An Improved Fisheye Zoom Algorithm for Visualizing and Editing Hierarchical Models

Tobias Reinhard, Silvio Meier, Martin Glinz
Department of Informatics, University of Zurich, Switzerland
reinhard, smeier, glinz @ifi.uzh.ch

Abstract

Hierarchical decomposition is an important means for organizing and understanding large requirements and design models. Fisheye zoom visualization is an attractive means for viewing, navigating and editing such hierarchical models, because local detail and its surrounding global context can be displayed in a single view. However, existing fisheye view approaches have deficiencies in terms of layout stability when model nodes are zoomed-in and zoomed-out. Furthermore, most of them do not support model editing (moving, adding and deleting nodes) well.

In this paper, we present an improved fisheye zoom algorithm which supports viewing and manipulating hierarchical models. Our algorithm solves the problem of having a user-editable layout which is nevertheless stable under multiple zooming operations. Furthermore, it supports multiple focal points, and runs in real-time.

1. Introduction

Fisheye view visualization is a technique for visualizing large, complex information structures [1, 3, 8]. In particular, fisheye view techniques have been developed for visualizing the structure of a software system in software maintenance and re-engineering [11] and for viewing, navigating and editing graphical requirements and design models [2, 10].

Fisheye views magnify the objects in a focal point while reducing the size of objects farther away, thus achieving views that show local detail and global context together in a single view. It is also possible to support multiple focal points. Fisheye views are particularly attractive for visualizing hierarchical structures, because the classic visualization techniques (scaling, scrolling and explosive zooming) either provide only a global view (scaling) or show only a part of the structure, without its surrounding context (scrolling and explosive zooming). Schaffer et al. [9] conducted an experiment to compare fisheye visualization with traditional full-zoom techniques to view hierarchically clustered networks. The results suggested that the context provided by the fisheye view significantly improved a user's performance.

The heart of any fisheye-view visualization technique is its zoom algorithm. By *zooming-in* and *zooming-out*, a user can control the level of detail of each node. Fig. 1a) shows an abstract view of a hierarchical model: only the three top-level nodes are visible. The ellipsis after a node name is an indicator that this node has an inner structure which is hidden in the current view. By successively zooming-in nodes B and E, we get a view (Fig. 1b)) which shows the details of the model in a focal point (node E) together with the global structure of the model. Conversely, by zooming-out nodes in an expanded model we get a more abstract view.

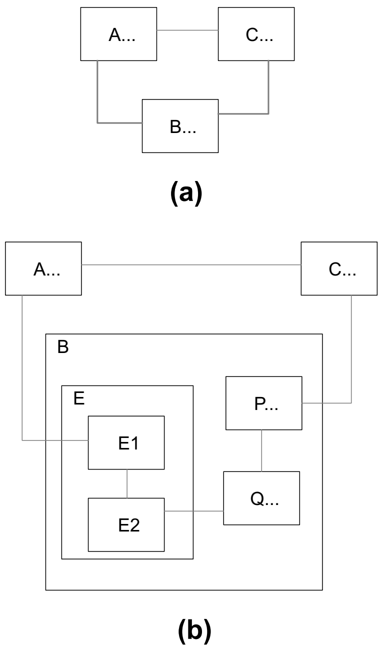


Figure 1. Fisheye zooming

From our experience with the fisheye zoom algorithm which we developed for visualizing ADORA models [4] and from the literature on fisheye views, we have derived a set of properties that a fisheye zoom algorithm should have. However, none of the existing algorithms, including the one that we have been using in our ADORA tool until now [2], satisfies all these properties.

In this paper, we

- present the properties that a fisheye zoom algorithm for viewing, navigating and editing hierarchical models should have;
- discuss existing fisheye zoom algorithms;
- and, as the main contribution of this paper, present an improved fisheye zoom algorithm that has all the desired properties.

In particular, our algorithm solves the problem of having a user-editable layout which is nevertheless stable under zooming (i.e., reproduces the initial layout when a sequence of zoom-in and zoom-out operations is undone by another sequence of zoom-out and zoom-in operations).

2. Desired Properties of a Fisheye Zoom Algorithm

When a node is zoomed-in or zoomed-out, the layout of the current view must be adapted accordingly by resizing and moving nodes. From the literature on fisheye visualization and from our experience with the ADORA tool [4], we have derived six properties that a fisheye zoom algorithm should have if we want to use it for visualizing and editing hierarchical, graphical models of requirements or design.

1. After a zoom operation, there should be *no overlap* between neighboring nodes, because overlapping nodes negatively influence the readability of a model and make it difficult to see hierarchical relationships (where nodes are contained in other nodes).

2. To maintain the understandability of a model, it is vital that the adjusted layout after a zoom operation resembles the original layout as far as possible. This is important because the modeler builds a cognitive map which consists of the positioning, the size and other user-defined visual properties of the model. According to Misue et al. [5], three properties should be maintained in an adjusted layout to *preserve the user's mental map*: orthogonal ordering, proximity relations and topology. The orthogonal ordering between nodes is preserved if the horizontal and vertical ordering of points is preserved. The proximity relations (or clusters) are maintained if nodes that were close in the original layout are still close in the adjusted layout. The topology is preserved if the adjusted layout is a homomorphism[1] of the original layout.

3. In order to maintain a user's cognitive map of a model, it is important to preserve the *stability* of the model: If a sequence of zoom operations is followed by a sequence of inverse operations, the result should be the original layout. A hard lesson that we learned from using our ADORA tool is that, when a sequence of zoom operations is reversed, it does not suffice to produce a layout similar to the original one in terms of orthogonal ordering, proximity and topology. For example, zooming-in nodes B and E in Fig. 1a) results in the layout of Fig. 1b). After zooming-out node B (or node E and then B), the resulting layout should be identical to that of Fig. 1a). The stability preservation should furthermore be commutative, i.e., the order in which a sequence of zoom operations is performed should not have an influence on the final layout.

4. A visualization technique for graphic requirements and design models must support manual modifications of the model: a user may re-arrange the layout of a model or s/he may change the model by adding or deleting model elements. Hence, the zoom algorithm must *permit editing operations* (add, remove, move and resize nodes) and preserve the modified layout as far as possible in subsequent zoom operations. Furthermore, the zoom algorithm should support automatic layout expansion when an inserted node is larger than the empty area at the insertion point as well as automatic layout contraction when a node is deleted.

5. Zooming is part of an interactive viewing, navigating and editing process. Hence, the zoom algorithm has to *run in real-time* (i.e., the zooming operations have to be performed with no remarkable delay).

6. It should be possible to view multiple areas of the model in detail simultaneously. The zoom algorithm should therefore support *multiple focal points*.

3. Existing Approaches

In this section, we briefly characterize the existing fisheye view visualization techniques.

[1] The links in a layout divide the plane into a set of regions. The *dual graph* [5] of this layout has these regions as nodes and an edge between two nodes that share a common boundary in the layout. Two layouts have the same topology if they have the same dual graph.

In 1986, Furnas implemented the concept of a Fisheye Lens [3] to provide a balance between local detail and global context in computer graphics. Furnas proposed a semantic zoom which shows or hides a point in a structure depending on its "Degree of Interest" (DOI). The DOI is a function of the "a priori importance" of the point and its distance from the current focal point. Furnas created systems for viewing and filtering structured programs, biological taxonomies and calendars. The original formulation by Furnas has no explicit control over the attributes that define the graphical layout and can therefore not be evaluated against the properties given in section 2.

Sarkar and Brown [8] brought layout considerations into the fisheye formalism by applying a wide-angle lens to a 2D layout. Their "Graphical Fisheye Views" produce a graphically distorted view of a layout by expanding the focal region and correspondingly contracting the other regions. Their fisheye views include planar and polar transformations of connected graphs and use Euclidean distance to calculate the degree of interest. All nodes within the network are shown, unless a particular node's "visual worth" falls below a threshold and is removed from the view. Graphical fisheye techniques are, however, non-trivial to implement and often have side effects that cause too much distortion (for example, text labels are very hard to read if they are geometrically distorted). Additionally, the layout cannot be adjusted by the user while parts of the model are magnified because the "fisheye view" is just a projection of the underlying constant "normal view".

Noik [6] provides an overview of existing layout approaches to visualize local detail and global context and proposes a classification schema for these techniques.

The SHriMP layout adjustment approach by Storey and Müller [11] tries to evenly distribute the distortion throughout the entire layout by uniformly scaling nodes outside the focal point(s). Nodes uniformly give up screen space to allow a node of interest to grow. The node that grows first pushes its sibling nodes outward by a translation vector. The node and its siblings are then scaled around the center of the screen so that they fit inside the available space. Different translation vectors to preserve different properties (orthogonality or proximity relationships) can be applied. The SHriMP approach guarantees overlapping free layouts if a node is zoomed-in; however, it cannot guarantee this property if the first zoom operation is a zoom-out (see section 4.3) or if multiple focal points are magnified in arbitrary order. The translation vectors can additionally be used to adjust a layout after an editing operation.

The "Continuous Zoom" by Bartram et al. [1] distributes a fixed amount of space (the display size) among nodes. The Continuous Zoom is inherently global, so that zooming-in a node (opening a cluster as this is called in [1]) changes the size of all nodes in the structure. The sibling nodes become smaller if a node's size is increased. The Continuous Zoom algorithm has two inputs: the "normal geometry" (the initial layout which is constant) and a scale factor for each leaf node. The amount of space requested by each node is summed up and a fixed space budget (the display size) is distributed according to the size of each request. The "zoomed geometry" (i.e., the current view on the model) is always derived from the normal geometry. The Continuous Zoom guarantees an overlapping free layout, preserves the orthogonal ordering and proximity relations, runs in real-time and allows multiple focal points. However, its global scaling changes the layout significantly (by changing the size of all nodes in the model) each time the size of just one node changes. Additionally, the algorithm does not permit editing operations while nodes are zoomed because the normal geometry is constant and the zoomed geometries are just projections of it.

In the ADORA project, we have developed a fisheye view technique for visualizing and manipulating ADORA models [2, 10]. The basic idea is similar to the one used in the SHriMP approach [11]. In contrast to SHriMP, our approach does not globally scale all nodes in the model after the sibling nodes were moved by the translation vector. The overall size of the model can therefore shrink or grow.

4. The Improved Zoom Algorithm

Our algorithm is based on the Continuous Zoom approach by Bartram et al. [1]: it works on an interval structure and employs the concept of scaling interval sizes. However, our approach does not assume a fixed basic layout, does not globally scale the whole model and uses different scaling functions. The improved zoom algorithm has all six properties described in Section 2, thus overcoming the limitations of the Continuous Zoom.

The purpose of the zoom algorithm is to produce a new layout if the size of a node changes because its inner structure is hidden or shown. It therefore takes an existing layout (the positions and sizes of a set of non-overlapping nodes) as well as a new size for an existing node η as input and calculates a new layout. The size of η is updated to the new size and the positions of the siblings[2] are adjusted to the new size of η in the new layout.

The zoomed-in or zoomed-out node η (or one of its siblings) pushes the boundaries of its parent outward or pulls them inward, respectively. The parent in turn pushes or pulls its siblings, which is done recursively until the root node is reached. Thus, the zoom algorithm is applied recursively on all the direct or indirect parents of η.

We assume a layout with rectangular representations of the nodes. However, this is no limitation as the boundaries

[2]Nodes which have the same parent in the hierarchy are called siblings.

of nodes with arbitrary shape can be represented by a rectangle. The zoom algorithm has to be applied multiple times if multiple nodes are zoomed-in or zoomed-out.

4.1. Data Structure

The algorithm works on an *interval structure* which is constructed by projecting the node boundaries on the X- and Y-axes. The spaces between the grid lines created by the projections are called *intervals*. Each node in the hierarchy which can be further decomposed has an interval structure of its own (in which the children's projections to the axes form the interval structure). Fig. 2 shows the projection of the three nodes A, B and C on the axes of the coordinate system. Y_1 to Y_7 are the *vertical intervals*, whereas X_1 to X_7 are the *horizontal intervals*. By $len(\delta)$ we denote the *length* of a specific interval δ. The length of a horizontal interval is the difference between its right and left boundary, while the length of a vertical interval is the difference between its bottom and top boundary.

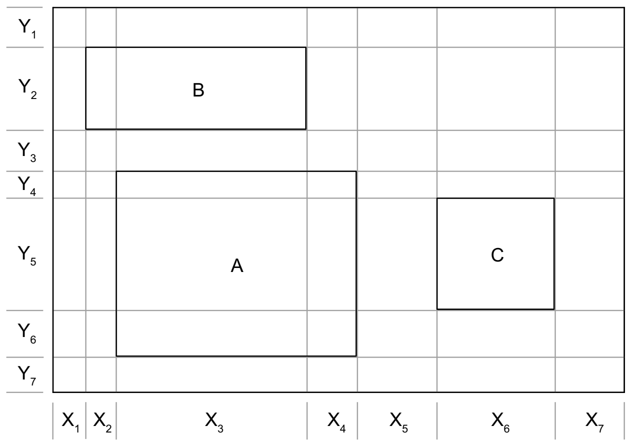


Figure 2. Interval structure

We distinguish between *node intervals* that are projections of nodes (or part of nodes) and *gap intervals* which are projections of the gaps between the nodes. Y_2, Y_4, Y_5 and Y_6 are the vertical node intervals in Fig. 2, while Y_1, Y_3 and Y_7 are the vertical gap intervals. We denote the set of horizontal node intervals of a specific node η by $proj_x(\eta)$ and the set of vertical node intervals of a specific node η by $proj_y(\eta)$. In Fig. 2 we have: $proj_y(A) = \{Y_4, Y_5, Y_6\}$ and $proj_x(A) = \{X_3, X_4\}$. The horizontal and vertical node intervals of a node η form together the node intervals of η: $proj(\eta) = proj_x(\eta) \cup proj_y(\eta)$.

Intervals can not overlap by definition, even though multiple nodes can be projected to the same node interval. Nodes A and B in Fig. 2 have, for example, both interval X_3 as horizontal node interval (i.e., $X_3 \in proj_x(A) \wedge X_3 \in proj_x(B)$).

The order of the vertical and horizontal intervals is always preserved if a zoom operation is applied to the interval structure even though the lengths of the intervals may change. This property is important to guarantee an overlapping-free layout after a zoom operation.

4.2. Zooming-In

The zoom algorithm has to increase the lengths of some of the intervals if node η is zoomed-in because η needs more space to show its internals. The intervals are adjusted by first calculating two independent scale factors, which are then applied to the vertical and horizontal node intervals of η. Adjusting the lengths of the node intervals of η repositions the siblings of η. The new size of η can be a constant value or it can be calculated with respect to the space that is needed by η's children. The two independent scale factors $s_x(\eta, \eta')$ and $s_y(\eta, \eta')$ are calculated as follows:

$$s_x(\eta, \eta') = \frac{width(\eta')}{width(\eta)} \tag{1}$$

$$s_y(\eta, \eta') = \frac{height(\eta')}{height(\eta)} \tag{2}$$

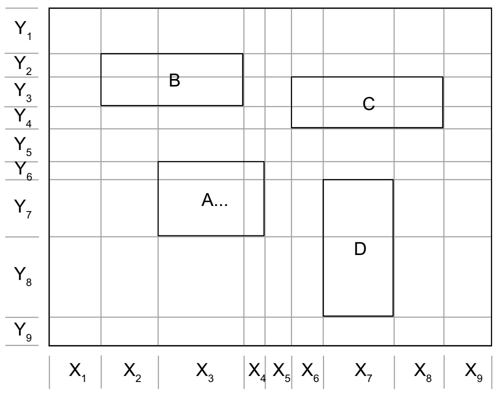
$width(\eta)$ and $height(\eta)$ denote the width and height of the boundaries of the node η. The node before zooming-in is represented by η, whereas η' is the node after zooming-in. The factors s_x and s_y are greater than one because the node is enlarged during a zoom-in. These zoom factors are then applied to the node intervals of η, i.e., the following predicate is satisfied

$$\forall \delta \in proj_x(\eta) : len(\delta') = len(\delta) * s_x(\eta, \eta') \tag{3}$$

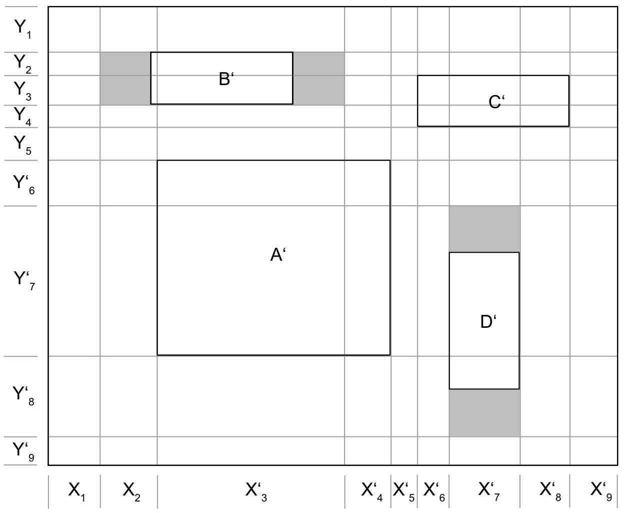
$$\forall \delta \in proj_y(\eta) : len(\delta') = len(\delta) * s_y(\eta, \eta') \tag{4}$$

δ' represents the interval δ after the zooming step. Fig. 3 gives an example: Fig. 3a) shows the initial situation with a zoomed-out node A. In a subsequent step, node A is zoomed-in, which is illustrated in Fig. 3b).

For zooming node A, the scale factors s_x and s_y are calculated as follows: the boundaries of node A in Fig. 3a) define the initial size and the boundaries of node A' in Fig. 3b) the new size. The lengths of the node intervals of A are then adjusted by these scale factors (the vertical intervals Y_6 and Y_7 by s_y and the horizontal intervals X_3 and X_4 by s_x). The lengths of the remaining horizontal and vertical intervals, i.e., those that are not in $proj_x(A)$ or $proj_y(A)$, are not changed.



(a)



(b)

Figure 3. Zooming-in node A

Fig. 3b) shows the situation after node A has been zoomed-in. The lengths of the vertical intervals Y_6 and Y_7 and the horizontal intervals X_3 and X_4 increased while the lengths (but not necessarily the boundaries) of the other vertical and horizontal intervals remained constant.

The two nodes A and D are both projected to the vertical interval Y_7 (i.e., $Y_7 \in proj_y(A) \wedge Y_7 \in proj_y(D)$) in Fig. 3a). The length of this interval is scaled by the factor s_y because the height of node A increases. In Fig. 3b), the area defined by the vertical projection of node D' (i.e., $proj_y(D') = \{Y'_7, Y'_8\}$) is no longer congruent with node D'. The difference between the lengths of the intervals and the boundaries of node D' is shown by the shaded area around node D' in Fig. 3b). We call this area *zoom hole*[3] of node D'. Moreover, the width of the zoom hole of node B' in Fig. 3b) is bigger than the width of node B' because the length of the vertical interval X_3 increased to X'_3 due to the zooming-in of node A.

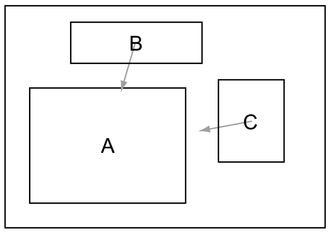
As shown for nodes B' and D' in Fig. 3b), the zoom hole's size can be bigger than the size of the node. The size of the zoom hole and the node may also be equal in size, as for nodes A' and C'. The zoom hole of node C' in Fig. 3b) is congruent with node C' because none of the node intervals of C is also a node interval of A (i.e., $proj_x(C) \cap proj_x(A) = \emptyset \wedge proj_y(C) \cap proj_y(A) = \emptyset$). Thus, each node is actually represented by a zoom hole in the interval structure.

A node is centered inside its zoom hole if the boundaries of the zoom hole are bigger than the boundaries of the node (as for example nodes B' and D' in Fig. 3b)). An alternative would be to maintain the relative position of the node's center inside the interval [1] or to enlarge the node to the size of the zoom hole. For our approach we chose the centering of the node.

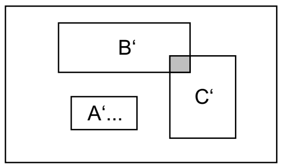
The zoom algorithm has to recursively adjust the interval structure of all the direct and indirect parent nodes of the zoomed-in node. This has to be done because their size has to be increased too if the size of one of its children changes.

4.3. Zooming-Out

Zooming-out is simple when the zoom-out operation immediately follows a zoom-in operation of the same node. In this case, zooming-out can be accomplished by just reversing the zoom-in algorithm. However, in the general case, zooming-out is not such easy. The algorithm should not generate overlapping nodes (Fig. 4) and the layout should be stable under multiple zooming-in and zooming-out operations (cf. property 3 in Section 2).



(a)



(b)

Figure 4. Node overlapping during zoom-out

[3]We follow here the terminology of [1].

Node overlapping cannot occur when using an interval structure, as long as the zoom-out algorithm preserves the order of the intervals. For achieving layout stability, however, the zoom algorithm by Bartram et al. [1] requires that the zoomed layout is a projection of a fixed base layout. This fixed layout prohibits any layout changes by the user.

Subsequently, we present an interval-structure-based zoom-out algorithm that produces non-overlapping and stable layouts and permits editing operations.

We first cover the simpler case in which the zooming-out is done after an initial zooming-in: Zooming-out node A' in Fig. 3b) results in the layout shown in Fig. 3a) because the scale factors which are applied to the node intervals of A' are the inverse of those that were applied to the node intervals of A during the zooming-in.

However, the stability property does not hold if the zooming-out operation is the first zoom operation (i.e., it is not following a zooming-in operation). The zoom algorithm has to be extended to guarantee the stability of the layout in these situations. Fig. 5a) shows an initial layout. Node A is zoomed-out in the subsequent step. The zoom operation results in the new node A' as shown in Fig. 5b). The scale factors s_x (for the horizontal intervals X_2, X_3 and X_4) and s_y (for the vertical intervals Y_4, Y_5 and Y_6) are now less than one because the size of node A decreases.

Fig. 5b) shows the layout after node A has been zoomed-out. The vertical interval Y_5 and the horizontal interval X_3 cannot be scaled down by the scale factors s_y and s_x because the lengths of the intervals would be smaller than the height of node C and the width of node B respectively. The zoom algorithm therefore has to maintain a *minimal length* for each interval that should be scaled down.

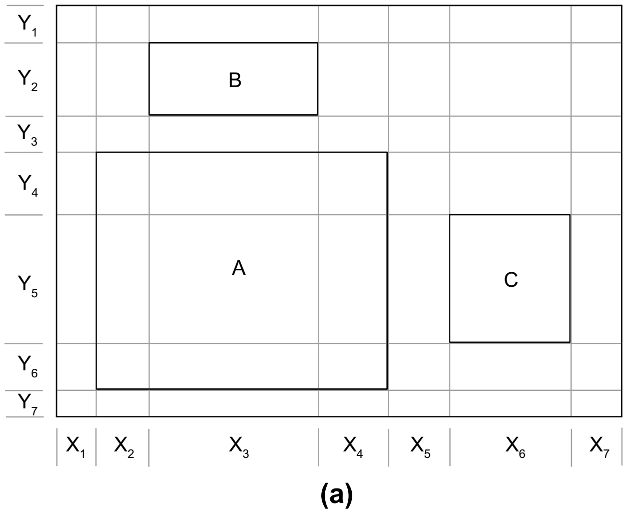
The minimal length l_{min} of an interval δ can be calculated as follows. Each zoom hole z_i defines for each interval it is projected to a property *real length* l_{real}. The value of l_{min} for interval δ is the maximal real length of all zoom holes z_i that are projected to δ, i.e., $\exists l \in \mathbb{N} : \forall k \in \mathbb{N} : 0 < l \leq m \wedge 0 < k \leq m \Rightarrow l_{real}(z_l, \delta) \geq l_{real}(z_k, \delta) \wedge l_{min}(\delta) = l_{real}(z_l, \delta)$, where m is the number of zoom holes that are projected to δ.

The real length for each interval is initially set to the length of the interval (i.e., $l_{real}(z, \delta) = len(\delta)$). It is set to the current real length of the interval scaled by s_x or s_y if node η is zoomed-in or zoomed-out ($z(\eta)$ denotes the zoom hole of node η and δ' the interval after the zoom operation):

$$\forall \delta \in proj_x(\eta) : l_{real}(z(\eta), \delta') = l_{real}(z(\eta), \delta) * s_x \quad (5)$$

$$\forall \delta \in proj_y(\eta) : l_{real}(z(\eta), \delta') = l_{real}(z(\eta), \delta) * s_y \quad (6)$$

For example, the real length of the zoom hole of node A' for the vertical interval Y_5' (i.e., $l_{real}(z(A'), Y_5')$) in the example of Fig. 5 is obtained by multiplying the scale factor s_y with the length of interval Y_5. The real length of the zoom hole of node C' for Y_5' (i.e., $l_{real}(z(C'), Y_5')$) is the height of node C' (which is the initially set value that wasn't changed during the zooming-out). The minimal length of interval Y_5' is the real length of the zoom hole of node C' because this length is bigger than that of the zoom hole of node A'.



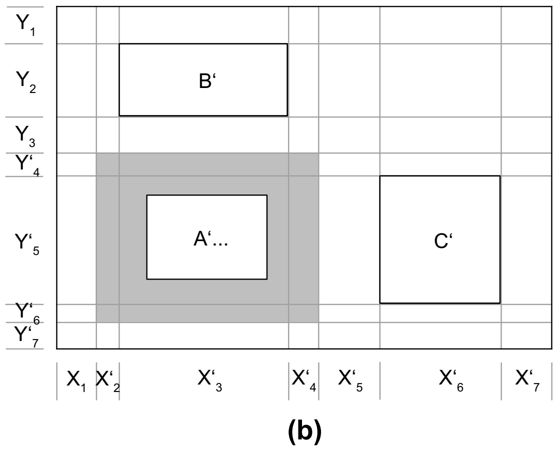


Figure 5. Zooming-out node A

The zoom algorithm has to store the real length for each interval in the zoom hole because the scaling up that may be done by a following zooming-in has to be done on this real length. The resulting node will be bigger than node A if node A' in Fig. 5b) is scaled up by multiplying the lengths of the intervals Y_5' and X_3' with the scale factors obtained from A' and A because Y_5 and X_3 are not scaled down during the zooming-out. This is due to the fact that our zoom algorithm always works on the existing interval structure and doesn't construct a new structure or adjust the structure to the new size of the nodes.

4.4. Editing Hierarchical Models

The insertion of a new node into a hierarchical graphical model is a tedious task because existing nodes have to be moved away to create the space required by the new node. On the other hand, the removal of existing nodes results in empty space in the model. A layout technique which automatically expands or contracts the layout if a node is inserted or removed is therefore valuable [10]. Most of the existing layout and fisheye zoom algorithms (including the Continuous Zoom) are not well suited for this task because they either scale the whole model up or down after a zoom operation or are based on a constant layout (a notable exception is the "Layout Adjustment Strategy" [5]), which prevents editing operations.

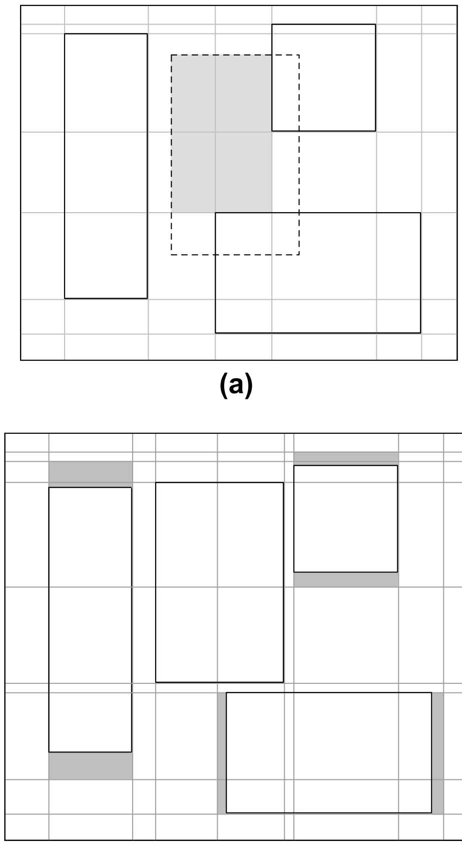


Figure 6. Inserting a node into the interval structure

A major advantage of our zoom algorithm is that it can easily be used for automatic layout adaptation when a node is inserted or deleted. For example, consider the situation in Fig. 6a), where a new node shall be inserted into the diagram at the position marked by the dashed rectangle. We first calculate the space the node can maximally occupy without moving any sibling nodes away which is shown by the shaded area in Fig. 6a). The node is then inserted with this reduced size. Finally, the zoom algorithm is used to increase the size of the node to its initially intended size, which moves its siblings away as shown in Fig. 6b).

When a node is deleted from a diagram, our zoom-out algorithm is used to contract the layout: First, the size of the deleted node η is reduced to the smallest possible size ($width(\eta') = 1$ and $height(\eta') = 1$, because the scale factor couldn't be calculated if the width or height would be zero) and the zoom algorithm is used to adjust the layout to the new size of the node (by moving its siblings). The node is, after this layout adjustment, removed from the interval structure.

Changing the position or the size of a node, resulting in a potential overlapping with other nodes, can be handled by the zoom algorithm by first removing and then reinserting the node (by the above described process).

4.5. Filtering Nodes

The described techniques to adjust the layout if a node is inserted or removed can also be used to hide or show selected nodes of a model [10]. Hiding a node η works the same way as removing the node (as described in the previous section), but without actually removing it but setting its height and width to an invisible size (i.e., $width(\eta') = 1$ and $height(\eta') = 1$).

A node that has been hidden in a previous filtering operation can be shown again by using the zoom algorithm to reset its size to the size it had when it was visible. The stability and commutativity of the zoom algorithm is of specific importance for this filtering of nodes because multiple filtering operations that are applied after each other can otherwise completely rearrange the layout of the model.

Furthermore, the zooming-in and zooming-out could be seen as a specific instance of the filtering: zooming-out just hides all children of the node while zooming-in shows them again.

5. Discussion

We briefly discuss now our zoom algorithm with respect to the desired properties presented in section 2:

Our zoom algorithm guarantees an *overlapping free layout* because the order of the vertical and horizontal intervals is always maintained.

The *orthogonal ordering* of the zoom holes in the interval structure is preserved if a node is zoomed-in or zoomed-out. However, the ordering of the nodes cannot always be maintained if a zoomed-out node becomes too small or a zoomed-in node too big. For example, the left boundary of node B lies in front of the left boundary of node A after node A has been zoomed out in Fig. 5. *Proximity relations* are maintained as far as this is possible if the size of a zoomed-in node increases significantly. The topological appearance of the layout is also preserved. However, we have not yet investigated whether the formal topology preservation criterion defined by Misue et al. [5] is met.

Our proposed zoom algorithm can guarantee the *stability* of the layout regardless of the order of the zoom operations due to the following properties: (i) the order of the vertical and horizontal intervals is always maintained, (ii) the real lengths are stored in the structure and (iii) the scaling of the node intervals is a commutative operation.

Editing and *Filtering* operations are supported by our zoom algorithm as described in sections 4.4 and 4.5 and our zoom algorithm supports *multiple focal points*.

A problem that remains is the fact that even with today's large display screens, fisheye views frequently grow beyond the size of the screen when many nodes are zoomed in. Most existing zoom algorithms [1, 11] scale the whole model up or down to a fixed size after a zoom operation, so that the view always fits the available screen size. However, this may result in very small nodes with unreadable labels. Our algorithm does not scale the adjusted layout after a zooming operation so that the size of the view grows or shrinks when nodes are zoomed-in or zoomed-out. When a view grows beyond the size of the available display area, scroll bars are provided to navigate the view. Additionally we offer a separate operation for linear view scaling. So for large views, the user can decide whether s/he prefers a view with reasonable node sizes (and readable labels) that needs scrolling or a scaled view that fits the size of the available display window.

Our presented zoom algorithm cannot adjust the layout of the model significantly if a node that "lies in the shadow" of another node is zoomed-out (node A is shadowed by node B if $proj_x(A) \cap proj_x(B) \neq \emptyset$ or $proj_y(A) \cap proj_y(B) \neq \emptyset$). Zooming-out nodes 2 and 3 in Fig. 8 results in a lot of free space between the two nodes because they are shadowed by the bigger node 1. However, this is a trade-off between maintaining the relative position between nodes 2 and 3 or the relative position between the two nodes and node 1 that cannot easily be resolved.

5.1. Complexity of the Zoom Algorithm

The runtime complexity of the zoom algorithm linearly depends on the number of intervals in the interval structure because the algorithm has to scale the intervals to adjust the layout. An interval structure with n nodes can have at most $2n + 1$ horizontal and $2n + 1$ vertical intervals.

Fig. 7a) illustrates the complexity of the algorithm for one node. There are at most three vertical and three horizontal intervals for a structure with one node (there may be less intervals if one boundary of the node overlaps with one boundary of the interval structure). Each additional node adds at most two vertical and two horizontal intervals. This is shown in Fig. 7b) for a second node.

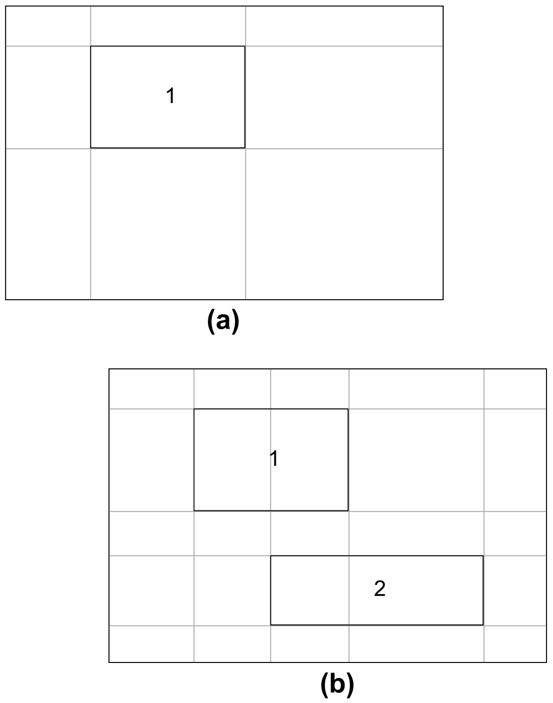


Figure 7. Number of intervals in the interval structure

The runtime and space complexity are therefore linear with respect to the number of nodes. The linear runtime complexity guarantees that our zoom algorithm can be used in an interactive environment.

The space complexity is important because some informations (boundaries of the intervals and the real lengths) of the structure have to be stored. The interval structures are constructed only once and then adjusted if a zoom operation is applied. Therefore, and the complexity of adjusting the interval structure dominates the complexity of constructing the structures.

6. Conclusions and Outlook

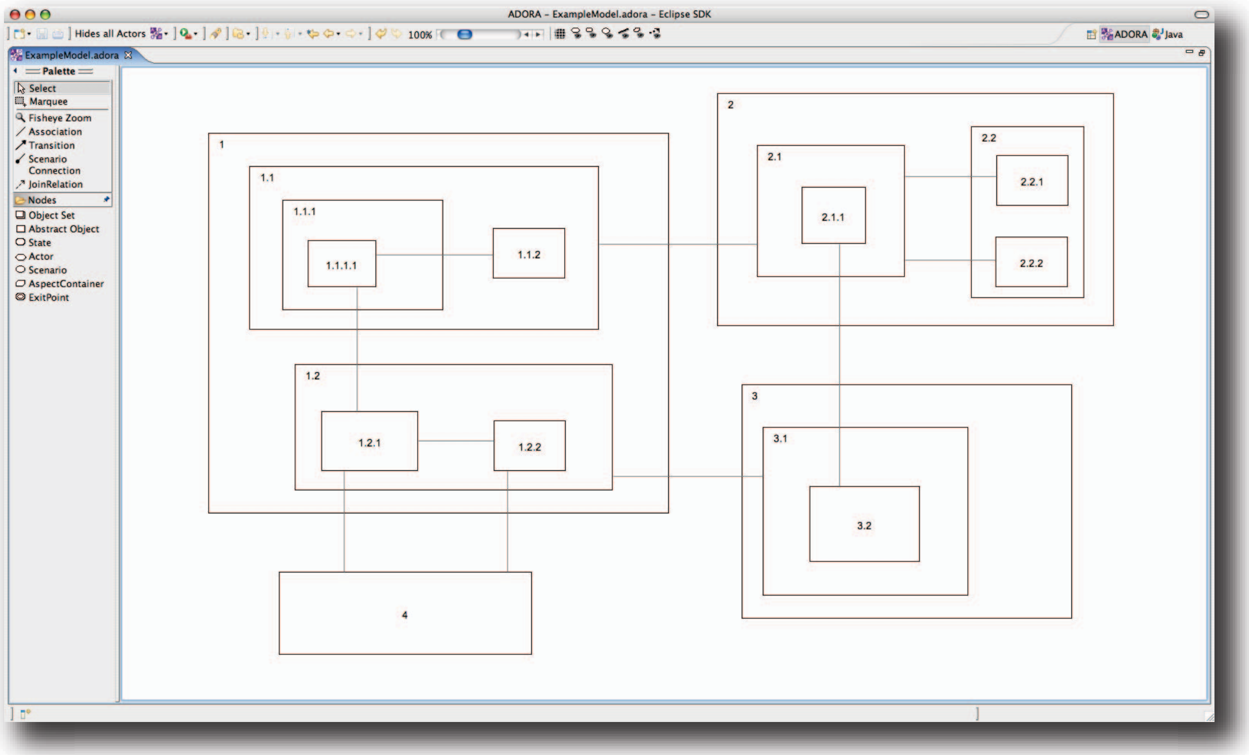
We have presented a fisheye zoom algorithm for visualizing and editing graphical hierarchical models. It guarantees the stability of the adjusted layouts and runs in linear time. In contrast to other approaches, it guarantees stable layouts independent of the order of zooming-out and zooming-in operations and allows model editing.

Our zoom algorithm has been implemented in our ADORA tool prototype[4].

Fig. 8 shows two screenshots of the ADORA tool: in the upper screenshot all nodes are in a zoomed-in state, whereas in the screenshot below nodes 1.1.1, 2 and 3 are zoomed-out. All figures in this paper (except Fig. 4) have been generated by our tool.

Our proposed zoom algorithm can, in addition to the tra-

[4]The ADORA prototype can be downloaded from http://www.ifi.uzh.ch/rerg/research/projects/adora/



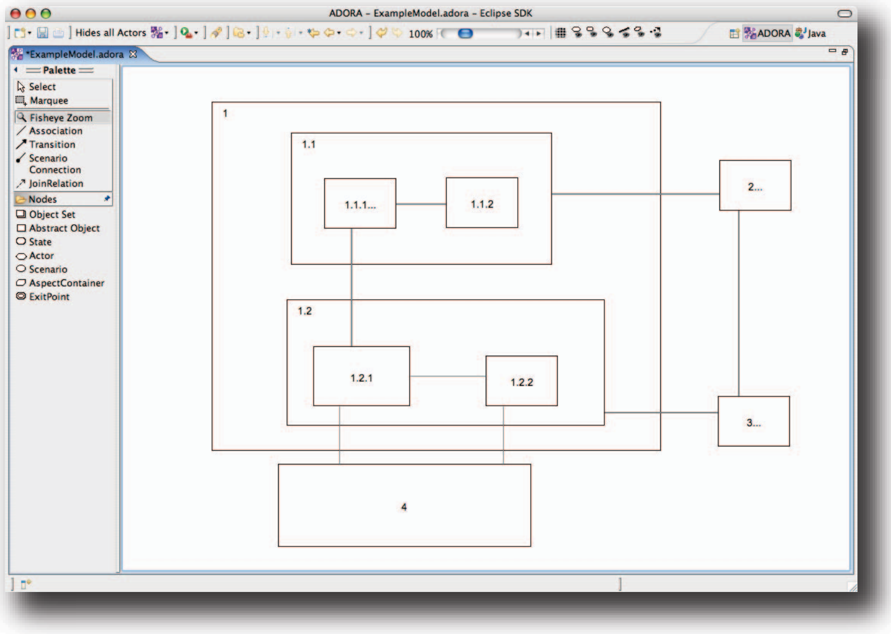


Figure 8. Screenshots of the ADORA tool

ditional concept of fisheye view visualization, also be used for the editing of models (moving, inserting and deleting nodes) and for the filtering of model elements.

We plan to integrate the algorithm with our previous work on efficient line routing in hierarchical diagrams [7]. The positioning of line labels is a major problem when generating views of graphical models, because they may easily overlap with nodes or other labels. Our zoom algorithm may be used to mitigate this problem by providing the necessary space for the labels while adjusting the layout just as far as necessary.

We also plan to further investigate the problem of supporting the user in editing models (especially if some nodes are currently filtered) and the filtering mechanism. Making the transitions between the layouts smooth by animating them may further help to maintain the user's mental map.

References

[1] L. Bartram, A. Ho, J. Dill, and F. Henigman. The Continuous Zoom: A Constrained Fisheye Technique for Viewing and Navigating Large Information Spaces. In *UIST '95: Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, pages 207–215, 1995.

[2] S. Berner, S. Joos, M. Glinz, and M. Arnold. A Visualization Concept for Hierarchical Object Models. In *Proceedings of the 13th IEEE International Conference on Automated Software Engineering (ASE-98)*, pages 225–228, 1998.

[3] G. W. Furnas. Generalized Fisheye Views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 16–23, 1986.

[4] M. Glinz, S. Berner, and S. Joos. Object-oriented modeling with ADORA. *Information Systems*, 27(6):425–444, 2002.

[5] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout Adjustment and the Mental Map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[6] E. G. Noik. A Space of Presentation Emphasis Techniques for Visualizing Graphs. In *Proceedings of Graphics Interface '94*, pages 225–234, 1994.

[7] T. Reinhard, C. Seybold, S. Meier, M. Glinz, and N. Merlo-Schett. Human-Friendly Line Routing for Hierarchical Diagrams. In *Proceedings of the 21st IEEE International Conference on Automated Software Engineering (ASE'06)*, pages 273–276, 2006.

[8] M. Sarkar and M. H. Brown. Graphical Fisheye Views. *Communications of the ACM*, 37(2):73–83, December 1994.

[9] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Navigating Hierarchically Clustered Networks through Fisheye and Full-Zoom Methods. *ACM Transactions on Computer-Human Interaction*, 3(2):162–188, June 1996.

[10] C. Seybold, M. Glinz, S. Meier, and N. Merlo-Schett. An Effective Layout Adaptation Technique for a Graphical Modeling Tool. In *Proceedings of the 25th International Conference on Software Engineering*, pages 826 – 827, 2003.

[11] M.-A. D. Storey and H. A. Müller. Graph Layout Adjustment Strategies. In *GD '95: Proceedings of the Symposium on Graph Drawing*, pages 487–499, 1996.