (Figure 4.5). After the students completed a 1 minute training session, the experiment started. There were four one minute trials given which consisted of two separate experiments. Each trial involved a map navigation task which required the use of the mouse for clicking and dragging and searching for white dots. Two of the four trials were the *single-axis* topology experiment and the other two were the *multi-axis* experiment. The order that the trials were presented changed with each participant. In order to control for the learning effect, all 24 ordering combinations of the trials were used. The software automatically logged when and where the node clicks occurred.

### 4.3 Results

The *single-axis* topology and the *multi-axis* topology results are separately analyzed below.

#### Single-Axis Topology Analysis

A paired-samples t-test was conducted to compare the number of dots clicked in a minute on the flat and spherical surfaces using the *single-axis* topology (Tables 4.1-4.2). There was a significant difference in the scores between the sphere (M=52.4, SD=7.31) and the flat surface (M=47.14, SD=9.29) conditions;  $t(34)=-4.25$ ,  $p = 0.05$ .

These results suggest that navigating the sphere on a single axis topology is faster than navigating a flat surface of the same topology when the topology almost spans the circumference of the sphere. Specifically, our results suggest that people's spatial perception of a sphere allows them to loop around and cover new ground faster.

Table 4.1: Singe-Axis Paired Samples Statistics

Paired Samples Statistics		
	Flat Surface	Spherical Surface
Mean	47.14	52.4
Observations (N)	35	35
Standard Deviation	9.29	7.31
Standard Error Mean	1.57	1.23

Table 4.2: Single-Axis Paired Samples Test

Paired Samples Test	
	Flat Surface - Sphere Surface
Mean	-5.26
Standard Deviation	7.32
Standard Error Mean	1.237
95% Conf Interval of Dif Lower	-7.77
95% Conf Interval of Dif Upper	-2.74
t Stat	-4.2502
df	34
P(T<=) two-tail	0.0002

### Multi-Axis Topology Analysis

A paired-samples t-test was conducted to compare the number of dots clicked in a minute on the flat and spherical surfaces using the *multi-axis* topology (Tables 4.3-4.4). There was a significant difference in the scores between the sphere (M=5.4, SD=1.14) and the flat surface (M=9.4, SD=1.14) conditions;  $t(34)=-4.25$ ,  $p = 0.05$ .

These results suggest that navigating the sphere on a topology that inverts the orientation is slower than navigating a flat surface of the same topology. Specifically, our results suggest that people's spatial perception on a sphere, when inverted, confuses them so they sometimes trace the same path.

Table 4.3: Multi-Axis Paired Samples Statistics

Paired Samples Statistics		
	Flat Surface	Spherical Surface
Mean	28.06	25.29
Observations (N)	35	35
Standard Deviation	4.3	6.28
Standard Error Mean	0.73	1.06

Table 4.4: Multi-Axis Paired Samples Test

Paired Samples Test	
	Flat Surface - Sphere Surface
Mean	2.77
Standard Deviation	5.02
Standard Error Mean	0.848
95% Conf Interval of Dif Lower	1.05
95% Conf Interval of Dif Upper	4.5
t Stat	3.267
df	34
P(T<=) two-tail	0.0025

### Discussion

As previously mentioned, the goal of this experiment was to test user performance on a 2D layout projected on the surface of a sphere and also a flat surface. This was done through presenting subjects with path tracing and clicking tasks that involved a *single-axis* and *multi-axis* topology. Each topology required different skills to navigate successfully. The data collected for both experiments contained information about how many unique nodes were clicked within a minute. In both experiments, there was a statistical significance between the number of dots clicked in a minute on the flat and sphere surfaces. The results of the *single-axis* experiment favor the use of a spherical network on a task that involves path tracing which almost spans the entire circumference. These findings are not to say that flat layouts are a bad choice, just that in some circumstances, the structure of a sphere layout allows for faster navigation, since there is no boundary to the surface. The results of the *multi-axis* experiment favor the use of a flat surface since inversion on a flat surface does not happen. This is a serious consideration when designing a visualization system that uses a sphere. Hence, the system should present better clues to indicate pole inversion.

# Chapter 5

## Internet Visualization

### Implementation

The topology of the Internet is collected using trace-route probes that are sent out from several vantage points. Our goal is to take that information and create a useful visualization that can be easily navigated using GerbilSphere. The connected entities of the Internet can be grouped in many ways. Autonomous Systems (ASes) of the Internet define entities that are under a single administrative authority. We chose to start with the visualization of the ASes since their peering relationships makes up the high level topology of the Internet. The first level of the Internet graph we visualize consists of the ASes and the user can then zoom into a specific AS for further investigation of router-level topology.

## 5.1 Data Source

We used AS data-set collected by CAIDA [38]. This data set is derived from individual traceroute probes that return IP addresses. These probes are sent out by monitors distributed over the world. The collected IP addresses are then looked up and their AS number is identified. Sometimes, an AS number cannot be derived from an IP addresses. Other times, ASes advertise the same number. We found 19,541 unique ASes with 49,270 links between them in the IPv4 routed /24 AS links dataset. We then created an AS graph that was converted into the XGMML format for use with GerbilSphere.

## 5.2 Internet Layout

We used our non-Euclidean force-directed grid based layout for the AS graph. After the layout was baked, the node positions were saved into a file for faster loading. The initial baking of the 19,541 nodes with the 49,270 edges on an Intel T2600 @2.16GHZ 2.17GHZ machine with 2.0GB of memory took 4 minutes. Loading a baked graph is considerably faster than waiting for it to bake.

## 5.3 Display Environments

The GerbilSphere Internet visualization was tested on two different display environments. The first display environment was a standard desktop application, as shown in Figure 5.1. This application prototype currently uses GLUT for window management and user interaction. Our own menu system was also implemented to allow easier cross-platform portability. If more screen real estate is needed for visualiza-

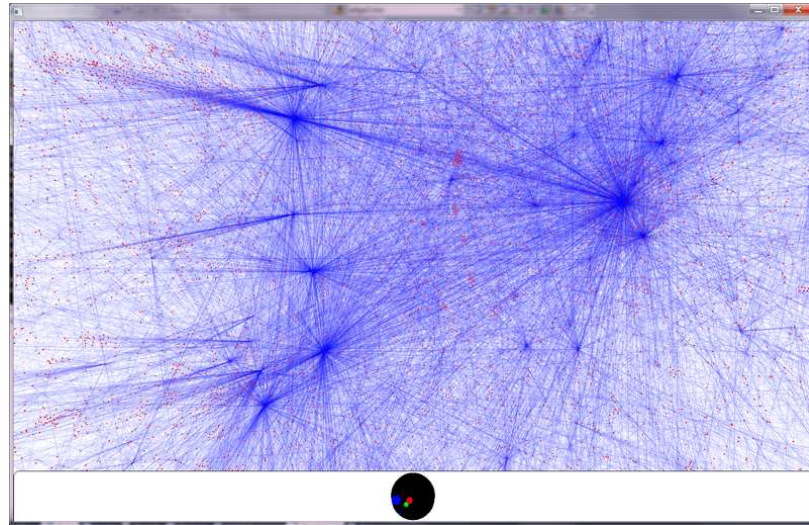


Figure 5.1: Desktop Version.

tion, multiple displays can be used to show more information at once. Navigation of GerbilSphere in this desktop application environment requires the user to click and drag their mouse for panning; furthermore, zooming is done with the mouse wheel.

The second display environment was an Interactive Virtual Environment (IVE) called drive6 which is located at the Desert Research Institute, Reno, Nevada campus. Drive6 is a virtual reality cave with six walls that uses multiple projectors in combination with shutter glasses and head and wand tracking systems to create a virtual reality experience. The rendering contexts of the walls was handled through the use of the Hydra library [35]. The Inner Sphere method of network visualization allowed for multiple users to view the network at the same time. This multi-user application can be seen in Figure 5.2. Navigation in this environment requires the user to look around inside the sphere, since the sphere is stationary. Node selection in this environment is done with a wand and multiple users can have their own wand.

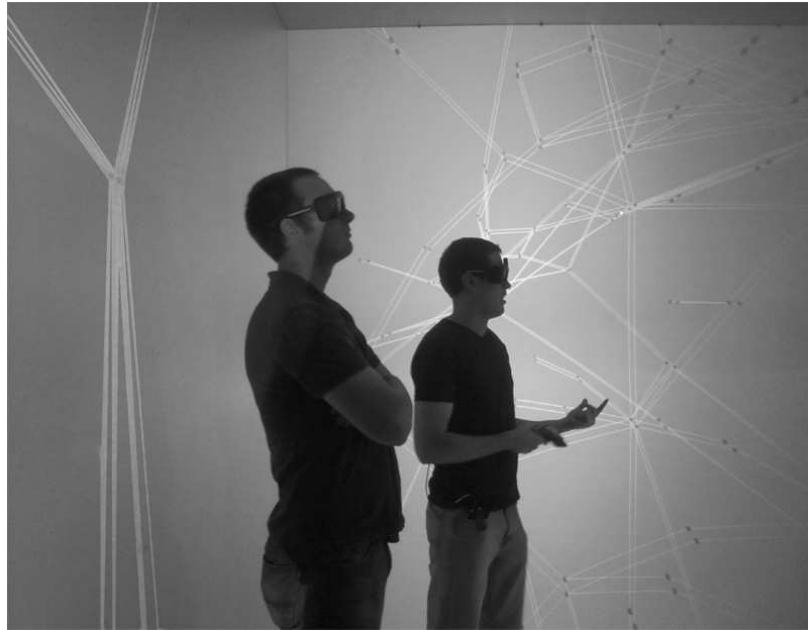


Figure 5.2: Multi-User VR Version.

## 5.4 Visual and Overall Assessment

The visual result of the AS-level Internet topology is highly subjective (Figure 5.3). We could base our measure on desired aesthetic criteria, such as the crossing number, but the final call we believe should be on its usefulness. Our goal was to design a useful system for the visualization of the Internet topology and we believe we have accomplished that. The visualization system is scalable since new vertices can be added to the currently baked graph layout without having to recalculate the entire layout. The system is also extensible since custom menus can be given to each vertex. The 2 1/2 dimensional interface is very user friendly and allows for easy management of the vertices from the menu system GerbilSphere provides.

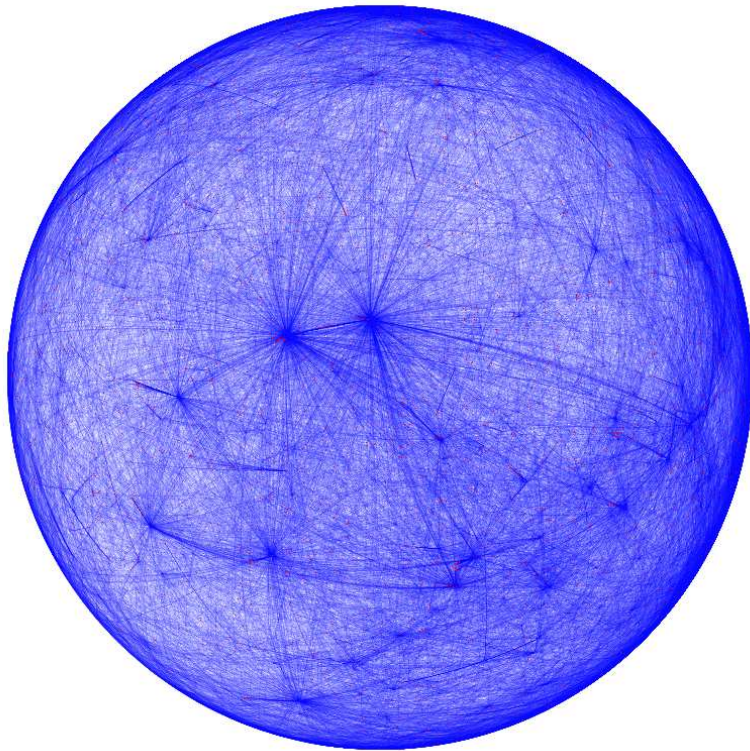


Figure 5.3: The AS Internet Sphere

# Chapter 6

## Conclusion and Future Work

### 6.1 Conclusions

In this thesis, we developed a prototype software system called GerbilSphere that provides an Inner Sphere visualization framework capable of visualizing large-scale network topologies. With limited screen space, the viewing of a large network such as the Internet is difficult since many objects can occlude each other. We used force-directed layout algorithms to neatly position the nodes of the Internet. To handle the issue of node occlusion, we placed the topology on the surface of a sphere and viewed the topology from the inside of the sphere. Design considerations for this visualization tool have been taken from several empirical studies that aimed to understand human computer interaction.

We also contributed to the understanding of Inner Sphere navigation through a user study. We conducted our own empirical study that tested the usefulness of visualizing networks from the inside of a sphere compared to a flat surface. We found that a person's understanding of a spherical surface allows for faster navigation with

the Inner Sphere method which is essential when navigating huge networks such as the Internet topology. It was also found, in our user study, that becoming inverted while navigating the inside of a sphere is confusing to the user. To address the issues from the user study, we devised stronger orientation cues in the GerbilSphere program to aid the user.

GerbilSphere strives for speed when visualizing large topologies. It stores the topology in two forms, one on the CPU and one on the GPU since it is slow to move data between the CPU and GPU at every frame. We have found that GerbilSphere generates reasonable frame rates for interactive visualization. The implementation of GerbilSphere is stable and ready for the future work discussed in the next section. GerbilSphere can be a worthy supplement for current large-scale network visualization systems.

## 6.2 Future Work

There are many ways to improve the GerbilSphere tool. The work completed so far strongly indicates that Inner Sphere visualization deserves further exploration in the visualization of different domain specific network topologies. Furthermore, rendering speed optimizations and the user interface are areas that would benefit from further investigation. To further improve GerbilSphere, different spatial data structures could be tested against the currently used volume grid structure. Another improvement, that could be investigated, is faster computing of the force-directed layout. Improvements with layout speeds have already been achieved through parallelization on the GPU and parallelization over multiple resources connected through a network. Faster layouts would improve transition times between nested graphs.

A user study was performed to assess the usefulness of Inner Sphere visualization on a desktop computer. However, the usability of GerbilSphere in a virtual environment was tested but no formal study was conducted. If GerbilSphere is to be used in a virtual environment, different studies may assess its true value in that context.

Finally, the visualization environment can be converted into a simulation tool where we can change the values of a few factors and see how much impact it can have on the visualized graph.

# Bibliography

- [1] InterMapper. Dartware. <http://www.intermapper.com/products/intermapper/>.
- [2] Ipsonar mapviewer visualizing and analyzing results. Lumeta. <http://www.lumeta.com/mapviewer/>.
- [3] J. I. Alvarez-hamelin, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in Neural Information Processing Systems 18*, pages 41–50. MIT Press, 2006.
- [4] S. Asafi and M. Sandler. The dimes visualizer - a 4 dimension graph visualizer. <http://www.netdimes.org/new/?q=node/17>.
- [5] D. Auber. Tulip-a huge graph visualization framework. <http://tulip.labri.fr/TulipDrupal/>.
- [6] G. Barbagallo, A. Carmignani, G. D. Battista, W. Didimo, and M. Pizzonia. Exploration and visualization of computer networks: Polyphemus and hermes. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, pages 444–445. Springer, 2002.
- [7] V. Batagelj and A. Mrvar. Pajek - analysis and visualization of large networks. In *Graph Drawing Software*, pages 77–103. Springer, 2003.
- [8] S. Benford, C. Brown, G. Reynard, and C. Greenhalgh. Shared spaces: Transportation, artificiality and spatiality. *Proceedings of ACM CSCW '96*, 18:77–86, 1996.
- [9] B.-J. J. Breitkreutz, C. Stark, and M. Tyers. Osprey: a network visualization system. *Genome biology*, 4(3), 2003.
- [10] I. Brus and A. Frick. Fast interactive 3-D graph visualization. In *Proceedings of Graph Drawing*, Lecture Notes Computer Science 1027:99–110, 1996.
- [11] C. Buchheim, M. Junger, and S. Leipert. Improving Walker's Algorithm to Run in Linear Time. In *Proceedings of Graph Drawing*, 2528 of Lecture Notes Computer Science:344–353, 2002.

- [12] J. Carrire and R. Kazman. Interacting with Huge Hierarchies: Beyond Cone Trees. In *Proc. IEEE Information Visualization '95, IEEE Computer Press, Los Alamitos, CA*, pages 74–81. IEEE, 1995.
- [13] B. Cheswick, H. Burch, and S. Branigan. Mapping and Visualizing the Internet. In *In Proceedings of the 2000 USENIX Annual Technical Conference*, pages 1–12, 2000.
- [14] A. Cockburn. Revisiting 2D vs 3D Implications on Spatial Memory. In *Australian Computer Society, Inc*, pages 25–31, 2004.
- [15] A. Cockburn and B. Mckenzie. An evaluation of cone trees. In *People and Computers XV (Proceedings of the 2000 British Computer Society Conference on Human Computer Interaction) University of Sunderland*, pages 425–436. Springer-Verlag, 2000.
- [16] A. Cockburn and B. Mckenzie. 3D or not 3D? evaluating the effect of the third dimension in a document management system. In *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 434–441. ACM Press, 2001.
- [17] L. Colitti, B. F. Mariani, M. Patrignani, M. Pizzonia, and U. D. R. Tre. Delistr-0027 - visualizing interdomain routing with bgplay. article 0027. *Journal on Graph Algorithms and Applications*, pages 2004–2005, 2005.
- [18] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
- [19] D. Egan and M. Gomez. Assaying, isolating, and accomodating individual differences in learning a complex skill. In *Individual Differences in Cognition*, pages 173–217, New York, USA, 1985. R. Dillon, Academic Press.
- [20] D. Estrin, M. Handley, J. Heidemann, S. McCanne, Y. Xu, and H. Yu. Network visualization with nam, the vint network animator. *Computer*, 33(11):63–68, 2000. <http://isi.edu/nsnam/nam/>.
- [21] R. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, June 1962.
- [22] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings of Graph Drawing '94*, Lecture Notes Computer Science 894:388–403, 1994.
- [23] Y. Frishman and A. Tal. Multi-level graph layout on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 13:1310–1319, 2007.

- [24] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21:1129–1164, Nov. 1991.
- [25] G. Furnas. Generalized fisheye views. *Proceedings of HCI*, pages 16–23, 1986.
- [26] D. Gagnon. Videogames and spatial skills: an exploratory study. *Educational Communication and Technology*, 33(4):263–275, 1985.
- [27] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. In *Graph Drawing'00 Conference Proceedings*, pages 211–221, 2000.
- [28] A. Godiyal, J. Hoberock, M. Garland, and J. C. Hart. Rapid multipole graph drawing on the gpu. In *Proc. Graph Drawing*, volume 5417 of LNCS, pages 90–101. Springer, 2008.
- [29] M. H. Gunes and K. Sarac. Resolving IP Aliases in Building Traceroute-Based Internet Maps. Technical report, University of Texas at Dallas, December 2006.
- [30] R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. In P. Widmayer, G. Neyer, and S. Eidenbenz, editors, *Graph-Theoretic Concepts in Computer Science*, volume 1665 of *Lecture Notes in Computer Science*, pages 262–277. Springer Berlin / Heidelberg, 1999.
- [31] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Journal of Graph Algorithms and Applications*, 6(3):179–202, 2002.
- [32] M. A. Hearst and C. Karadi. Cat-a-cone: An interactive interface for specifying searches and viewing retrieval results using a large category hierarchy. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 246–255. ACM Press, 1997.
- [33] M. Hemmje. Lyberworld: a 3D graphical user interface for fulltext retrieval. In *Proc. ACM SIGCHI '95*, pages 417–418. ACM, 1995.
- [34] M. Himsolt. GML: A portable Graph File Format. Technical report, University of Passau 1997, 94030 Passau, Germany., 1997 (accessed November 30, 2010). <http://www.infosun.fim.uni-passau.de/Graphlet/GML/gml-tr.html>.
- [35] R. Hoang. *Hydra*, 2010 (accessed November 16, 2010). <http://www.cse.unr.edu/hpcvis/hydra>.
- [36] T. Hughes, Y. Hyun, and D. Liberles. Visualising very large phylogenetic trees in three dimensional hyperbolic space. *BMC Bioinformatics*, 5(1):48, 2004.

- [37] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, M. Luckie, K. Claffy, and C. Shannon. *Caida Walrus ries-t2-thumb*, 2010 (accessed November 16, 2010). <http://www.caida.org/tools/visualization/walrus/gallery1/ries-t2-thumb.png>.
- [38] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, M. Luckie, K. Claffy, and C. Shannon. *The IPv4 Routed /24 AS Links Dataset - 11/10/2010*, 2010 (accessed November 16, 2010). [http://www.caida.org/data/active/ipv4\\_routed\\_topology\\_aslinks\\_dataset.xml](http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml).
- [39] H. Jones. Network weathermap. <http://www.network-weathermap.com/>.
- [40] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
- [41] S. G. Kobourov and K. Wampler. Non-euclidean spring embedders. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):757–767, 2005.
- [42] M. Lad, L. Zhang, and D. Massey. Link-rank: A graphical tool for capturing bgp routing dynamics. In *in IEEE/IFIP NOMS, Seoul, Korea*, 2004.
- [43] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*, pages 401–408. ACM, 1995.
- [44] L. Lamping and R. Rao. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7(1):33–35, 1996.
- [45] B. Leitheiser and D. Munro. An experimental study of the relationship between spatial ability and the learning of a graphical user interface. In *Proceedings of the Inaugural Americas Conference on Information Systems*, pages 122–124, Pittsburgh, PA, 1995. R. Dillon, Academic Press.
- [46] B. Lyon. the opte project. <http://opte.org/>.
- [47] C. Mueller, D. Gregor, and A. Lumsdaine. Distributed force-directed graph layout and visualization. In *Eurographics Symposium on Parallel Graphics and Visualization*, 2006.
- [48] T. Munzner. *Exploring large graphs in 3D hyperbolic space.*, 1998 (accessed November 16, 2010). <http://graphics.stanford.edu/papers/h3cga/html/>.
- [49] T. Munzner and P. Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. *Proceedings of the VRML '95 Symposium, ACM SIG-GRAPH*, ACM Press, 1995.

- [50] T. Munzner, E. Hoffman, K. Claffy, and B. Fenner. Visualizing the global topology of the mbone. In *Proceedings of the 1996 IEEE Symposium on Information Visualization (INFOVIS '96)*, INFOVIS '96, pages 85–92, Washington DC USA, 1996. IEEE Computer Society.
- [51] J. Oberheide, M. Karir, and D. Blazakis. Vast: visualizing autonomous system topology. In *VizSEC '06: Proceedings of the 3rd international workshop on Visualization for computer security*, pages 71–80, New York, NY, USA, 2006. ACM.
- [52] R. V. Oliveira, B. Zhang, and L. Zhang. Observing the evolution of internet as topology. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 313–324, New York, NY, USA, 2007. ACM.
- [53] J. Punin and M. Krishnamoorthy. *XGMML (eXtensible Graph Markup and Modeling Language*, 2001 (accessed November 30, 2010). <http://www.cs.rpi.edu/~puninj/XGMML/draft-xgmml.html>.
- [54] H. Purchase, D. Carrington, and J. Alder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7:233–255, 2002.
- [55] A. Quigley and P. Eades. Fade graph drawing, clustering, and visual abstraction. In *Proceedings of the 8th Symposium on Graph Drawing*, volume 1984 Lecture Notes in Computer Science, pages 197–210, 2000.
- [56] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, Mar. 1981.
- [57] K. Ridsen, M. P. Czerwinski, T. Munzner, and D. B. Cook. An initial examination of ease of use for 2D and 3D information visualizations of web content. *International Journal of Human-Computer Studies*, 53:695–714, 2000.
- [58] G. Robertson, S. Card, and J. Mackinlay. Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):57–71, April 1993.
- [59] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. V. Dantzich. Data mountain: Using spatial memory for document management. In *UIST '98*, pages 153–162, 1998.
- [60] G. Robertson, M. V. Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Ridsen, D. Thiel, and V. Gorokhovskiy. The task gallery: a 3D window manager. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems*, pages 494–501. ACM Press, 2000.

- [61] G. Robertson, J. Mackinlay, and S. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proc. ACM SIGCHI 91 Conf. on Human Factors in Computing Systems*, pages 189–194, April 1991.
- [62] H. S. Smallman, M. S. John, H. M. Oonk, and M. B. Cowen. Information availability in 2D and 3D displays. *IEEE Computer Graphics and Applications*, 21:51–57, 2001.
- [63] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rock-etfuel. In *In Proc. ACM SIGCOMM*, pages 133–145, 2002.
- [64] M. Tavanti and M. Lind. 2d vs 3d, implications on spatial memory. In *INFOVIS '01: Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, page 139, Washington, DC, USA, 2001. IEEE Computer Society.
- [65] A. Tikhonova and K.-L. Ma. A scalable parallel force-directed graph layout algorithm. In K. L. M. J. Favre and D. Weiskopf, editors, *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, pages 25–32, 2008.
- [66] K. J. Vicente, B. C. Hayes, and R. C. Williges. Assaying and isolating individual differences in searching a hierarchical file system. *Hum. Factors*, 29(3):349–359, 1987.
- [67] J. Walker. A node-positioning algorithm for general trees. *Software: Practice and Experience*, 20(7):685–705, 1990.
- [68] C. Walshaw. A multilevel algorithm for force-directed graph drawing. *Journal of Graph Algorithms and Applications*, 7(3):253–285, 2003.
- [69] C. Ware. Designing with a 2 1/2-D attitude. *Information Design Journal*, 10(3):255–262, 2001.
- [70] C. Wetherell and A. Shannon. Tidy drawings of trees. *IEEE Transactions on Software Engineering*, SE-5:514–520, 1979.