Generalized Fisheye Views of Graphs*

Arno Formella and Jörg Keller

Universität des Saarlandes, FB 14 Informatik
Postfach 151150, 66041 Saarbrücken, Germany

Abstract. Fisheye views of graphs are pictures of layouted graphs as seen through a fisheye lens. They allow to display, in one picture, a small part of the graph enlarged while the graph is shown completely. Thus they combine the features of a zoom—presenting details— and of an overview picture—showing global structure. In previous work the part of the graph to be enlarged—the focus region—was defined by a focus point. We generalize fisheye views such that the focus region can be defined by a simple polygon and show efficient algorithms to compute generalized fisheye views. We present experimental results on two applications where generalized fisheye views are advantageous: travel planning and ray tracing.

1 Introduction

Graphs are a common data structure in computer programs. Graph layout, i.e. the science to display a graph, has become important to visualize the underlying data sets and their relations. Focusing on one or several regions in a two dimensional layout is also important in other applications, e.g. CAD–systems. Interesting features in layouted graphs or generally in two dimensional images are both the global structure and the local structure in special regions. If the data set to be displayed is large, a picture showing the whole graph only allows to assess the global structure. Details can be obtained by zooming into some part of the graph, but global information is lost.

Fisheye views of graphs—the name is taken from the similarity to viewing a picture through a magnifying lens—try to combine both features. The node positions of the layouted graph are transformed such that a part of the layouted graph is displayed enlarged, but that the graph is still completely visible.

Fisheye views can be computed by defining a focus point and transforming node positions with respect to their euclidean distance from this focus point. The distance function can also be some other relation such as the length of the shortest path between the nodes. The different types are called graphical and logical fisheye views, respectively. A recent paper by Sarkar and Brown [3] gives a survey.

* This research was partly supported by DFG (German Science Foundation) under SFB 124 – TP D4. The first author is partly supported by Universidade de Vigo/Xunta de Galicia, the second author is partly supported by DFG through a habilitation fellowship.

Often however, one is not only interested in details in one part of the graph, but in several parts or even a whole region of the graph. The first demand can be accomplished by multiple focus points. The second demand can be accomplished by having one or several focus regions instead of a focus point. Misue and Sugiyama [2] used focus regions but restricted themselves to rectangles. Sarkar et. al. [4] allow only convex polygons. We will derive fisheye views based on a focus region from transformations based on a focus point. In contrast to Sarkar et. al. we allow arbitrary simple polygons, and our implementation is more efficient because we need not iterate. Moreover, the magnifying process is easy to reverse, so that editing in the distorted view is possible.

The remainder of this paper is organized as follows. In Sect. 2, we review graphical fisheye views as described in [3] and we derive general properties of the transformation function. In Sect. 3, we extend these transformations to focus regions formed by simple polygons, we describe our implementation and give performance results. In Sect. 4, we discuss the use of fisheye views on our applications, travel planning and ray tracing. Section 5 concludes the paper.

2 Computation of Fisheye Views

In a layouted graph, each node v is given a position $p(v)$ in the plane respective to an orthogonal coordinate system. Edges (v, w) are represented by straight lines from $p(v)$ to $p(w)$. All nodes and edges of the graph are positioned within the *frame*, an axis-parallel rectangle.

If we define a *focus point* f within the frame, then the position of each node v can be represented as

$$p(v) = f + \mathbf{a}_{v,f} \quad .$$

A fisheye view of a graph moves each node v from $p(v)$ to $p'(v)$ by applying a transformation function g to the vector $\mathbf{a}_{v,f}$ such that v is moved "away" from f but does not leave the frame:

$$p'(v) = f + g(\mathbf{a}_{v,f}) \quad .$$

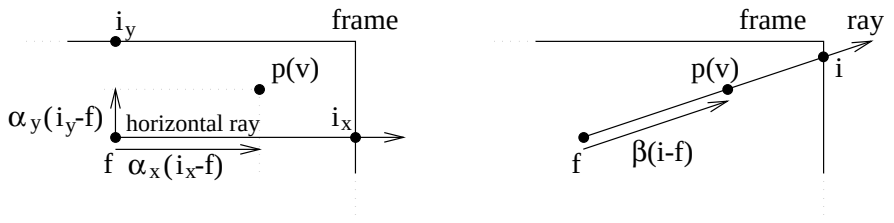
In [3], two possibilities for g are presented: cartesian and polar transformation.

2.1 Cartesian Transformation

We consider a vertical line through f that partitions the plane into two halfplanes. We define a horizontal ray starting in f such that $p(v)$ and the ray are in the same halfplane. Analogously, we consider a horizontal line through f and define a vertical ray starting in f. Let i_x and i_y be the intersections of these rays with the frame. Then $\mathbf{a}_{v,f}$ can be uniquely represented as

$$\mathbf{a}_{v,f} = \alpha_x \cdot (i_x - f) + \alpha_y \cdot (i_y - f) \quad ,$$

where α_x and α_y are in $[0:1]$. An example is given in Fig. 1(a).



(a) cartesian transformation (b) polar transformation

Fig. 1. Representation with respect to focus f

We define the *cartesian transformation function* g_c with the help of a bijective *distortion function* $h : [0:1] \rightarrow [0:1]$ as

$$g_c(\mathbf{a}_{v,f}) = h(\alpha_x) \cdot (i_x - f) + h(\alpha_y) \cdot (i_y - f) \quad .$$

We adopt from [3] the function $h(x) = (d+1)x/(dx+1)$, where $d \geq 0$ controls the amount of movement. More general, a function h should satisfy the following requirements, also partly given in [3]:

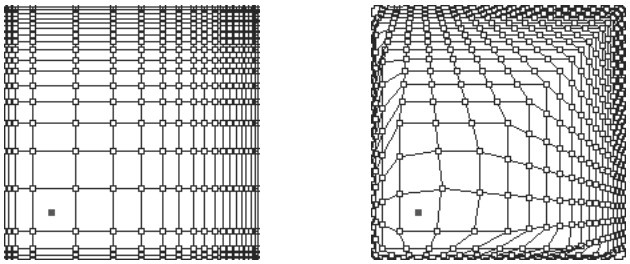
$h(0) = 0$**:** The focus point itself is not moved.
$h(1) = 1$**:** Points on the frame are not moved.
$h(x) > x$ **for all** $x \in (0:1)$**:** All other points are moved away from f.
$h(x)$ **be strictly monotonous increasing:** Points cannot overtake during the transformation, i.e. points being closer 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 to f before the transformation:
$h(x) - x$ is the absolute distance a point is moved during the transformation. By division through x, this is set in relation to the distance between the point and f before the transformation.

The above function satisfies these requirements. Another possibility would be $h_2(x) = \sin(\pi/2 \cdot x)$. It is obtained by considering fisheye views as projections of hemispheres onto planes.

2.2 Polar Transformation

Consider a ray through $p(v)$ that starts in f as shown in Fig. 1(b). Let i be the intersection of the ray and the frame. Then $\mathbf{a}_{v,f}$ can be uniquely represented as

$$\mathbf{a}_{v,f} = \beta \cdot (i - f) \quad , \tag{1}$$



(a) cartesian transf. (b) polar transf.

Fig. 2. Example graph after transformation

where $\beta \in [0:1]$. We define the *polar transformation function* g_p as

$$g_p(\mathbf{a}_{v,f}) = h(\beta) \cdot (i - f) \quad .$$

For both transformations, one can prove that no point will disappear. For cartesian transformations, one can prove that horizontal and vertical edges keep their orientation. Figure 2 shows a graph forming a regular 20×20-grid. It is transformed with $d = 3$, the focus point is marked with a filled square. All further pictures showing transformations will use $d = 3$ to have comparable results. The influence of d has already been analyzed by others [3].

3 Extension to Focus Polygons

For many applications, a simple focus point is not satisfying; e.g. for travel planning with graphs that represent road maps, one might want to focus on the whole area around the chosen route. Thus instead of a focus point, one needs a *focus region* surrounded by a simple closed curve. We will restrict to using polygons that completely lie within the frame.

To define transformations based on polygons, we will first deal with nodes that are positioned within the polygon. To derive transformations for nodes outside the focus 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 polygons consist of only one point.

3.1 Scaling Polygons

While in the previous section the area around the focus point was enlarged, here the focus region itself should be enlarged, but it should not be "deformed". Hence, we will scale the focus region. To do this, we have to define a center point c and a scaling factor γ. Then, any node v that is positioned within the polygon can be represented as

$$p(v) = c + \mathbf{a}_{v,c}$$

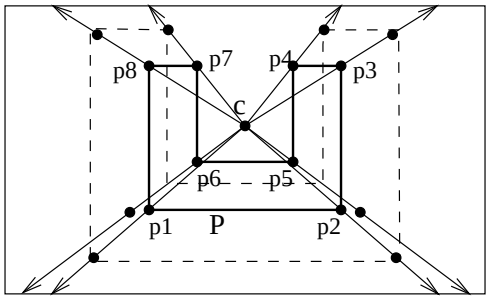


Fig. 3. Computation of the scaled polygon

and will be moved to

$$sc(p(v)) = c + \gamma \cdot \mathbf{a}_{v,c} \quad .$$

For a polygon P with n corners $p_1, \ldots, p_n$, we define the center point c as the simple barycentric combination

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

For a convex polygon, c will lie inside the polygon.

The scaling factor must guarantee that the scaled polygon still lies within the frame, and that the amount of scaling is somehow related to the transformation function g. As in (1), we represent each corner p_i of the polygon by a parameter β_i with respect to the center point c. The scaling factor γ then is computed as

$$\gamma = \min\{h(\beta_i)/\beta_i \ : \ 1 \leq i \leq n \text{ and } \beta_i \neq 0\} \quad .$$

The term $h(\beta_i)/\beta_i$ represents the factor by which corner p_i would be scaled from center c when moved by a polar transformation with focus point c. The minimum guarantees that the scaled corners and thus the scaled polygon completely lies within the frame. Figure 3 shows an example.

Scaling of the focus polygon P to $P' = sc(P)$ induces the following: we use P to determine parameters α_x and α_y or β for nodes v with position $p(v)$, and we use the scaled polygon P' to compute the new position $p'(v)$.

3.2 Cartesian Transformation

For each node v that is positioned outside P, we define two focus points $f_{v,x}$ and $f_{v,y}$, such that a line through $p(v)$ and $f_{v,x}$ ($f_{v,y}$) is horizontal (vertical).

We first assume that the horizontal line through $p(v) = (x_v, y_v)$ intersects P in $i_1 = (x_1, y_v), \ldots, i_t = (x_t, y_v)$. Then $f_{v,x}$ is chosen to be the intersection point (x_j, y_v) closest to $p(v)$. If P is convex, then $t = 2$ and $p(v)$ will be either

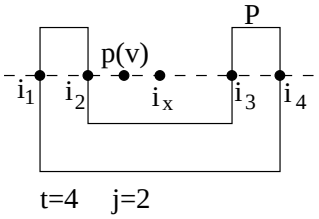


Fig. 4. Transformation with a non–convex polygon

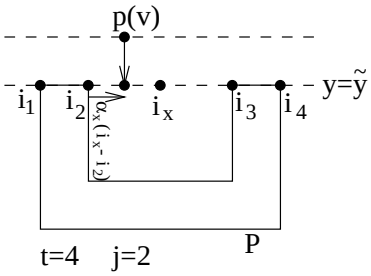


Fig. 5. Computation of α_x by a second horizontal line

left of i_1 or right of i_2. Then we can define i_x exactly as in Sect. 2.1 to be the intersection of a horizontal ray from $f_{v,x} = (x_j, y_v)$ and the frame, and

$$x_v = x_j + \alpha_x \cdot |i_x - f_{v,x}| \quad .$$

However, if P is not convex, then t might be a multiple of two and v might be positioned between two intersection points i_j and i_{j+1}. An example is given in Fig. 4. We choose as focus point the intersection point that is closer to $p(v)$. This property must also hold after the transformation. Thus, we choose i_x to be the point on the horizontal line with equal distance to both i_j and i_{j+1}:

$$i_x = ((x_j + x_{j+1})/2, y_j) \quad .$$

If the horizontal line through $p(v)$ does not intersect P, we choose the closest horizontal line $y = \tilde{y}$ that intersects P. We replace $p(v) = (x_v, y_v)$ by $(x_v, \tilde{y})$ and compute $f_{v,x}$, i_x and α_x for this point. The procedure is illustrated in Fig. 5. In exactly the same way, we compute $f_{v,y}$, i_y and α_y.

To compute $p'(v) = (x'_v, y'_v)$, we use the scaled polygon P'. We first scale $f_{v,x} = (x_j, y_j)$ to $f'_{v,x} = sc(f_{v,x}) = (x'_j, y'_j)$. If i_x was a point on the frame, i'_x is a point on the same edge of the frame, only the position on that edge is adjusted such that a line between i'_x and $f'_{v,x}$ is horizontal.

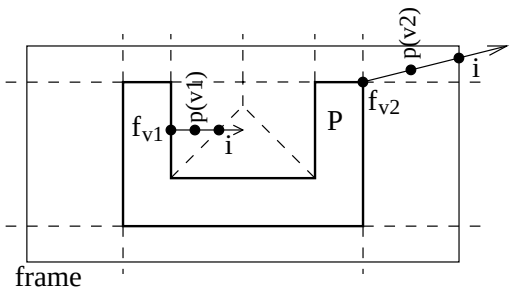


Fig. 6. Voronoi diagram of a simple polygon

If i_x lay in the center between two intersection points, then i'_x lies in the center between the two scaled intersection points:

$$i'_x = \left((x'_j + x'_{j+1})/2, y'_j\right) \quad .$$

Now, we can define $p'(v) = (x'_v, y'_v)$ as

$$\begin{aligned} x'_v &= x'_j + h(\alpha_x) \cdot |i'_x - f'_{v,x}| \quad , \\ y'_v &= y'_j + h(\alpha_y) \cdot |i'_y - f'_{v,y}| \quad . \end{aligned}$$

3.3 Polar Transformation

For each node v outside P, we define its focus point f_v to be the point on P with minimum distance to $p(v)$. This might either be a corner of P or the intersection point of one edge of P and the perpendicular line through $p(v)$. As the minimum distance property must also hold after the transformation, we have a problem in defining the intersection i. While we could use the center between two intersections with P in cartesian transformation, we need more notation here.

The *voronoi diagram* $\mathrm{VD}(P)$ 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 make all regions finite by intersecting $\mathrm{VD}(P)$ with the frame. We denote by $\mathrm{VR}(x, P)$ the curve that encloses the voronoi region (with respect to polygon P) that contains point x. Now we can represent $p(v)$ as

$$p(v) = f_v + \alpha \cdot (i - f_v) \quad ,$$

where i is the intersection of the ray r from f_v through $p(v)$ and $\mathrm{VR}(p(v), P)$. Figure 6 gives an example.

To compute $p'(v)$ we use once again the scaled polygon P'. We scale f_v to $f'_v = sc(f_v)$, and we compute i' to be the intersection of a ray from f'_v, parallel to r, and $\mathrm{VR}(f'_v, P')$. Then we obtain

$$p'(v) = f'_v + h(\alpha) \cdot (i' - f'_v) \ .$$

3.4 Implementation

We use scanlines to decide whether a point is located within the polygon or within a voronoi region, and to compute intersections with the polygon in cartesian transformations. To avoid recomputation of intersections between scanlines and polygons, we use a software cache to store scanlines and intersection points. We tested the usefulness of the software cache on a regular 30×30–grid and a graph with 900 randomly positioned nodes, using a rectangular polygon. For the grid, hit ratios are 98 % (cartesian) and 95 % (polar). For the random graph, hit ratios drop to 45 % and 0 %, respectively. This indicates that caching is mostly useful for regular graph layouts and for cartesian transformation.

We benchmarked our prototype implementation on a SPARCstation2. The user times to transform a 512×512–grid with respect to a rectangle are 30 s for cartesian and 57 s for polar transformation. The times with respect to a ring with 64 corners are 44 s and 206 s, respectively. Cartesian transformation is much faster due to caching effects and because computations are simpler (no intersections between arbitrarily oriented lines and polygons). Polar transformation time increases with the number of voronoi regions, i.e. polygon corners. This could be improved by a better search strategy to find the voronoi region of a point.

The overall performance is between 1.14 and 7.86 Milliseconds per transformed node. Hence, for graphs with a few hundred nodes, response times are still acceptable for interactive systems.

4 Applications

We illustrate the concept of focus regions on two applications, travel planning and ray tracing. Further applications where the concept of a dynamical viewport is of great value are CAD–Systems. Here, for instance, a designer needs to place wires in a certain region without loosing the global view of the long wires.

4.1 Travel Planning

Figure 8(a) shows the motorways in the southern part of Germany. Suppose we plan a travel from Nürnberg to Munich but are not sure whether to take the direct route or whether to visit Regensburg. With a single focus point, one can only concentrate on Nürnberg or on Munich (see Fig. 8(b) and (c)). However, the route between the towns is deformed. With the concept of a focus region, we can simply put a pentagon around the possible routes (see Fig. 8(d)). Cartesian and polar transformation with respect to this rectangle are shown in Fig. 8(e) and (f).

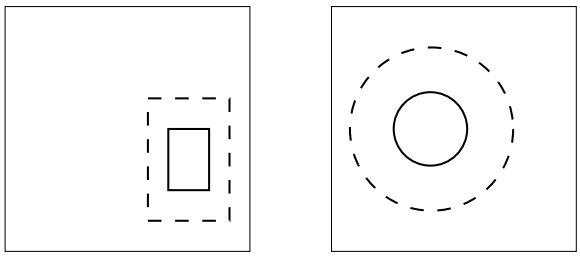


Fig. 7. Polygons to transform the example protein

The cartesian transformation in Fig. 8(e) seems better for us, because the surrounding motorways reflect more the original picture. This, however, may well be a matter of taste. Figures 8(b) and (c) show cartesian transformations for the same reason. In a complete travel planning environment, one would assume that each node and each edge has a certain priority. A node or edge is only visible if its priority supersedes a threshold that depends on the distance from the focus region. Then, in Fig. 8(e) and (f), smaller towns and roads along the main route would appear. The details of such features are described in [3].

4.2 Ray Tracing

Fisheye lens effects are also of interest in computer generated pictures. One of the main techniques to generate photorealistic pictures on a computer is ray tracing [1]. The basic idea is to have, in a 3–dimensional space, a camera position z, a view plane with pixel points, and a scene consisting of objects. From z, a ray is sent through each pixel. An intensity value for that pixel is obtained by computing intersections with objects and applying illumination models.

To have the impression of a fisheye lens effect on the scene, we simply apply a transformation with the inverse function h^{-1} to all pixel points on the view plane before calculating the primary rays through the pixel points. As h is bijective, this is always possible.

We incorporated our transformation code into a ray tracing program. Figure 9(a) shows the protein pdb2sni from the PDB[2] data base to illustrate the effect of the distortion on curved surfaces. We use the two polygons shown in Fig. 7. The circle is a ring with 64 corners. Figures 9(b) and (c) show the cartesian and polar transformations of the protein with respect to the rectangle. Figures 9(d) and (e) show the transformations with the ring.

An advantage of the polar transformation are fewer deformations of spheres close to the focus region. A disadvantage, however, are the deformations of spheres near the corner of the picture. Here, spheres get a corner. The reason for this is the rectangular form of the frame.

[2] Brookhaven Protein Data Bank

The concept of fisheye lenses introduces a new kind of special effect which can be used for zooming–in, zooming–out, image distortion, image morphism, etc., especially when animations are produced by an artist.

5 Conclusion

We have shown how to extend fisheye views based on a single focus point to views based on simple polygons. We have presented efficient algorithms and data structures to implement these views, and we have given performance results of a prototype implementation. We have tested our concepts on two applications: travel planning with computer generated road maps, and ray tracing. By allowing concave polygons, we are also able to handle multiple focus regions.

The prototype can still be extended in several ways. For maps, it might be useful to implement other features of fisheye views, e.g. presenting additional information like town names and population depending on the distance from the focus region. This extension is fairly simple given the representation with parameters (α_x, α_y) or β [3]. Another possible extension is hierarchical application, i.e. a lens within a lens.

In ray tracing, it might be helpful to use an animated fisheye lens in animated films. This allows to inspect three–dimensional close ups without loosing the overview. Further, a dynamic viewport might be useful in many two–dimensional display tools.

It is possible to extend cartesian transformation to arbitrary simple closed curves. Only the computation of intersection points between the curve and scanlines has to be changed. Scaling of the curve can be realized by approximation through a large number of points on the curve and computation of center and scaling factor with respect to these points.

Extension of polar transformation to arbitrary curves is more difficult because when given a point p, one has to find the point of the curve closest to p.

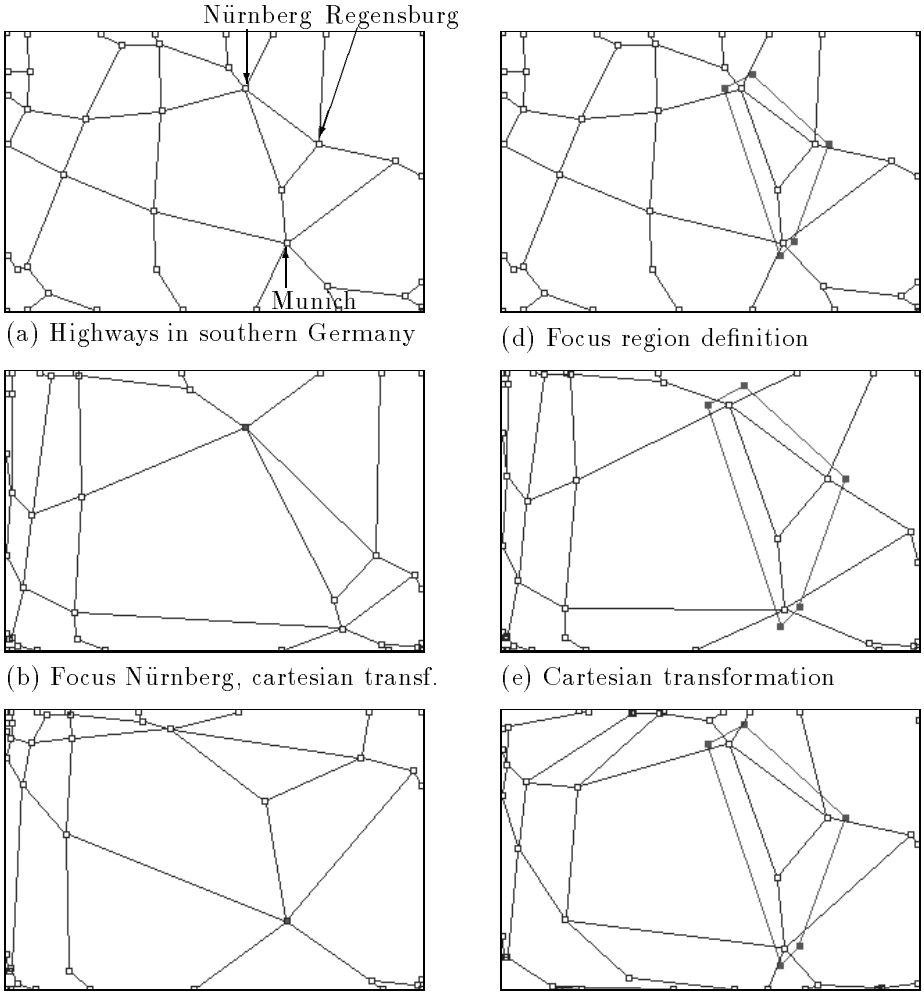
Acknowledgements

We would like to thank H.-P. Lenhof for providing the protein description.

References

1. Glassner, A. S. (Ed.): An Introduction to Ray Tracing (Academic Press, 1989)
2. Misue, K., Sugiyama, K.: Multi-viewpoint perspective display methods: formulation and application to compound graphs. Human Aspects in Computing, Proc. 4th Int.l Conf. on Human-Computer Interaction (Elsevier, 1991) 834–838
3. Sarkar, M., Brown, M. H.: Graphical fisheye views. Comm. ACM **37**(12) (1994) 73–84
4. Sarkar, M., Snibble, S. S., Tversky, O. J., Reiss, S. P.: Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. Proc. Symp. User Interface Software and Technology (ACM, 1993) 81–91

This article was processed using the LaTeX macro package with LLNCS style



(a) Highways in southern Germany

(b) Focus Nürnberg, cartesian transf.

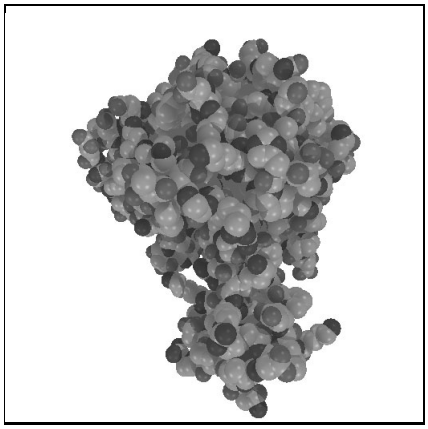
(c) Focus Munich, cartesian transf.

(d) Focus region definition

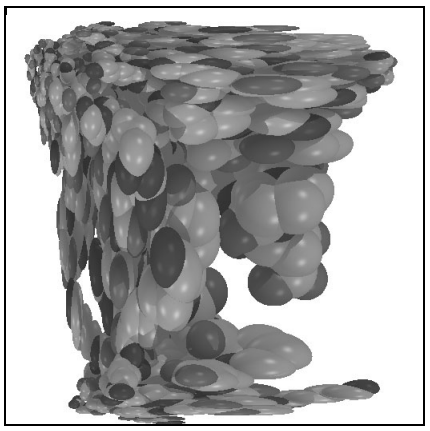
(e) Cartesian transformation

(f) Polar transformation

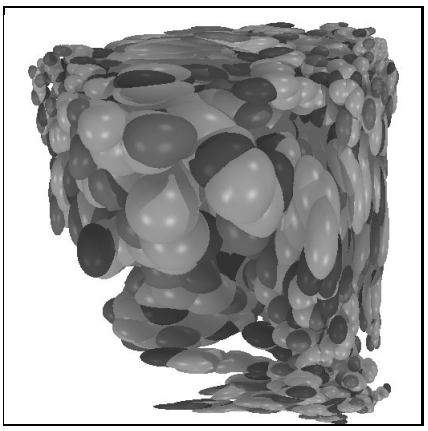
Fig. 8. Travel planning with focus region



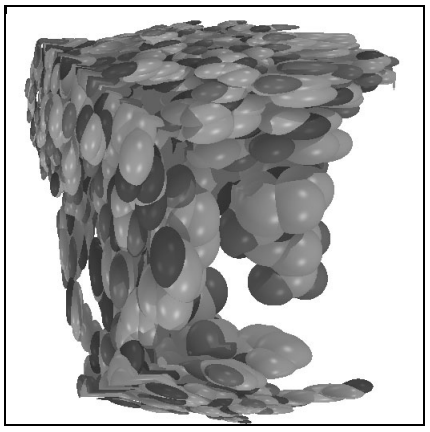
(a) Protein



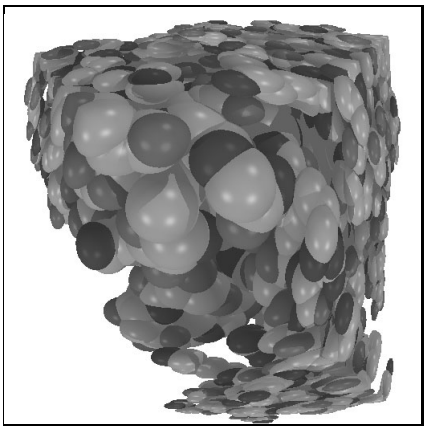
(b) Cartesian transf. with rectangle



(d) Cartesian transf. with ring



(c) Polar transf. with rectangle



(e) Polar transf. with ring

Fig. 9. Protein visualization