Fisheye Views of 2D Graphs and 3D Objects

Jörg Keller
(FernUniversität–GH Hagen, Germany
Joerg.Keller@FernUni-Hagen.de)

Leiming Zhu
(University of Science and Technology of China
oliverzhu@btamail.net.cn)

Abstract: We investigate the computation and implementation of fisheye views of 2D graphs and 3D objects. Fisheye views display a graphics just like we see it through a wide angle fisheye lens. They provide new ways to display the graphics such that one part of the graphics is enlarged while a view of the whole graphics is still maintained. In a fisheye view of a 2D graph, a fisheye focus point or a focus polygon is defined and the necessary computations are provided to transform every node in the 2D graph with respect to its euclidean distance from the focus point or the focus polygon. In a fisheye view of a 3D object, a fisheye focus point and a focus direction are defined and the computations are discussed to transform every vertex in the 3D object with respect to its relative position to the focus point and the focus direction. The fisheye views can be displayed based on the transformed nodes and vertices. We have implemented fisheye views to 2D graphs and 3D objects because 2D graphs and 3D objects are typical 2D and 3D graphics. Our implementation allows to change the fisheye view parameters interactively.

Key Words: fisheye view, graph, solid object, java, VRML
Category: I.3

1 Introduction

A Fisheye view, as the name suggests, is a particular type of view to an image. Its visual effect is similar to a view produced by a very wide angle fisheye lens. The fisheye lens distorts the view so that objects close to the centre of the lens are magnified greatly. This magnification drops rapidly to the objects farther from the centre of the lens. This view results in objects which are the centre of attention being shown in great detail, whereas objects on the periphery are shown in lesser detail. This allows objects of interest to be studied in detail, while still maintaining a view of the context to other objects.

Fisheye views can be used in data visualizations and nonlinear magnification of graphics. They provide special distorted displays of 2D or 3D graphics. Compared to the normal view of the graphics, the fisheye view has the advantage of zooming part of the graphics and overviewing the whole graphics at the same time.

In fisheye view of 2D graphics, a fisheye focus point is defined and every coordinate point in the 2D graphics is transformed with respect to its euclidean distance from the focus point [see Sarkar and Brown 1994]. Then the fisheye view can be displayed based on the transformed coordinate points. Often one is interested in details of a particular region of the graphics. This demand can be accomplished by defining a focus region instead of a focus point. The focus region can be specified by a simple polygon [see Formella and Keller 1995].

We extend the notion of fisheye views to 3D graphics. Here a fisheye focus point and a focus direction are defined and every coordinate point in the 3D graphics is transformed with respect to its relative position to the focus point and the focus direction. And the fisheye view can be displayed based on the transformed coordinate points.

The remainder of this paper is arranged as follows:

In section 2, we discuss fisheye views of graphs layouted in 2D. The data structure of graphs is presented. The fisheye focus point and the focus polygon are defined. Then the computations of fisheye views to 2D graphs are derived.

In section 3, we discuss fisheye views of 3D objects. The data structure of solid objects is introduced. The fisheye focus point and focus direction are defined. Then the computations of fisheye views to 3D objects are derived.

In section 4, we discuss a implementation of fisheye views. Java is used to implement the fisheye views of 2D graphs and 3D objects. We can interactively change the fisheye focus point, the focus region and the focus direction. We can display the normal views and fisheye views of 2D graphs and 3D objects respectively.

In section 5, we give a conclusion.

2 Fisheye Views of 2D Graphs

Graphs layouted in 2D are widely used in CAD systems, electronic maps and other application fields. Here we are concerned about how to apply fisheye views to the graphs.

First we represent a graph with a data structure. We briefly review fisheye views based on a focus point [see Sarkar and Brown 1993] because they form the basis of our work. Then we derive the computations for the fisheye views of the graph based on a focus polygon, which are partly given in [Formella and Keller 1995].

2.1 The Data Structure of 2D Graph

A layout graph is a group of nodes and edges which connect the nodes. Each node is represented by a coordinate point. Each edge is represented by two node indices. The data structure of a graph can be represented as follows:

$$\begin{aligned}&\textit{2D Graph} = \{\textit{NodeList},\textit{EdgeList}\}\\&\textit{NodeList} = \{N_1,N_2,\ldots,N_n\}\\&N_1=(x_1,y_1),\ldots,N_n=(x_n,y_n)\\&\textit{EdgeList} = \{\{v_1,w_1\},\{v_2,w_2\},\ldots,\{v_m,w_m\}\}\\&v_i,w_i \in \{1,2,\ldots,n\}\end{aligned}$$

To provide a fisheye view to the graph, we must transform each node position in the graph based on a fisheye focus point and then use the transformed nodes to display the distorted graph. The graph is displayed by using squares for nodes and straight lines for edges. Edges remain straight lines in the transformed graph due to our representation. It is however possible to partition longer edges into line segments by introducing pseudopoints which are transformed as well. The transformed edges then will have a curved shape.

2.2 Computation of Fisheye Views Based on a Focus Point

To provide a fisheye view to a graph based on a focus point, we should restrict the fisheye view transformation to a finite area, i.e. we should define a bounding frame for the fisheye view transformation. Each node of the graph is moved away from the focus point up to the frame. The closer a node is to the focus point the farther away should it be moved.

We define the frame to be the smallest axis parallel rectangle containing all the nodes of the graph. We also define a focus point to be a coordinate point within the frame. Note that the frame could very well be defined as a circle or other convex, simple closed curve (at least for polar transformation). The frame could also be larger or smaller than the graph. In the latter case, we restrict the transformation to a part of the graph.

Let the focus point be f and the node be n. Two types of transformations are possible: polar and cartesian transformation. The different applications of both types are discussed in [Sarkar and Brown 1994].

2.2.1 Polar Transformation

As shown in Figure 1(a), we connect a line which start from f and go through n and intersect with the frame at i. The node n can be represented by the focus point f and the vector $(i - f)$:

$$n = f + A(i - f) \qquad A \in [0:1]$$

A fisheye view of a graph moves the node to n' along the line such that the node n is moved away from the focus point f but does not leave the frame.

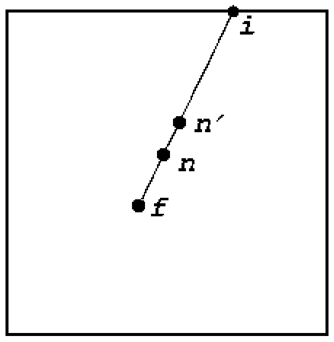
Thus, the moved node n' can also be represented by the focus point f and the vector $(i - f)$:

$$n' = f + h(A)(i - f)$$

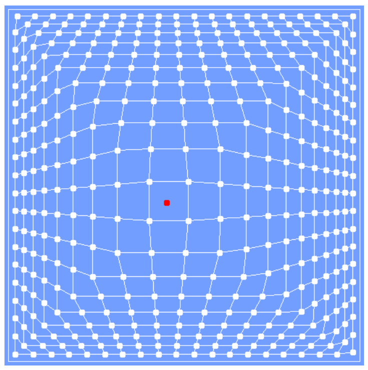
where $h : [0:1] \rightarrow [0:1]$ is a bijective mapping fuction.

According to the fisheye view characteristic, the function h should satisfy the following requirements:

$h(0) = 0$: The focus point itself is not moved.
$h(1) = 1$: Points on the frame are not moved.
$h(x) \geq x$ for all x in $(0:1)$: All other points are moved away from f.
$h(x)$ be strictly monotonous increasing: Points cannot overtake during the transformation, i.e. points being closer to f before the transformation must be closer afterwards as well.
$(h(x) - x)/x$ be strictly monotonous decreasing: The closer a point is to f, the farther away should it be moved. The moving distance is taken relative to the point's distance of f before the transformation.



(a) Definition



(b) Sample fisheye view of a grid

Figure 1: Polar transformation of a fisheye view based on a focus point

$h(x) = (d+1)x/(dx+1)$ is a possible solution, where $d \geq 0$ controls the amount of movement [see Sarkar and Brown 1994].

To parameterize the distortion as a percentage $p \in [0:1)$, we define:

$$d = \frac{p}{1-p}$$

See Figure 1(b) for a sample fisheye view of a grid. It is transformed with $p = 0.6, d = 1.5$, the nodes are filled squares and the edges are straight lines between nodes. The focus point is shown in black.

2.2.2 Cartesian Transformation

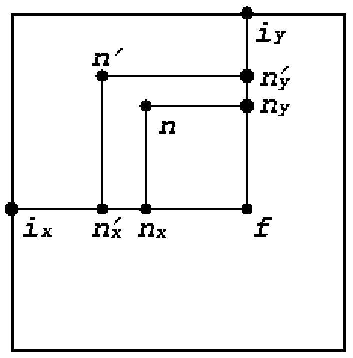
Assume the coordinate of $f = (f_x, f_y)$ and the coordinate of $n = (n_x, n_y)$. We apply the fisheye view distortion along horizontal and vertical direction. We project n onto a horizontal line through f, and intersect the line with the frame at $i_1 = (i_x, f_y)$. We proceed similarly with a vertical line through f and obtain an intersection point at $i_2 = (f_x, i_y)$ as shown in Figure 2(a).

The coordinates of the transformed node $n' = (n'_x, n'_y)$ are computed by:

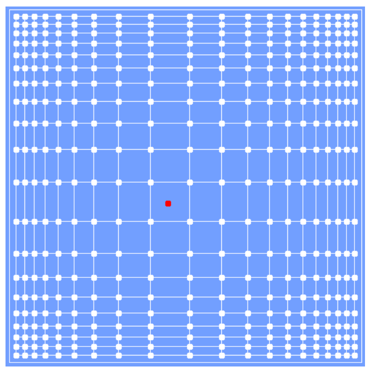
$$n_x = f_x + B_x(i_1 - f_x); \quad n'_x = f_x + h(B_x)(i_1 - f_x)$$

$$n_y = f_y + B_y(i_2 - f_y); \quad n'_y = f_y + h(B_y)(i_2 - f_y)$$

See Figure 2(b) for a sample fisheye view of a grid. It is transformed with $p = 0.6, d = 1.5$, the nodes are filled squares and the edges are straight lines between nodes. The focus point is shown in black.



(a) Definition



(b) Sample fisheye view of a grid

Figure 2: Cartesian transformation of a fisheye view based on a focus point

2.3 Computation of Fisheye views based on a focus polygon

To define transformations based on a focus polygon P, we will first deal with nodes that are positioned within the polygon. To derive transformations for nodes outside the polygon, we will define focus points on the polygon and apply the techniques from the previous section. For both transformations to be developed, the case of a focus point is obtained if the polygon consists of only one point.

2.3.1 Scaling the Polygon

The focus region itself should be enlarged, but it should not be deformed, because it is the center of our interest. Therefore, we scale the focus region. To do this, we define a center point c of P by barycentric combination of the polygon corners and compute a scaling factor S. Then any node n that is within the polygon can be represented as:

$$n = c + (n - c)$$

And will be moved to:

$$n' = c + S(n - c)$$

For the polygon P with n corners $p_1, ...p_n$, we define the center point c as follows:

$$c = \frac{1}{n} \cdot \sum_{i=1}^{n} p_i$$

The scaling factor must guarantee that the scaled polygon still lies whihin the frame. For each corner $p_i \neq c$ of polygon P, we intersect a line starting from c, through p_i with the frame at i_i. We get the maximum scaling factor for each corner point:

$$S_i = \frac{i_i - c}{p_i - c}$$

So the maximum scaling factor for the polygon P is as follows:

$$S_{max} = \min\{S_i, 1 \leq i \leq n\}$$

We must choose a scaling factor $S \geq 1$ which is less than S_{max}.
To parameterize the scaling as a percentage $p \in [0:1)$, we define:

$$S = 1 + p \cdot (S_{max} - 1)$$

In the case of a single point we define:

$$S = S_{max} = 0$$

2.3.2 Polar Transformation

For each node n outside P, we define its focus point f to be the point on P with minumum distance to n. This might either be a corner of P or the intersection point of one edge of P and the perpendicular line through n.

The voronoi diagram $VD(P)$ of a simple polygon P partitions the plane outside the polygon in regions such that all points in one region are closer to one edge or corner of the polygon than to all other edges and corners. We can make all regions finite by intersecting $VD(P)$ with the frame. We denote by $VR(n,P)$ the outline of the enclosed voronoi region that contains point n as shown in Figure 3(a).

We intersect a line L starting from f, through n with $VR(n,P)$ at point i. Then

$$n = f + A(i - f) \qquad A \in [0:1]$$

When the polygon P is scaled to polygon P' the focus point f is scaled to f'. We intersect a line starting from f', parallel to previous line L with $VR(f',P')$ at point i'. Then the moved node n' is defined as:

$$n' = f' + h(A)(i' - f')$$

Here we must choose h such that the behaviour at the border of the polygon is in some sense continuous, i.e. $\frac{h(A)}{A}$ is close to S when A is close to zero:

$$\lim_{x \to 0} \frac{h(x)}{x} = S$$

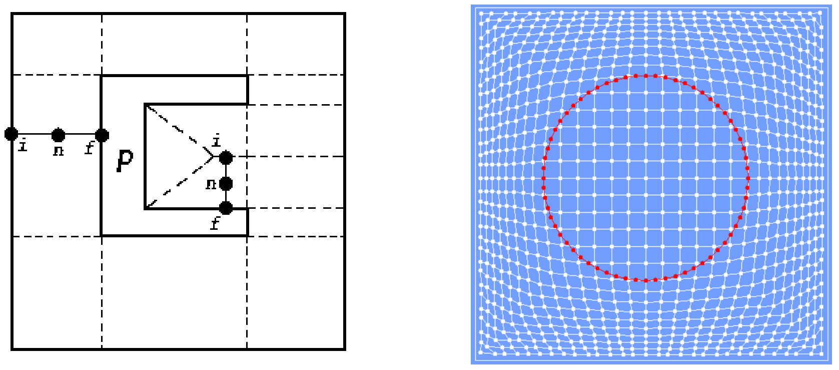
As $h(0) = 0$, we can derive:

$$\lim_{x \to 0} \frac{h(x)}{x} = \lim_{x \to 0} \frac{h(x) - h(0)}{x - 0} = h'(0) = S$$

Because the first derivative of h is $h'(x) = \frac{(d+1)}{(dx+1)^2}$, we obtain $h'(0) = d + 1$. So we choose

$$d = S - 1$$

See Figure 3(b) for a sample fisheye view of a grid. It is transformed with $p = 0.6, S = 1.5, d = 0.5$, the polygon is a circle formed with 64 corners.



(a) Definition (b) Sample fisheye view of a grid

Figure 3: Polar transformation of a fisheye view based on a focus polygon

2.3.3 Cartesian Transformation

For each node $n = (n_x, n_y)$ that is positioned outside P, we first find its horizontal focus point and its vertical focus point on the polygon.

Here we only discuss y dimension. The x dimension is handled similarly. As shown in Figure 4(a), we use a vertical line $x = n_x$ to intersect polygon P. If it does not intersect with P, we choose the closest line $x = c_x$ that intersects with P and project n onto this line. On line $x = n_x$ or $x = c_x$, we choose the intersection point closest to n as the focus point $f = (x, f_y)$.

If P is convex the line intersects with P in at most two points and n is located between its focus point and the frame. We intersect the vertical line starting from f with the frame at i_y. We get:

$$n_y = f_y + B_y(i_y - f_y) \qquad B_y \in [0, 1]$$

If P is not convex the number of intersection points is a multiple of two. If n is above or below all of the intersection points we obtain i_y as before. If n is between two intersection points $j = (x, j_y)$ and $k = (x, k_y)$, we choose the middle point as the end point of fisheye view, i.e. we define $i_y = (j_y + k_y)/2$. We still get:

$$n_y = f_y + B_y(i_y - f_y)$$

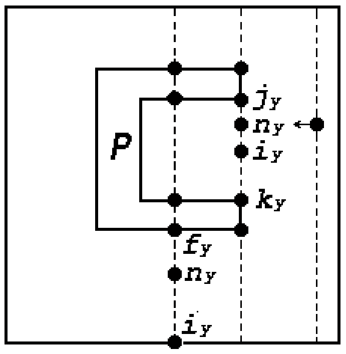
When the polygon P is scaled to polygon P' the focus point f is scaled to f'. If n is not between intersection points i and j, the end point is on the frame and $i'_y = i_y$ is not changed. If n is between intersection points i and j, i and j are scaled to i' and j' and $i'_y = (j'_y + k'_y)/2$.

Let the moved node be $n' = (n'_x, n'_y)$. We define:

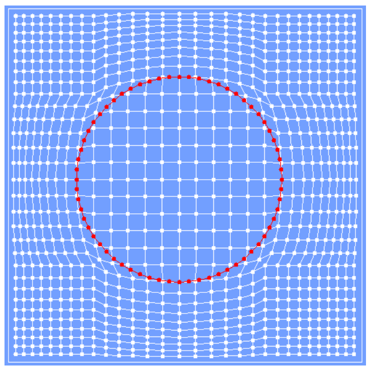
$$n'_y = f'_y + h(B_y)(i'_y - f'_y)$$

Similarly, we compute:

$$n_x = f_x + B_x(i_x - f_x)$$



(a) Definition



(b) Sample fisheye view of a grid

Figure 4: Cartesian transformation of a fisheye view based on a focus polygon

$$n'_x = f'_x + h(B_x)(i'_x - f'_x)$$

See Figure 4(b) for a sample fisheye view of a grid. It is transformed with p=0.6, S=1.5, d=0.5, the polygon is a circle formed with 64 corners.

3 Fisheye Views of 3D objects

Now we consider how to apply fisheye view to 3D objects. 3D graphics is more complicated than 2D graphics. In 2D graphics the basic elements are points and line segments. But in 3D graphics the basic elements are points and little faces. Solid objects are composed of vertices and outside faces. They are building blocks of 3D graphics. They are widely used in 3D games, virtual reality systems and other application fields. So we are concerned about how to apply fisheye views to the 3D objects. First we should represent an object by a data structure. Then we derive the computations for the fisheye views of the objects based on a focus point and a focus direction.

Why should we define a focus direction here in fisheye view of 3D graphics? To apply a fisheye view is just like we look forward through a fisheye lens, the object in front of the lens is zoomed. In the 2D plane we always look at objects with the viewing direction perpendicular to the plane. So the focus direction need not to be defined. But In 3D space we always need to look at objects from different viewpoints and different viewing directions. So it is necessary to define a focus direction to apply fisheye views to 3D graphics.

3.1 The Data Structure of 3D object

An object is a group of vertices and facets. Each vertex is represented by a 3D coordinate point. Each facet is represented by a polygon or simply a triangle. So each facet can be represented by a list of three vertex indices. The data structure of an object can be represented as follows:

$\mathit{3D\ object} = \{\mathit{VertexList},\mathit{FacetList}\}$
$\mathit{VertexList} = \{V_1,V_2,\ldots,V_n\}$
$V_1=(x_1,y_1,z_1),\ldots,V_n=(x_n,y_n,z_n)$
$\mathit{FacetList} = \{\{u_1,v_1,w_1\},\{u_2,v_2,w_2\}\ ,\ldots,\{u_m,v_m,w_m\}\}$
$u_i, v_i,w_i \in \{1,2,\ldots,n\}$

To provide a fisheye view to the solid object, we must transform each vertex in the object based on the fisheye focus point and the focus direction and then use the transformed vertices and facets to display the distorted object.

3.2 Computation of Fisheye Views Based on a Focus Point and a focus Direction

We define a focus point which is a 3D coordinate point outside the object. We also define a focus direction which is a vector starting from the focus point.

To provide a fisheye view to the object based on the focus point and focus direction, each vertex in the object is moved towards the focus point along the focus direction. The closer a vertex is to the focus point the farther should it be moved. Vertices which are closer to the focus point before the transformation must be closer afterwards as well.

Let the focus point be $f = (f_x, f_y, f_z)$, the vertex be $v = (v_x, v_y, v_z)$ and the fisheye view moved vertex be $v' = (v'_x, v'_y, v'_z)$. We derive our computation for the special case that the focus direction is parallel to the z axis. For an arbitrary focus directions, we apply 3D transformations before and after the fisheye view computation such that the general case is reduced to the special case.

We define a frame which is an axis parallel cube containing all the vertices and facets of the object. For the moved vertex $v' = (v'_x, v'_y, v'_z)$, we first discuss how to compute v'_z. Suppose we connect a line along the focus direction which start from f and go through v(in z dimension) and intersect with the frame at $i = (f_x, f_y, i_z)$. Because the vertex v is to be moved towards f we compute:

$$v_z = i_z + C(f_z - i_z); \quad v'_z = i_z + h(C)(f_z - i_z)$$

For the computation of v'_x and v'_y, we project the vertex and the bounding cube to the xy plane. We can use the same computation techniques that we have derived for 2D graphs.

See Figure 5 for a sample normal view and a sample fisheye view of a cylinder. They are rendered by a VRML browser. The fisheye view is transformed with $p = 0.6, d = 1.5$.

4 Implementation in Java

So far we have discussed how to apply fisheye views to 2D graphs and 3D objects. Now we discuss a software implementation of fisheye views. We choose java for our implementation because the java program support windowing and networking on all major OS platforms without specialized tailoring or even recompilation and java is also simple, object-oriented and extendible.

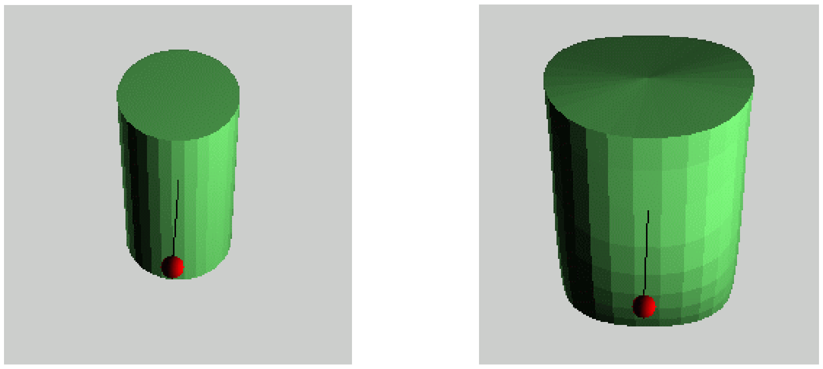

(a) Normal view (b) Fisheye view

Figure 5: Sample views of a cylinder with a focus point and a focus direction

4.1 Software Design

The objective of the program is to develop a software tool to display normal view and fisheye view of 2D graphs and 3D objects interactively. With this objective in mind, We want to develop a java application which has a GUI and a menu system to manipulate the fisheye views interactively.

For 2D graphs, we use windows to display a normal view and a fisheye view of the graph. The size of the window can be changed and the displayed graphics is scaled interactively with the mouse .

For 3D objects, we first convert the object data into VRML files and then run a VRML browser to display the normal view and fisheye view of the 3D objects. The menu system should include the following commands:

- Load the graph data files and the object data files.
- Load the focus point, focus polygon, focus direction.
- Save the fisheye view transformed graph data files and object data files.
- Open a dialog box to change the fisheye view parameters.
- Display a normal view or a fisheye view of graphs.
- Run a VRML browser to render a normal view or fisheye view of objects.

4.2 Software Implementation

The implementation is done under SOLARIS 2.6. The java programs are compiled by JDK2. The major programs are FishEyeView.java, Graph.java and Object.java.

FishEyeView.java is the main class which setup the window and respond to the various menu commands.

Graph.java is the supporting class which process the loading and saving of 2D graphs data, computations of fisheye views transformation to the 2D graphs and displaying of the normal view and fisheye view of the graphs.

Object.java is the supporting class which process the loading and saving of 3D objects data, computations of fisheye views transformation to the 3D objects, convertion of object data into VRML file and rendering of normal view and fisheye view of the objects by a VRML browser.

5 Conclusion

Fisheye views provide new ways to display 2D and 3D graphics. In previous works, the fisheye views of 2D graphs were introduced. We adopted the basic ideas and extended it to focus polygons. We have put some new ideas into the fisheye views of 3D graphics. For the first time, we have introduced how to apply fisheye views to the 3D graphics. In addition to the focus point, we introduced the focus direction as a new element to apply fisheye views to 3D graphics. Now we can render 3D graphics in a new way.

In our implementation we have developed an interactive program to display normal view and fisheye view of 2D graphs and 3D objects. We can change the focus point, the focus polygon and the focus direction interactively and easily. It is a useful tool to show how to apply fisheye views to 2D and 3D graphics.

Our program can be extended to deal with more general 2D and 3D data structures. Because the display of almost all the 2D and 3D graphics are based on their coordinate points, all we need is more complicated data structures to define the 2D and 3D graphics and more computations to display the fisheye views of the 2D and 3D graphics.

Our implementation is a java application. It can also be converted into a java applet so that we can show how to provide fisheye views to the 2D and 3D graphics directly in the web pages.

In our program the 3D graphics data is converted into a VRML file first and then rendered by a VRML browser. This approach is easy to render the 3D graphics but the 3D graphics can not be controlled by the program. So it might be advantageous to use a 3D API such as java3d to render 3D graphics directly in the program.

References

[Sarkar and Brown 1994] Sarkar, M., Brown, M.H.: “Graphical fisheye views”; Comm. ACM 37(12)(1994) 73-84

[Sarkar and Snibble 1993] Sarkar,M., Snibble, S.S., Tversky,O.J., Reiss,S.P.: “Stretching the rubber sheet, a metaphor for viewing large layouts on a small screen”; Proc. Symp. User Interface Software and Technology (ACM,1993) 81-91

[Formella and Keller 1995] Formella, A.,Keller, J.: “Generalized Fisheye Views of Graphs”; Proc. Graph Drawing’95 Passan(1995)

[Winder and Roberts 1998] Winder, R., Roberts, G.: “Developing Java Software”; John Wiley & Sons (1998)