Graphical Fisheye Views

Manojit Sarkar and Marc H. Brown

Department of Computer Science
Brown University
Providence, Rhode Island 02912

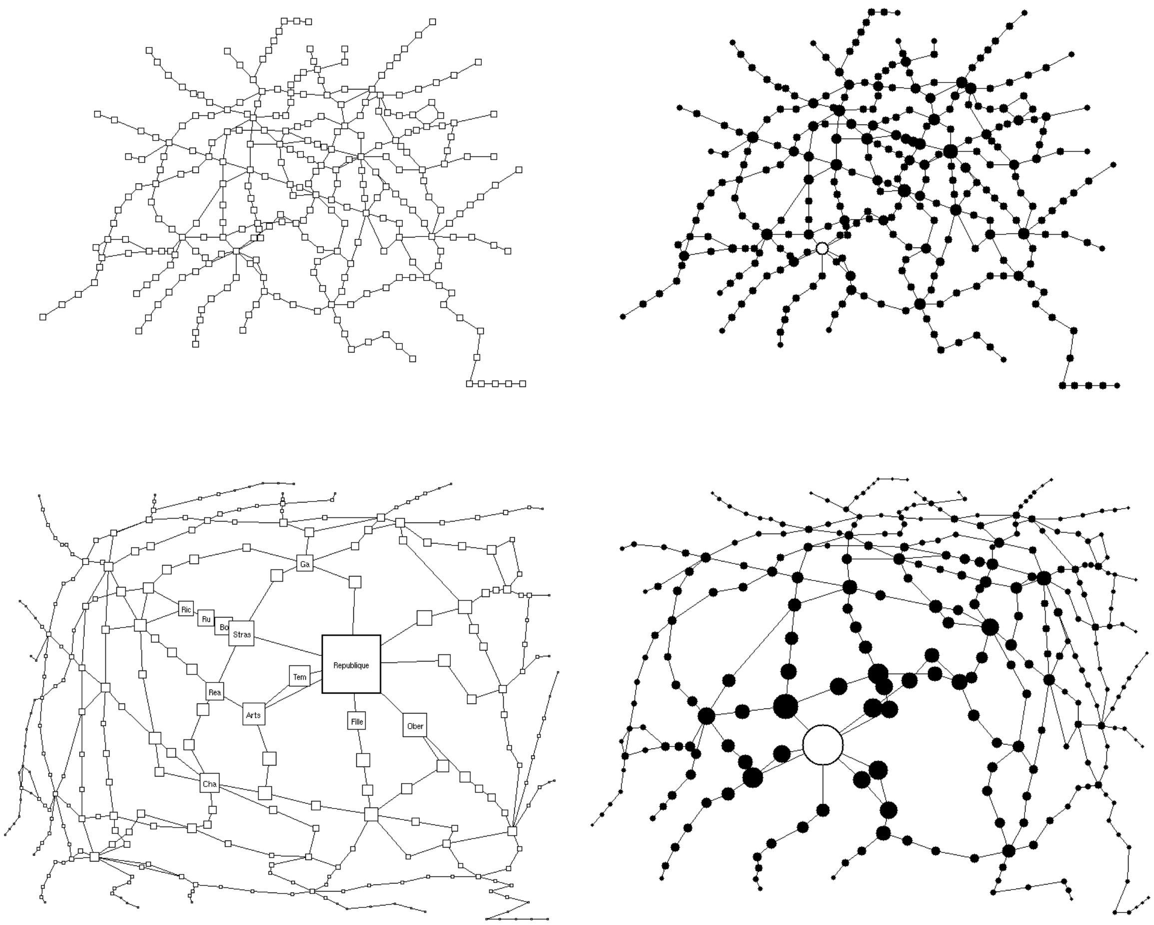
CS-93-40
September 1993

Graphical Fisheye Views

Manojit Sarkar and Marc H. Brown

March 17, 1992

March 23, 1993 (Revised)



About the Title-Page Images

The images on the preceeding page are views of a graph representing the Paris Metro system. The vertices in the graph are the stations, and the edges are the connections between stations. All images are screen dumps from the prototype system described in this paper.

The upper-left image is a normal view of the Metro; the other images are fisheye views of the Metro. In all graphs, the *a priori importance* (API) assigned to each station is the number of connecting stations.

In the upper-right image, the sizes of vertices vary according to the API of each station. The *focus* is the Montparnasse-Bienvenue station, displayed as a hollow circle. The user selects a focus by clicking with the mouse.

In the lower-right image, the vertices that are close (using Euclidean distance) to the focus station are magnified, and those far away are shrunk. In addition, the locations of all vertices are changed slightly in order to give the larger vertices more space.

In the lower-left image, the focus station is changed to be Republique, and the representation of the vertices is changed to one that displays the name of the station, space permitting.

(The technical details for the images, the meanings of which is explained in Section 4, are as follows: In all images, $c = 0.3$, $e = 0.3$, and $VWcutoff = 0$. In the upper images, $d = 0$. In the lower images, $d = 2$. All transformations are polar.)

Of course, a series of static snapshots cannot do justice to an interactive system: You need to use your imagination to visualize how the upper-right image smoothly transformed into the lower-right image, as the user moved a slider controlling the amount of "distortion" from 0 to 2. Visualize also how the lower-right image smoothly transformed into the lower-left image, as the user dragged the mouse from Montparnasse-Bienvenue to Republique.

Publication History

A preliminary version of this paper appeared in the ACM CHI'92 conference proceedings, May 1992, under the title *Graphical Fisheye Views of Graphs*. A videotape showing the prototype system in action is part of the ACM CHI'93 conference video program, April 1993.

Author Affiliation and Contact Information

Manojit Sarkar is a Ph.D. candidate at Brown University. The bulk of the work described here was performed while he was supported by a research internship from SRC during the summer of 1991. Subsequent work has been supported in part by ONR Contract N00014–91–J–4052, ARPA Order 8225.

Department of Computer Science
Brown University
Providence, RI 02912
(401) 863–7672
`ms@cs.brown.edu`

Marc H. Brown is a member of the research staff at Digital Equipment Corporation's Systems Research Center in Palo Alto.

Systems Research Center
Digital Equipment Corporation
Palo Alto, CA 94301
(415) 853–2152
`mhb@src.dec.com`

Abstract

A *fisheye* camera lens is a very wide angle lens that magnifies nearby objects while shrinking distant objects. It is a valuable tool for seeing both "local detail" and "global context" simultaneously. This paper describes a system for viewing and browsing graphs using a software analog of a fisheye lens. We first show how to implement such a view using solely geometric transformations. We then describe a more general transformation that allows global information about the graph to affect the view. Our general transformation is a fundamental extension to previous research in fisheye views.

Categories and Subject Descriptors:

D.2.2 [**Software Engineering**]: Tools and Techniques–*User Interfaces*

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces–*Interaction styles*

I.3.6 [**Computer Graphics**]: Methodology and Techniques–*Interaction Techniques*

General Terms: Visualization

Additional Key Words and Phrases: Fisheye Views, Graph Layout, Information Visualization, Interactive Graphics, User Interfaces.

1 Introduction

Graphs with hundreds of vertices and edges are common in many application areas of computer science, such as network topology, VLSI circuits, and graph theory. There are literally hundreds of algorithms for positioning nodes to produce an aesthetic and informative display [2]. However, once a layout is chosen, what is an effective way to view and browse the graph on a workstation?

Displaying all the information associated with the vertices and edges (assuming it can even fit on a screen) shows the global structure of the graph, but has the drawback that details are typically too small to be seen. Alternatively, zooming into a part of the graph and panning to other parts does show local details but loses the overall structure of the graph. Researchers have observed that browsing a large layout by scrolling and arc traversing tends to obscure the global structure of the graph [6]. Using two or more views — one view of the entire graph and the other of a zoomed portion — has the advantage of seeing both local detail and overall structure, but has the drawbacks of requiring extra screen space and of forcing the viewer to mentally integrate the views. The multiple view approach also has the drawback that parts of the graph adjacent to the enlarged area are not visible at all in the enlarged view(s).

This paper explores a *fisheye* lens approach to viewing and browsing graphs. A fisheye view of a graph shows an area of interest quite large and with detail, and shows the remainder of the graph successively smaller and in less detail. It achieves this smooth integration of local detail and global context by repositioning and resizing elements of the graph. A fisheye lens seems to have all the advantages of the other approaches for viewing and browsing a graph, but without suffering from any of the drawbacks.

A typical graph is displayed in Figure 1, and fisheye versions of it appear in Figures 2–6. In the fisheye view, the vertex with the thick border is the current point of interest to the viewer. We call this point the *focus*. In our prototype system, a viewer selects the focus by clicking with a mouse. As the mouse is dragged, the focus changes and the display smoothly updates in real time. The size and detail of a vertex in the fisheye view depend on the distance of the vertex from the focus, a preassigned importance associated with the vertex, and the values of some user-controlled parameters. All figures in this paper are screen dumps of views generated by our prototype system.

Our work extends Furnas's pioneering work on fisheye views [5] by providing a graphical interpretation to fisheye views. We introduce layout considerations

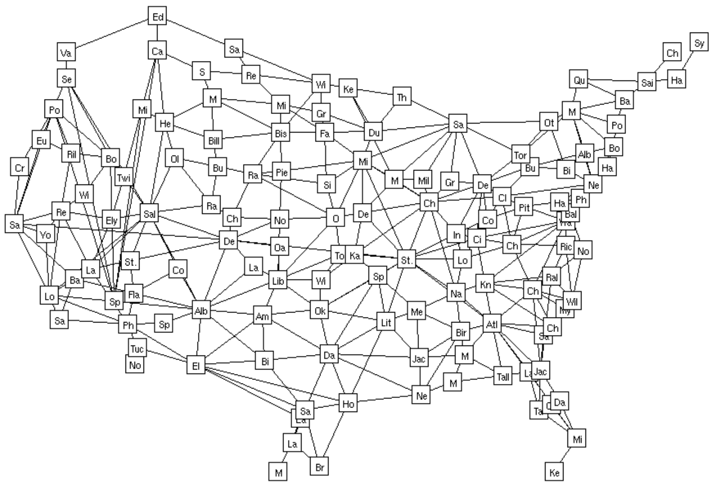


Figure 1: A graph with 134 vertices and 338 edges. The vertices represent major cities in the United States, and the edges represent paths between neighboring cities. (Typically, the edges would be annotated with the distance and driving time between the cities.) The *a priori importance* value assigned to each vertex is proportional to the logarithm of population of the corresponding city. Fisheye views of this graph appear in Figures 2–6.

into the fisheye formalism, so that the position, size, and level of detail of objects displayed are computed based on client-specified functions of an object's distance from the focus and the object's preassigned importance in the global structure. In Furnas's original formulation of the fisheye view, a component is either present in full detail or is completely absent from the view, and there is no explicit control over the attributes controlling the graphical layout. Also, Furnas's work is oriented for output on the (ascii) printers of that era; moreover, because components are either present or missing, it would probably not have lent itself well to an interactive system.

The next section defines the terminology and conventions used in the remainder of this paper. In Section 3 we present a formal framework for generating graphical fisheye views. Section 4 describes the strategy we used to implement the prototype viewer based on our formal framework, and Section 5 describes some alternative implementation strategies we explored. Section 6 describes our prototype system.

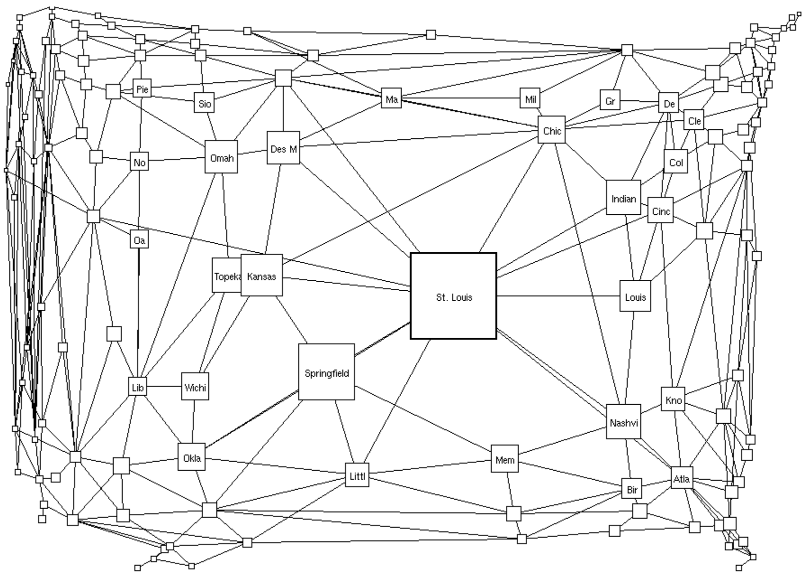


Figure 2: A fisheye view of the graph in Figure 1. The focus is on St. Louis. (The values of the fisheye parameters are $d = 5, c = 0, e = 0$, *VWcutoff* $= 0$; the meanings of these parameters are explained in Sections 4 and 6.)

In Section 7, we describe logical fisheye views (of the sort that Furnas described), and show how an implementation within our framework can be used for creating them. In the remaining sections, we review other related efforts and offer some thoughts on future directions.

2 Terminology

A graph consists of *vertices* and *edges*. The initial layout of the graph is called the *normal view* of the graph, and its coordinates are called *normal coordinates*. Vertices are graphically represented by shapes whose bounding boxes are square (chosen arbitrarily). Each vertex has a *position*, specified by its normal coordinates, and a *size* which is the length of a side of the bounding box of the vertex. Each vertex is also assigned a number to represent its relative importance in the global structure. This number is called the *a priori importance*, or the API, of the vertex.

An edge is represented by either a straight line from one vertex to another, or by a set of straight line segments to approximate curved edges. An edge consisting

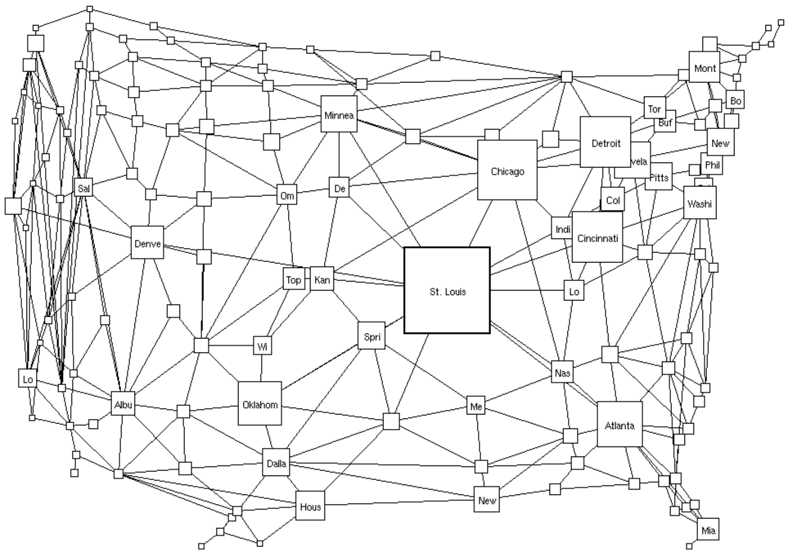


Figure 3: A fisheye view of the graph in Figure 1, with less distortion than in Figure 2. The values of the fisheye parameters are $d = 2, c = 0.5, e = 0.5, \mathit{VWcutoff} = 0$.

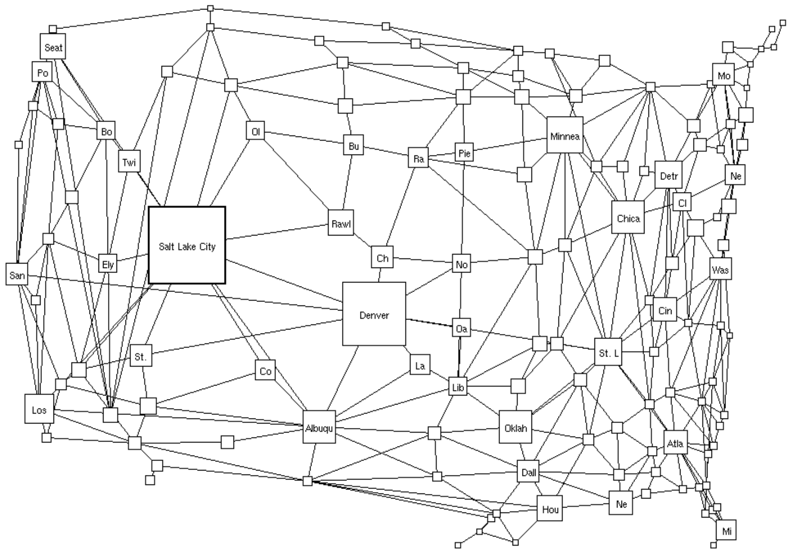


Figure 4: A fisheye view of the graph in Figure 1, with the focus on Salt Lake City. The level of distortion is the same as in Figure 3; only the location of the focus has changed. The values of the fisheye parameters are $d = 2, c = 0.5, e = 0.5, \mathit{VWcutoff} = 0$.

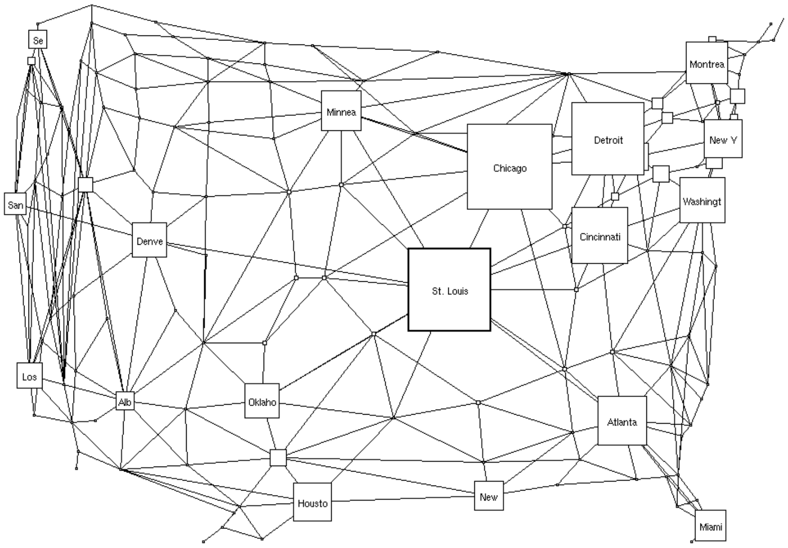


Figure 5: A fisheye view of the graph in Figure 1. Compare this to Figure 3, with the same distortion and the same focus. Here, the important vertices are larger than in Figure 3, but the unimportant ones are smaller. The values of the fisheye parameters are $d = 2, c = 0.75, e = 0.75, \mathit{VWcutoff} = 0$.

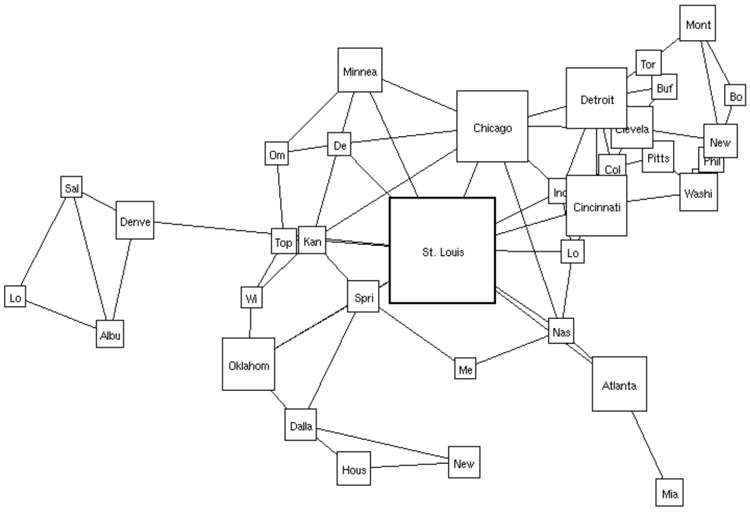


Figure 6: A fisheye view of the graph in Figure 1, with unimportant vertices eliminated. Compare this to Figure 3, with the same values of the fisheye parameters, except for the value at which unimportant vertices are eliminated. The values of the fisheye parameters are $d = 2, c = 0.5, e = 0.5, \mathit{VWcutoff} = 0.2$. The vertices have been made to appear larger than vertices in Figure 3 by increasing the *vertex-size scale factor* defined in Section 4.

of multiple straight line segments is specified by a set of intermediate *bend points*, the extreme points being the coordinates of its corresponding vertices.

The coordinates of the graph in the fisheye view are called the *fisheye coordinates*. The viewer's point of interest is called the *focus*. Each vertex in the fisheye view is defined by its position, size, and the *amount of detail* to display. Finally, each vertex in fisheye view is assigned a *visual worth*, or VW, computed based on its distance to the focus (in normal coordinates) and its *a priori* importance.

3 A Formal Framework

Generating a fisheye view involves magnifying the vertices of greater interest and correspondingly demagnifying the vertices of lower interest. In addition, the positions of all vertices and bend points must also be recomputed in order to allocate more space for the magnified portion so that the entire view still occupies the same amount of screen space.

Intuitively, the position of a vertex in the fisheye view depends on its position in the normal view and its distance from the focus. The size of a vertex in the fisheye view depends on its distance from the focus, its size in the normal view, and its API. The amount of detail displayed in a vertex in turn depends on its size in the fisheye view. We now formalize these concepts.

The position of vertex v in the fisheye view is a function of its position in normal coordinates and the position of the focus f:

$$P_{feye}(v, f) = \mathcal{F}_1(P_{norm}(v), P_{norm}(f)) \tag{1}$$

The size of a vertex in the fisheye view is a function of its size and position in normal coordinates, the position of the focus, and its API:

$$S_{feye}(v, f) = \mathcal{F}_2(S_{norm}(v), P_{norm}(v), P_{norm}(f), API(v)) \tag{2}$$

The amount of detail to be shown for a vertex depends on the vertex's size in the fisheye view and the maximum detail that can be displayed:

$$DTL_{feye}(v, f) = \mathcal{F}_3(S_{feye}(v, f), DTL_{max}(v)) \tag{3}$$

Finally, the visual worth of a vertex depends on the distance between the vertex and the focus in normal coordinates (found by examining the positions of the vertex and the focus in normal coordinates) and on the vertex's API:

$$VW(v, f) = \mathcal{F}_4(P_{norm}(v), P_{norm}(f), API(v)) \tag{4}$$

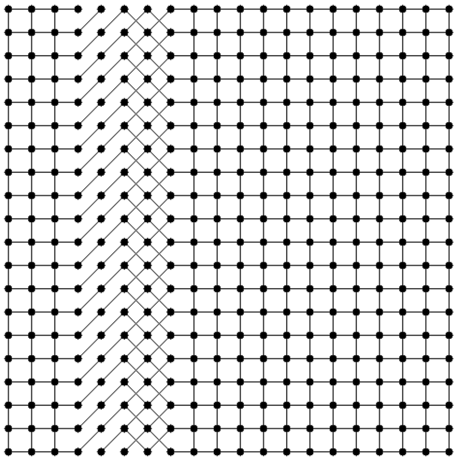


Figure 7: An undistorted nearly-symmetric graph. This graph will be a useful basis for understanding the fisheye transformations in Sections 4 and 5.

One has to choose the functions $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$, and $\mathcal{F}_4$ judiciously to generate useful fisheye views. These four functions express the relationship between a normal and fisheye view in most general terms, and are the heart of the graphical interpretation that we've introduced for fisheye views. In the next section, we present the set of functions we used in our prototype system.

4 An Implementation Strategy

We experimented with a number of alternative sets of functions for $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$, and $\mathcal{F}_4$. This section describes the set of functions that produced the most effective fisheye views of the test graphs we used. In Section 5 we discuss other implementation strategies.

Generating fisheye views is a two step process. First we apply a geometric transformation to the normal view in order to reposition vertices and magnify and demagnify areas close to and far away from the focus respectively. Second, we use the API of vertices to obtain their final size, detail, and visual worth.

4.1 Computing Position

Transforming a point P_{norm} from normal coordinates to fisheye coordinates, using focus position P_{focus}, requires us to implement the function $\mathcal{F}_1$ in Equation 1. We map the x and y coordinates independently as follows:

$$P_{feye} = \left\langle \mathcal{G}\left(\frac{D_{norm_x}}{D_{max_x}}\right) D_{max_x} + P_{focus_x}, \right. \\ \left. \mathcal{G}\left(\frac{D_{norm_y}}{D_{max_y}}\right) D_{max_y} + P_{focus_y} \right\rangle \tag{5}$$

where

$$\mathcal{G}(x) = \frac{(d+1)x}{dx+1} \tag{6}$$

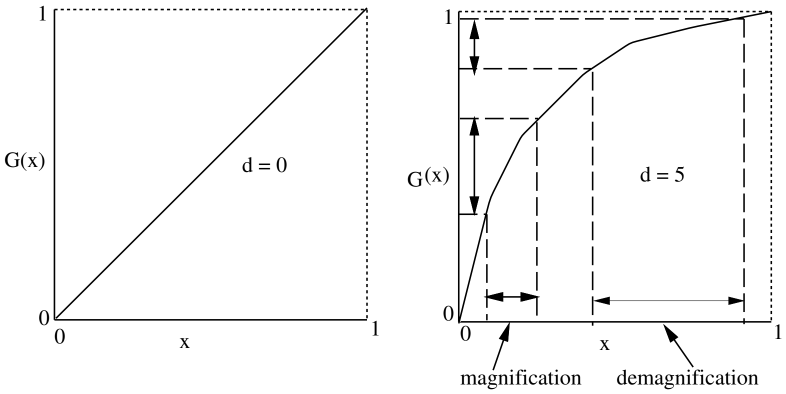
Here, D_{max_x} is the horizontal distance between the boundary of the screen and the focus in normal coordinates, and D_{norm_x} is the horizontal distance between the point being transformed and the focus, also in normal coordinates. We divide D_{norm_x} by D_{max_x} so that the argument to $\mathcal{G}$ is normalized to be between 0 and 1. Multiplying the results of $\mathcal{G}$ by D_{max_x} unnormalizes the "fisheye distance," so that adding the position of the focus (which is the same in normal and fisheye coordinates) yields the fisheye coordinates. The meanings of D_{max_y} and D_{norm_y} are similar, in the vertical dimension.

The constant d in function $\mathcal{G}$ is called the *distortion factor*. The function $\mathcal{G}(x)$ is monotonically increasing and continuous for $0 \leq x \leq 1$ with $\mathcal{G}(0) = 0$, and $\mathcal{G}(1) = 1$. The derivative of $\mathcal{G}(x)$ is

$$\mathcal{G}'(x) = \frac{d+1}{(dx+1)^2} \tag{7}$$

This indicates that for large values of d the slope of the plot of x versus $\mathcal{G}(x)$ near $x = 0$ is very large. This results in high magnification. The plot has a very small slope near $x = 1$ which causes high demagnification. A graph of $\mathcal{G}(x)$ for $d = 0$

and $d = 5$ is as follows:



When $d = 0$, the normal and the fisheye coordinates of every point are the same. In our prototype system, the user can interactively modify the value of d. The effect of altering d on the fisheye view can be seen by comparing Figure 2 to Figure 3, and Figure 7 to the columns of images in Figure 8.

We call the mapping in Equation 6 the *cartesian* transformation. In Section 7, we show a different transformation called the *polar* transformation.

4.2 Computing Size

We chose to preserve the square shape of the bounding boxes of the vertices in the fisheye view. The new size of a vertex, determined by the mapping function $\mathcal{F}_2$ in Equation 2, is implemented in two steps. The first step uses the geometric transformation just found in order to compute the geometric size $S_{geom}(v, f)$ by ignoring v's API. This mapping has the special property that if no two vertices in the normal view overlapped, no two vertices in the transformed view overlap. The second step then uses $S_{geom}(v, f)$ and v's API to complete the implementation of $\mathcal{F}_2$. However, the vertices may overlap after the second step.

The geometric size of a vertex is found by comparing the fisheye coordinates of the vertex with a point that is on the perimeter of the vertex's bounding box. To be precise, let's call the length of a side of the bounding box of the undistorted vertex S_{norm}, and introduce another parameter s, called the *vertex-size scale factor*, that the user will be able to control in our prototype system. We take a point that is $s \cdot S_{norm}/2$ away from the center of the vertex in the direction away from the

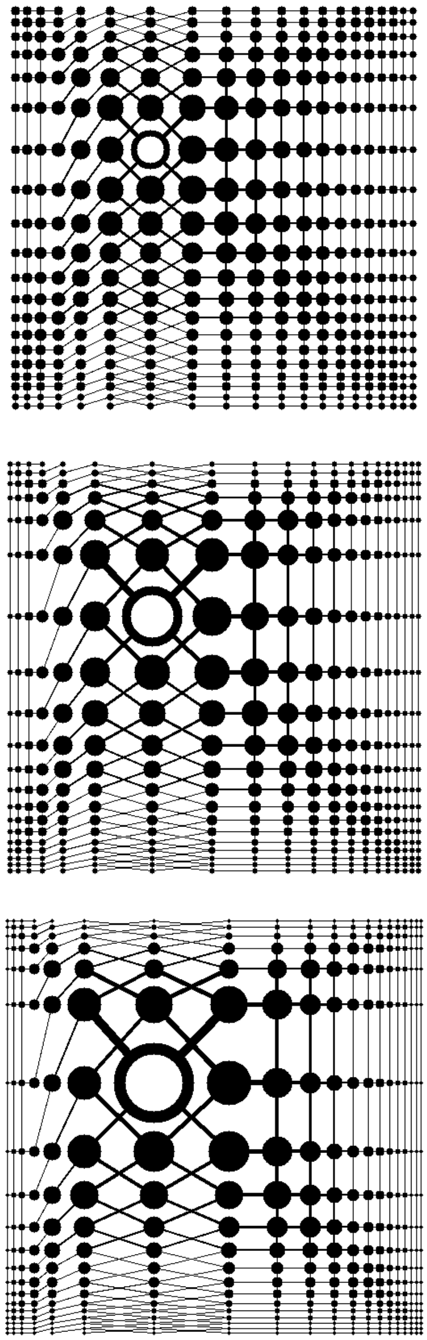


Figure 8: Fisheye views of the nearly-symmetric graph from Figure 7 using a cartesian mapping. The left column uses a focus in the northwest, and the right column uses a focus in the southeast. The distortion increases from top to bottom: In the top row $d = 1.46$, in the middle row $d = 2.92$, and in the bottom row $d = 4.38$. Note that the thickness of each edge varies with the sizes of the vertices it joins.

focus, and transform it to Q_{feye} using $\mathcal{F}_1$ in Equation 5. (Because magnification decreases as we move away from the focus, taking a point farther away from the focus rather than closer to the focus is conservative. It ensures that vertices that do not overlap in the normal view do not overlap in the fisheye view either.)

Now, the geometric size is simply

$$S_{geom} = 2 \min(\left|Q_{feye_x} - P_{feye_x}\right|, \left|Q_{feye_y} - P_{feye_y}\right|).$$

The minimum function keeps the bounding box square. The factor of 2 converts back into the length of a side.

Finally, the function $\mathcal{F}_2$ in Equation 2 is implemented by

$$S_{feye} = S_{geom}(c \cdot API)^e \tag{8}$$

where the coefficient c and exponent e are constants. In our prototype system, the user can interactively control the values of c, e, and also s. Figures 3 and 5 show the effects of varying these parameters.

4.3 Computing Detail

We chose to set the amount of detail that is displayed with a vertex in the fisheye view to be proportional to the size of the vertex in the fisheye view. However, the amount of detail for a vertex v cannot exceed the maximum available detail $DTL_{max}(v)$. Thus, function $\mathcal{F}_3$ in Equation 3 is as follows:

$$DTL_{feye}(v, f) = \min(DTL_{max}(v), \alpha S_{feye}(v, f)) \tag{9}$$

where α is a constant.

4.4 Computing Visual Worth

We chose to make visual worth of a vertex proportional to the vertex's size. Hence we implemented function $\mathcal{F}_4$ as follows:

$$VW(v, f) = \beta S_{feye}(v, f) + \gamma \tag{10}$$

where β and γ are constants.

Although our prototype system does not let the user control the values of α, β, and γ, the user can control the minimum level of visual worth that is necessary in order for a vertex to be displayed. Compare Figure 3 with Figure 6.

4.5 Mapping Edges

Straight line edges of the normal view are mapped to straight line edges in the fisheye view automatically when vertices at their end points get mapped. The edges with intermediate bend points are mapped by mapping each bend point separately. Figure 8 demonstrates the effect of cartesian transformations on a symmetric graph. Note in particular that parallelism between lines is not preserved, except for vertical and horizontal lines.

Unfortunately, mapping just the end points of edges may lead to edges that intersect in the fisheye view but not in the normal view. This artifact is quite noticeable in the border between Washington and Idaho in Figure 11. Fortunately, this problem is easily circumvented by mapping a large number of intermediate points on each straight line segment individually. Mapping many points on each edge would result in curved lines with the property that if the edges did not intersect in the normal view, the edges will not intersect in the fisheye view. However, mapping a very large a number of points may not be computationally feasible for real time response on current hardware.

As we noted, a cartesian transformation has the property that all the vertical and horizontal lines remain vertical and horizontal after the transformation. Because of this property, our transformations are especially well-suited for layouts with edges consisting of mostly horizontal and vertical line segments, for example VLSI circuits.

5 Other Implementation Strategies

Early users of our prototype system commented that transformations seemed somewhat unnatural, especially when applied to familiar objects, such as maps. Our framework allows us to address this complaint by using domain-specific transformations.

Consider for instance, the non-fisheye view of a map of the United States shown in Figure 9 and a corresponding fisheye view in Figure 10. A more natural fisheye view of such a map might be to distort the map onto a hemisphere, as is done in Figure 11. To do so, we developed a transformation based on the polar coordinate system with the origin at the focus. In this transformation, a point with normal coordinates (r_{norm}, θ) is mapped to the fisheye coordinates (r_{feye}, θ) where

$$r_{feye} = r_{max} \frac{(d+1)\frac{r_{norm}}{r_{max}}}{d\frac{r_{norm}}{r_{max}} + 1} \tag{11}$$

Here, r_{max} is the maximum possible value of r in the same direction as θ. Note that θ remains unchanged by this mapping. Figure 12 shows the polar transformations on the nearly-symmetric graph from Figure 7. It is instructive to compare these mappings with the cartesian transformations of the same nearly-symmetric graph in Figure 8.

Kaugars and Brazma describe a graphical fisheye viewer of graphs that uses the arc tangent function to implement the function $\mathcal{F}_1$ of our framework [9]. Their system always moves the focus to the center of the screen, whereas our prototype preserves the position of the focus. Studies by Hollands, Carey, Matthews, and McCann indicate that preserving focus position is important to help maintain the user's orientation [6].

Another factor contributing to the perceived unnaturalness of the fisheye view is that the shapes of vertices remain undistorted and edges remain straight lines (ignoring bend points). We could remedy this by mapping many points on the outline of the vertex, and mapping a large number of intermediate points for the edges, thus allowing the vertices and edges to become curved. However, in our prototype system, we chose not to do so, in order to achieve real time performance.

Figure 9: An outline of the United States

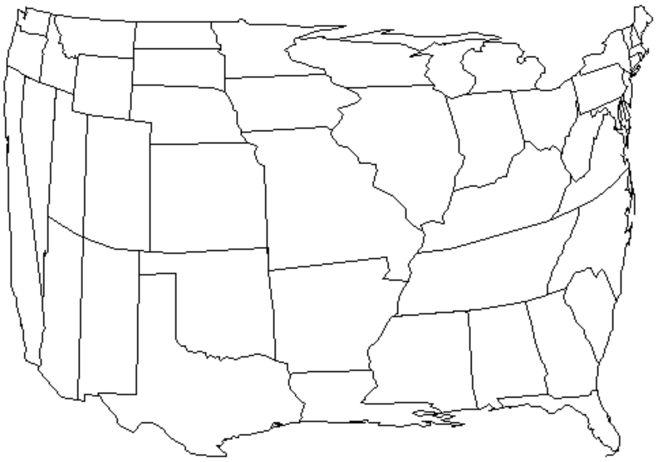


Figure 10: A cartesian transformation of Figure 9. The focus is at the point where Missouri, Kentucky, and Tennessee meet.

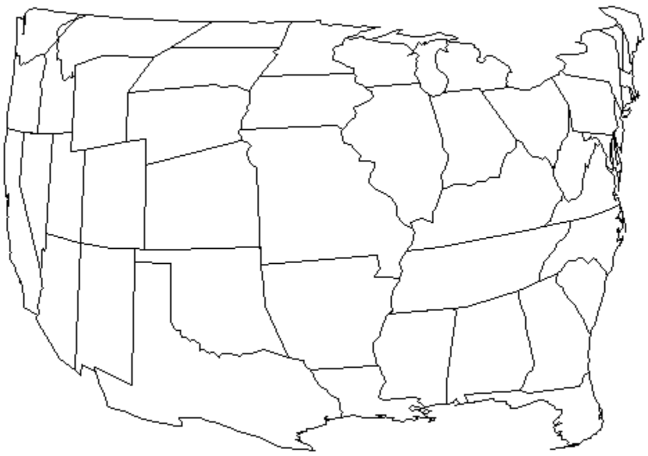


Figure 11: A polar transformation of Figure 9. As in Figure 10, the focus is at the point where Missouri, Kentucky, and Tennessee meet. Notice the infelicity in northern Idaho. The crossing lines result from the fact that the database represents the western edge of Idaho as a single segment along the state of Washington; the eastern edge comprises many small segments. This problem would go away if our system mapped every point in each edge, or had the database represented the western edge of Idaho by multiple small (and colinear) segments. See Section 4.5 for more details.

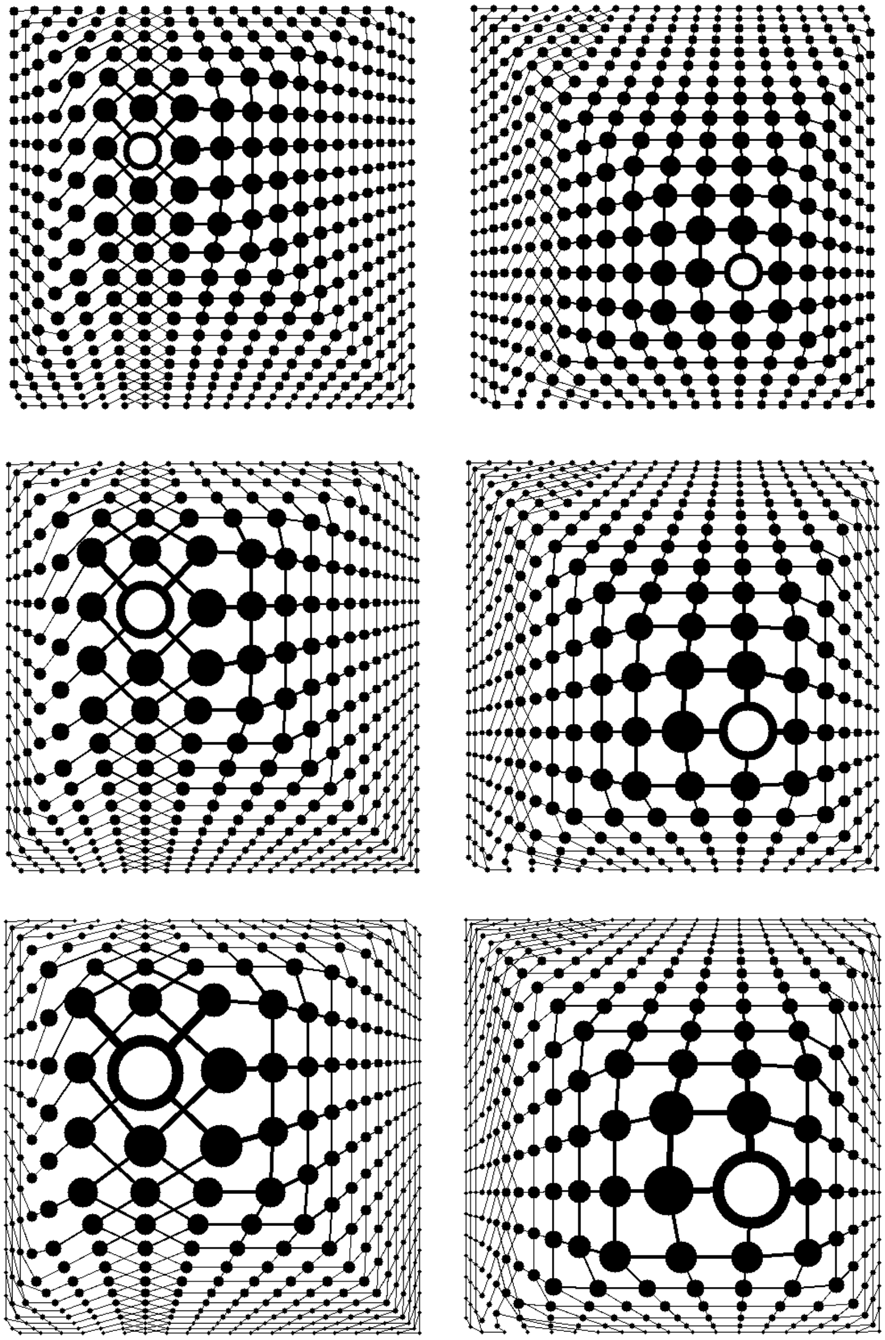


Figure 12: Fisheye views of the nearly-symmetric graph from Figure 7 using a polar mapping. As in Figure 8, the left column uses a focus in the northwest, and the right column uses a focus in the southeast. The distortion increases from top to bottom: In the top row $d = 1.46$, in the middle row $d = 2.92$, and in the bottom row $d = 4.38$. The thickness of each edge varies with the sizes of the vertices it joins.

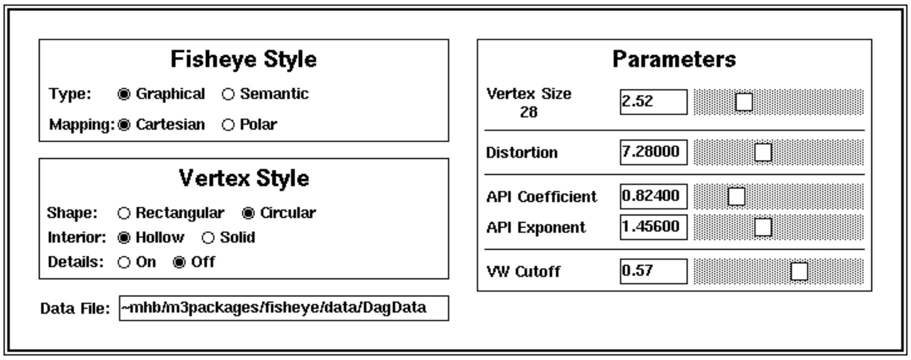


Figure 13: The control panel of our prototype system.

6 The Prototype System

Our system displays a fisheye view of a user-specified graph, and updates the display in real time as the user moves the focus by dragging with the mouse. The graph is displayed in one top-level window and the control panel, shown in Figure 13, is displayed in another top-level window. The control panel has sliders and numeric type-in field that allow the user to control of the value of the distortion factor d in Equation 6, the coefficient c, exponent e, and vertex scaling factor s in Equation 8, and a cutoff point at which vertices and their incident edges should no longer be displayed. The coefficient c, the exponent e, and the vertex scaling factor s control the effect of the API of the vertices on the non-geometric part of the transformation, while d affects the geometric part of the transformation. The combined effect of these parameters on the graph in Figure 1 is illustrated in Figures 2– 6.

The prototype is implemented in an event-driven style. Each time the user moves the mouse while the button is held down, the system computes the new focus position. The position, size, and detail of each vertex are then computed, and the bendpoints for the non-straight lines are mapped. Using the minimum and the maximum vertex size, the system computes the visual worth of the vertices in a scale from 0 to 1. Next, for each edge, if the visual worth of both of its end vertices are greater than or equal to the cut off point, then the edge is painted on the screen. The vertices are then sorted in the non-decreasing order of their visual worth, and those vertices whose visual worth is greater than the cut off point are then painted in the non-decreasing order just computed.

The prototype system is able to maintain real time response on a DECstation 5000 for graphs of up to about 150 vertices and about 350 edges, such as that shown in Figure 2. Computing the fisheye transformations takes an insignificant amount of time compared to the time required for painting the graph.

7 Logical Fisheye Views

Our work follows from the *generalized* fisheye views by Furnas [5]. We will call this type of fisheye views as *logical* fisheye views as the latter seems more appropriate to us. Furnas gave many compelling arguments describing the advantages of fisheye views. The essence of Furnas's formalism is the "degree of interest" function for an "object" relative to the "focal point" in some "structure." Our notion of "visual worth" (see Equation 4) is nearly identical to Furnas's degree of interest. The difference is that we have (thus far) described distance as the Euclidean distance separating two vertices in a graph, whereas Furnas defined the distance function as an arbitrary function between two objects in a structure. Our framework supports logical fisheye views by recoding the distance function used explicitly in Equation 4 and implicitly by Equations 1–3.

For instance, consider the graph in Figure 14 and the graphical fisheye view of it in Figure 15. The distance between vertices is their Euclidean distance. A vertex is displayed only if its visual worth is above some threshold, and its position, size, and level detail are computed using Equations 1, 2, and 3, respectively. A "logical" fisheye view of that same graph, with the same focus, is shown in Figure 16. Here, the API is as before, but the distance function not geometrical; it is the length of the shortest path between a vertex and the vertex defining the focus, as proposed by Furnas [5]. Notice that in the logical fisheye view, each vertex is either displayed or omitted; there is no explicit way to vary size and level of detail. This causes both space and time discontinuity in the information space, vertices appear and disappear from the view as the viewer changes focus. In general, discontinuities disturb the orientation of the viewer, and make navigation more difficult [10].

Recently Koike proposed a type of logical fisheye view, called a "Generalized Fractal View," that keeps the total number of nodes constant, and this constant can be set by the user [7]. The utility of this method has not been established yet.

8 Other Related Work

Furnas cites a 1973 doctoral thesis by William Farrand [4] as one of the earliest uses of fisheye views of information on a computer screen. The thesis suggests transformations similar to our cartesian and polar transformations, but provides few details.

In 1982, Spence and Apperley [13] developed a technique called the Bifocal Display for visualizing office information. Briefly, in their system the workspace is a set of information items positioned in a horizontal strip. The display is a combination of a detailed view of the strip and two distorted views, where items on either side of the detailed view are squashed horizontally into narrow vertical strips. The Bifocal Display is the conceptual ancestor of the Perspective Wall system described below.

At CHI '91, Card, Mackinlay, and Robertson presented two views of structured information that have fisheye properties. The *perspective wall* [10] maps a large linear information base into a 3-dimensional visualization. The center panel shows detail, while the two side panels, receding in the distance, show the context. The *cone tree* [11] displays a tree in 3-D with each node the apex of a cone, and the children of the node positioned around the rim of the cone. Nodes close to the synthetic camera appears larger than those nodes far away from the synthetic camera, thereby providing a fisheye effect. The tree is also rendered with shadows, and transparency, and the tree can be rotated to bring nodes of interested to the front of the view with smooth animated movement. Cone tree displays however look cluttered as the number of nodes go beyond a couple of hundred, so other, specialized, views of large trees may be more appropriate [8].

Also for viewing large amounts of data, it may be fruitful to combine fisheye views with hierarchical abstraction. For example, related nodes in a graph can be combined to form cluster nodes, and the member nodes of a cluster node can be thought of as the detail of the cluster node, as done in the SemNet system [3]. Our framework of Section 3 allows one to combine fisheye views with hierarchical abstraction simply using a more sophisticated $\mathcal{F}_3$ function than we used.

Indeed, a recent experiment by Schaffer et. al. compared fisheye views to traditional zooming on a hierarchically clustered network [12]. Subjects were given a problem that required them to navigate a network and to reroute some edges. The study found that using fisheye views significantly improved a user's performance. Users completed the task faster and with fewer unnecessary traversals in the structure.

We are aware of one formal study that investigated the efficacy of using graphical fisheye views (without hierarchy and clustering) rather than scrolling a zoomed image was done by Hollands, Carey, Matthews, and McCann [6]. Subjects located stations, selected optimal routes, and constructed optimal itinerary between stations on some imaginary subway network. The results of the study favors fisheye views only slightly. The fisheye viewer used in the study moved the focal point to the center in abrupt motion causing disorientation. We believe that repeating the experiment using our prototype system would favor fisheye views more significantly for two reasons. First, our prototype eliminates the disorientation problem by preserving the focus position and by updating the view in real time with smooth transition between successive views. Second, our system provides the user with more flexibility for controlling the view. For example, the viewer can control the distortion factor, the vertex-size scale factor, contribution of the API towards the visual worth, and the visual worth cut off point.

9 Summary

We feel that the graphical fisheye views is a promising technique for viewing large structures. Our major contribution is to introduce layout considerations into the fisheye formalism. This includes the position of items, as well as the size and level of detail displayed. These are computed as a function of an object's distance from the focus and the object's preassigned importance in the global structure. As we pointed out, our framework encompasses logical fisheye views of arbitrary structures (by changing the interpretation of "distance").

It is important to realize that we do not claim that a fisheye view is *the* correct way to display and explore a graph or any other structure. Rather, it is one of the many ways that are possible. Discovering and quantifying the strengths and weaknesses of fisheye views are challenges for the future.

Acknowledgments

Jorge Stolfi helped with various ideas concerning geometric transformations. Steve Glassman, Bill Kalsow, Mark Manasse, Eric Muller, and Greg Nelson extricated us from numerous Modula-3 and Trestle entanglements. Mike Burrows and Lucille Glassman helped to improve the clarity of this presentation. Finally, George Furnas provided us with a wealth of information that improved many aspects of

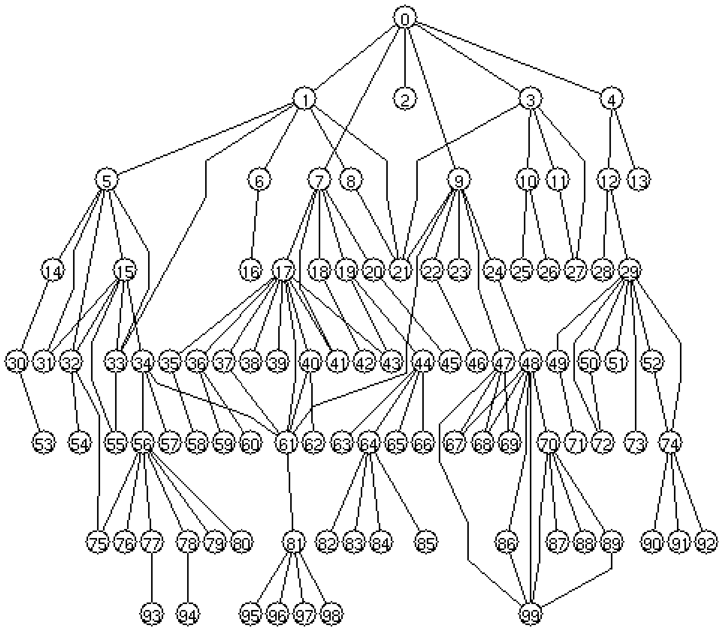


Figure 14: A graph with 100 vertices and 124 edges. All edges point downwards. The API of each vertex is related to its display level (e.g., the root has the highest API of 8, node 33 has an API of 4, and node 86 has an API of 2).

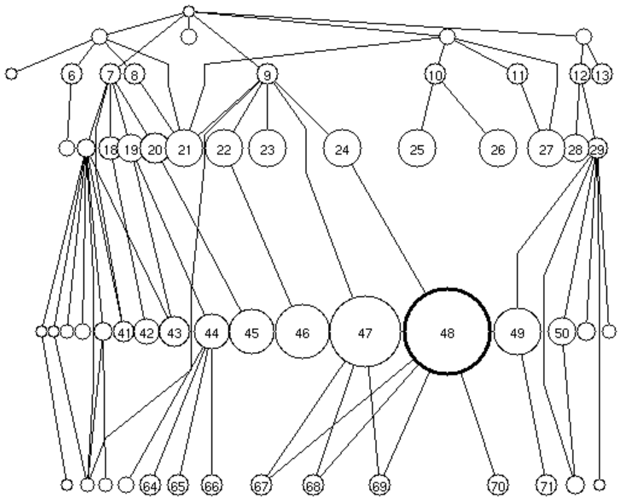


Figure 15: A graphical fisheye view of Figure 14. The focus is the vertex labeled 48.

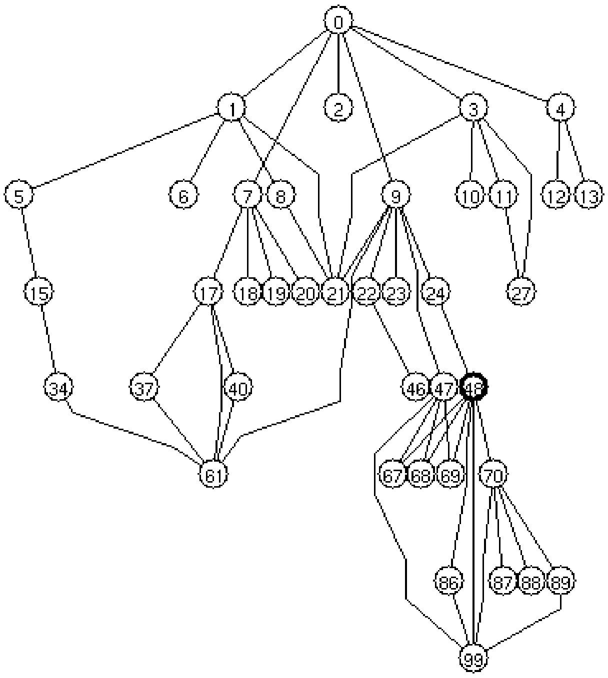


Figure 16: A logical fisheye view of Figure 14. The focus is the vertex labeled 48.

our prototype system and also of this paper.

System and Videotape Availability

The prototype system described in this paper is implemented in Modula-3 and is available via anonymous ftp from `gatekeeper.dec.com`. It is located in the directory `pub/DEC/Modula-3/release/vbtapps`.

You can find out information about Modula-3 in the Usenet news group `comp.lang.modula3`. If you do not have access to Usenet, you can be added to a relay mailing list by sending a message to `m3-request@src.dec.com`.

A videotape showing the prototype system in action is available upon request. Send electronic mail to `m3-request@src.dec.com`, and request SRC Report #84b. Be sure to specify what format VHS you prefer, NTSC, PAL, or SECAM.

References

[1] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The Information Visualizer, an information workspace. In *Proc. ACM SIGCHI '91 Conf. on Human Factors in Computing Systems*, pages 189–194, April 1991.

[2] Peter Eades and Roberto Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical Report CS–89–90, *Department of Computer Science*, Brown University, Providence, RI, 1989.

[3] Kim M. Fairchild, Steven E. Poltrok, and George W. Furnas. SemNet: Three-dimensional graphic representations of large knowledge bases. In *Cognitive Science and Its Applications for Human Computer Interaction*, pages 201–233, 1988.

[4] William Augustus Farrand. Information display in interactive design. Ph.D. Thesis, *Department of Engineering*, UCLA, Los Angeles, CA, 1973.

[5] George W. Furnas. Generalized fisheye views. In *Proc. ACM SIGCHI '86 Conf. on Human Factors in Computing Systems*, pages 16–23, 1986.

[6] J. G. Hollands, T. T. Carey, M. L. Matthews, and C. A. McCann. Presenting a Graphical Network: A Comparison of Performance Using Fisheye and

Scrolling Views. In *Proc. of 3rd Int'l Conf. on Human-Computer Interaction*, pages 313–320, September, 1989.

[7] Hideki Koike, "Generalized Fractal Views," In *Proc. of Advanced Visual Interfaces*, 1992.

[8] Bob Johnson and Ben Schneiderman. Space-filling approach to the visualization of hierarchical information structures. In *Proc. IEEE Visualization '91*, pages 284–291, 1991.

[9] Karlis Kaugars, and Alvis Brazma. CATGraph: Visualizing Large Labeled Graphs. Technical Report 92-CS-08, *Computer Science Department*, New Mexico State University. April, 1992.

[10] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: Detail and context smoothly integrated. In *Proc. ACM SIGCHI '91 Conf. on Human Factors in Computing Systems*, pages 173–179, April 1991.

[11] George G. Robertson, Jock D. Mackinlay, and Stuart K. 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.

[12] Schaffer et. al. Comparing fisheye and full zoom techniques for navigation of hierarchically clustered networks. In *Proc. of Graphics Interface*. 1993.

[13] Robert Spence and Mark Apperley. Database navigation: An office environment for the professional. *Behavior and Information Technology*, vol. 1, no. 1, pp. 43–54, 1982.