

# The Continuous Zoom: A Constrained Fisheye Technique for Viewing and Navigating Large Information Spaces

Lyn Bartram, Albert Ho<sup>†</sup>, John Dill and Frank Henigman  
Graphics and Multimedia Research Laboratory  
Center for Systems Science  
Simon Fraser University  
Burnaby, B.C. V5A 1S6 Canada  
+1.604.291.4369  
{lyn, dill, henigman}@cs.sfu.ca  
<sup>†</sup> present address: Reboot Inc., Vancouver, B.C.

## ABSTRACT

Navigating and viewing large information spaces, such as hierarchically-organized networks from complex real-time systems, suffer the problems of viewing a large space on a small screen. Distorted-view approaches, such as fish-eye techniques, have great potential to reduce these problems by representing detail within its larger context but introduce new issues of focus, transition between views and user disorientation from excessive distortion. We present a fisheye-based method which supports multiple focus points, enhances continuity through smooth transitions between views, and maintains location constraints to reduce the user's sense of spatial disorientation. These are important requirements for the representation and navigation of networked systems in supervisory control applications. The method consists of two steps: a global allocation of space to rectangular sections of the display, based on *scale factors*, followed by *degree-of-interest* adjustments. Previous versions of the algorithm relied solely on relative scale factors to assign size; we present a new version which allocates space more efficiently using a dynamically calculated degree of interest. In addition to the automatic system sizing, manual user control over the amount of space assigned each area is supported. The amount of detail shown in various parts of the network is controlled by pruning the hierarchy and presenting those sections in summary form.

## KEYWORDS:

graphical user interface, supervisory control systems, information space, hierarchical network, information visualization, fisheye view, navigation.

## INTRODUCTION

People tend to perceive the world using both local detail and global context; indeed our eyes supply detail for only a relatively small portion of the total field of view. Yet we rely on global context for orientation and to understand local

detail. Because of the importance of being able to deal with detail-in-context and the ubiquity of large information spaces, there have been several efforts to improve the ability to visualize large information spaces. While the need for such aids is widespread, ranging from understanding Prolog programs[6] to road maps[15], our interest is large real time systems such as power generation/distribution, telecommunications and process control where the operators must work with very large networks representing the system being controlled.

## PROBLEM AND MOTIVATION

Representing these large complex networks suffers from the well-known problems of viewing a large information space on a small screen. Approaches to alleviate these problems typically fall into one of three general classes: traditional pan/zoom, multiple window (or map view), and distorted view. The traditional pan/zoom uniformly scales the entire space and allows scrolling about it but suffers from the keyhole problem. The multiple window method (often referred to as *you are here*) shows a small overview and a large more detailed view. The overview contains a rectangle which can be moved and resized and whose contents are shown in the large view. It suffers from the extra space required for the overview, and from forcing the viewer to mentally integrate detail and context. Distorted-view methods reconfigure the display *in situ* to emphasize certain details within the larger context. A variety of such methods ([7][8][9][10][15][16]) have been proposed and have met with varying degrees of success. While distorted-view methods provide the most promise for our area, each of the reported methods suffered from some drawback as an effective interface technique in our domain, where user and display response times are critical and the complexity of the control task discourages increasing it with an unnecessarily complicated user interface.

We identified six important properties of an interface technique to show detailed views (sometimes called *focus points*) in a global context for time-critical systems with large information spaces:

1. It should be easy to see multiple detailed views concurrently.

2. Information, whether in a detailed or summary view, should never be occluded. Thus overlapping windows or viewports are not appropriate.
3. Spatial distortion should be minimized where possible. Excessive distortion in some methods has made them more difficult to use than standard techniques [19]. In particular, our case studies of network operators indicated they have a strong “geographic” model. We deemed it important to maintain the relative locations of nodes to reduce user disorientation.
4. The user interaction overhead required to achieve the desired effect should be reduced as much as possible, but the user should remain in control at all times.
5. The transition between views should be smooth.
6. Display reconfiguration should be immediate (within several frame times).

## RELATED WORK

Spence and Apperly’s Bifocal Display [18] represents one of the earliest computer-based distorted view approaches. In their method, the information space was represented by a sheet of paper, folded to compress the outer portions and leaving the central detail visible while providing global context. Mackinlay et al.’s Perspective Wall [8] updates this approach with a smooth transition between detail and context. A major contribution was Furnas’ *fisheye* method [7] which achieved a balance of local detail with global context in a manner quite different from the bifocal display and perspective wall. In his generalized fisheye view formalism, each element in a graph is assigned a *degree of interest* (DOI), or priority, based on *a priori* interest and on distance from the user’s current focus. The data is displayed only if the perceived value is greater than a user-set threshold. Each of these approaches supports only one detailed view at a time. Mitta [9] generalized these notions to non-hierarchical information structures and applied them to a complex 3D structure. This method is promising for complex diagram comprehension but does not generalize to 2D graph structures. An additional disadvantage is the occlusion of information inherent in 3D representations.

Noik [11] summarizes recent work in the use of distorted views in visualizing large information spaces represented with graph structures. Various methods are used to indicate the degree of interest; Noik classifies these emphasis techniques as *implicit* (use of perspective), *filtered* (removing items of low interest), *distorted* (size, shape and position of elements are transformed) and *adorned* (via attributes such as color). Other classification dimensions include priority method (how the DOI is obtained) and number of focal points allowed. The Perspective Wall, for example, is classed as a distorting method with a fixed distance calculation and is limited to a single focal point.

Sarkar and Brown’s method [14], more suited to full 2D graphs, uses filtering and distortion and supports multiple foci. The resulting images are impressive, though the authors

note users sometimes perceive the resulting view as too distorted and unnatural. Orthogonal and polygonal stretching, in which rectangular regions of the screen are selectively (de)magnified was investigated in [15]. The former maintains orthogonal ordering of points (nodes maintain their left-of, above, etc. relationships), while the latter does not. They also suggest it is important to keep precise user control of the space allocation. The interaction in [15] is somewhat cumbersome: the user acts upon the “rubber sheet” containing the object of interest rather than zooming in directly upon the object itself. Moreover, the method is limited to two hierarchical levels. Another approach to hierarchical structures is the Cone Tree method [13], which appears useful for tree-structured objects, but is not suited to networks such as those used in our application where nodes at the same level may not obscure one another.

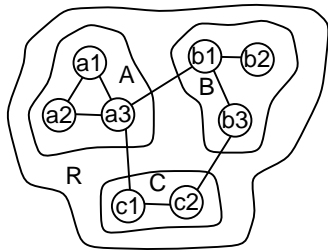
An interesting development is the semantic zooming notion of Pad++ [12][3] where one zooms into an infinite 2D sheet of paper. However it is not clear how changes to one portal affect another in the overall “global” environment; in other words, what happens when a portal claims extra screen space. Schaffer et al. report the Variable Zoom method for providing both detail and context for dealing with large, hierarchically structured networks[16]. It uses filtering and distortion, supports multiple foci, and allows the user to directly manipulate selected nodes. A later improvement to the method supports overlapping nodes [17]. However, the user has no direct control over the sizes of nodes aside from opening or closing them, and the transition between views is immediate and abrupt. Noik’s use of fisheye views in hypertext viewing also supports multiple foci. However, the lack of location constraints in his approach [10] allows elements to “float” around the screen space, causing a sense of distortion excessive for our purposes.

Based on the requirements listed earlier, and on the work in [16], we have developed a new approach, called the *continuous zoom*, which allows us to display detail and context in a flexible, timely and effective manner. Though this technique is particularly well suited to network visualization, it is applicable to many domains.

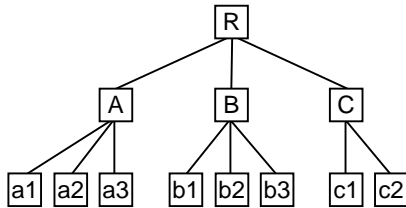
## THE CONTINUOUS ZOOM APPROACH

### Behavior

As we have previously described in [1] and [5], the continuous zoom manages a rectangular 2-D display space by recursively breaking it up into smaller rectangular areas, creating a hierarchy of nested rectangles. Hierarchy is a natural way to organize and understand a complex system. In a network viewing application, the display hierarchy would result from a hierarchical organization of the network nodes. Figure 1 shows the correspondence between a network and its display hierarchy. The network nodes correspond to the *leaf* nodes of the hierarchy. We call the interior nodes *clusters*. The user controls the amount of detail in different areas of the display by opening and closing clusters. The contents of an open cluster are visible, allowing one to see the deeper (more detailed) levels of the hierarchy. Closing a cluster



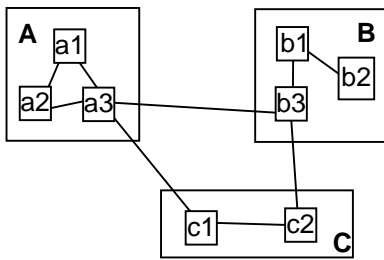
a) a small network with hierarchy indicated



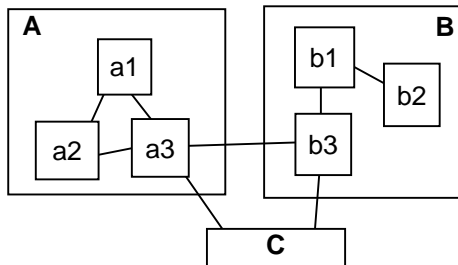
b) the display hierarchy of the network

**Figure 1.** Network and display hierarchy

effectively prunes a portion of the tree from the display, reducing the detail shown for that part of the network (Figure 2). Open clusters are allocated more space than closed clusters. In addition to this automatic resizing of cluster nodes, the user can enlarge or reduce any node on the display. When a cluster changes size, its contents are resized accordingly.



a) continuous zoom display with all clusters open



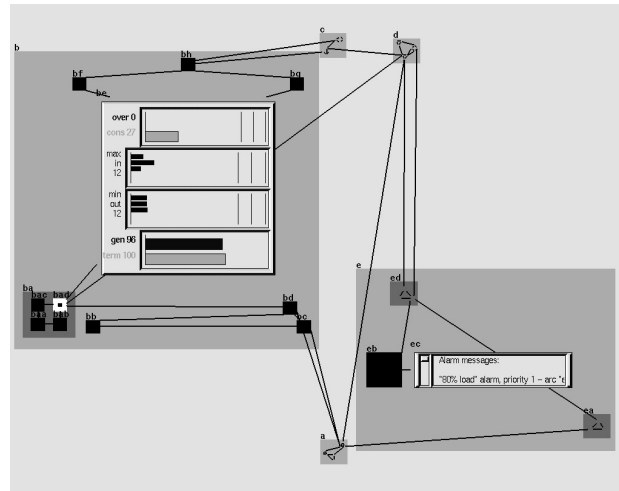
b) after closing cluster C

**Figure 2.** Network of Figure 1 with cluster C closed.

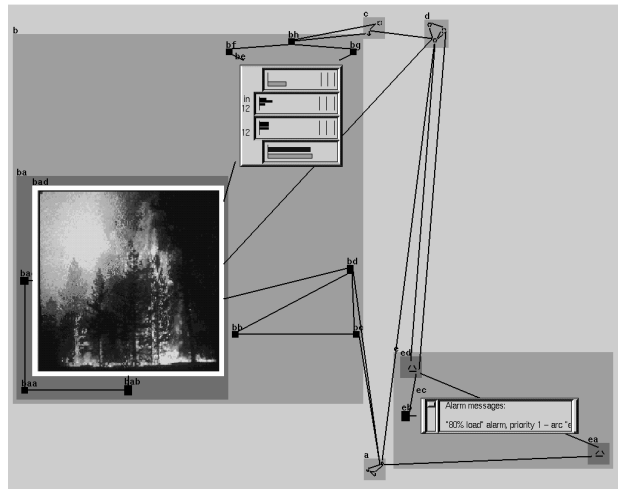
Whenever a node shrinks, it gives up display space to siblings so that they may grow. Through opening and closing clusters, and resizing nodes, the user has complete control over the amount of detail seen in each part of the display.

Since the entire hierarchy is visible at all times (though some of it is “summarized” by closed clusters) the detailed portions always appear in context. Multiple areas can be zoomed simultaneously; the technique allows more than one *focal point*.

We have implemented the continuous zoom as part of a prototype network supervision and control system (Figure 3). Several different node representations are used for moni-



Screen with node “in alarm” bordered in white.



Screen after user has selected node in alarm.

**Figure 3.** The continuous zoom in a prototype interface for a network control system

toring and controlling traffic flow through the net, which is modeled by a simulator. Each representation has a minimum size, so the algorithm must determine if the requested set of representations will fit on the screen. In a real application the

level of detail in the node representations might be adapted to the available space; we simply scale them.

Node size is controlled with the mouse. Pointing at a node and holding down a button increases or decreases the size of the node until the button is released. To reduce the amount of size manipulation required of the user, we also maintain a *degree of interest* (DOI) for each node, which is reflected in the node's size. Whenever a node representation changes, the new representation is automatically given the largest size possible based on its minimum size requirement, on its DOI and on available space (Figure 3). DOIs are dynamically calculated based on a node's a priori importance, its current state (in or out of alarm) and its proximity to interesting (high DOI) nodes. Thus neighbours of interesting nodes may get more space than those nodes farther away, contributing to the fish-eye effect.

The arcs (which we refer to as *links*) of the network are always drawn on top of the nodes. As the nodes shift around on the display, the link vertices are transformed with them, preserving adjacency. Then the vertices are connected to form the links.

The algorithm consists of two portions, an initial distribution of space and subsequent DOI-based size adjustments. The basic space redistribution was described in [5]; we review it here to set the context, and then discuss the improvements using the DOI approach.

### The Zoom Algorithm

The algorithm has two inputs: the initial layout of the network, or *normal geometry*, and a set of scale factors (one for each leaf node). The normal geometry and the scale factors are combined to produce the *zoomed geometry*, which is then displayed. In our application the normal geometry is constant; the display is controlled by changing scale factors and by opening and closing cluster nodes (i.e., by pruning subtrees of the hierarchy).

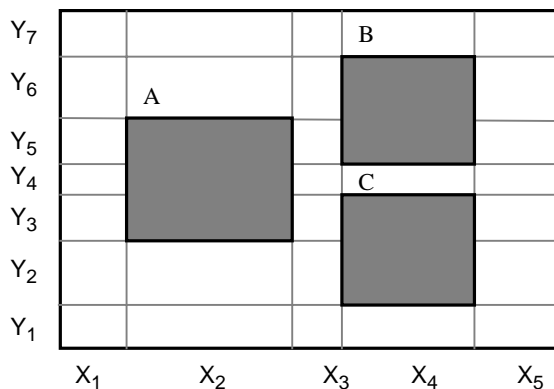


Figure 4. Sample three node network, initial size

The algorithm uses a “budgeting” process to distribute space among the nodes of a network. It sums the amount of space requested by each node and then distributes a fixed overall space budget according to the size of each request.

We first describe the algorithm using a simple network consisting of three nodes within a cluster node (Figure 4), and later discuss variations.

The algorithm works independently in the  $X$  and  $Y$  axes. It first breaks up the network into intervals by projecting all node boundaries, or edges, onto the  $X$  and  $Y$  axes. *Intervals* are the spaces between the “grid lines” created by these projected edges (the  $X_i$  and  $Y_i$  in Figure 4). Thus, by definition, intervals never overlap. However, since a node may occupy more than one interval, node projections may overlap. For example, in Figure 4, node C occupies two intervals in the  $Y$  axis:  $Y_2$  and  $Y_3$ . An interesting effect of this use of intervals for space management is that the north-south and east-west, or *relative*, geometric relationships between sibling nodes are always maintained, since nodes stay within their intervals.

A scale factor for each node controls the node's size. Since the algorithm works independently in  $X$  and  $Y$ , a separate scale for each axis is needed, and the user can easily control the size of a node by adjusting its scale factors. For example, to enlarge node A in Figure 4, its  $X$  and  $Y$  scale factors are increased together. Since the scales of nodes B and C do not change in an absolute sense, but decrease *relative* to the scale factor of node A, nodes B and C shrink as node A grows (Figure 5).

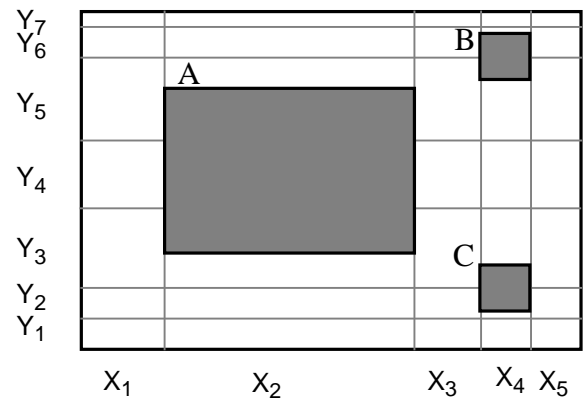


Figure 5. Node A increases at the expense of nodes B and C.

A scale factor is first computed for every interval. Each interval is either a projection of (a portion of) one or more nodes, or is an “inter-node” or *gap interval*. The scale factor for intervals corresponding to a node projection is just that of the node (e.g., when node A in Figure 4 grows, the scale of interval  $X_2$  increases along with node A's scale). When the projections of node boundaries overlap, as in the case of nodes B and C on the  $X$  axis, the interval  $X_4$  simply takes the maximum scale factor of all the nodes that project onto it. This ensures the size of the node does not exceed the sum of the sizes of the intervals that contain it. Intervals  $X_1$ ,  $X_3$  and  $X_5$ , with no nodes projecting onto them, are gap intervals. Since these intervals also require space, a scale must be defined for them; it could be fixed, equal to that of its neigh-

boring intervals, or some other value. We chose a value equal to the maximum scale of its neighboring intervals. The result of this choice is that when a node grows, the surrounding gaps also grow.

Given the scale factors of all nodes within a cluster, the total amount of space requested by the cluster (in the X direction) is:

$$X_{req} = \sum_i x_i s_i \quad (1)$$

where  $x_i$  is the normal length of the  $i^{\text{th}}$  interval and  $s_i$  is its scale factor. Space requested in the Y direction is calculated similarly. Intervals are used instead of node widths in (1) because intervals never overlap and because using intervals takes gaps as well as nodes into account.

As the nodes within a cluster increase in size, the cluster node must also grow in order to accommodate its children. This suggests the scale of the cluster node is “inherited” from its children’s scales:

$$S_p = \frac{X_{req}}{X_p} \quad (2)$$

where  $S_p$  and  $X_p$  are the scale factor and normal length of the parent in X.

The scale factor of and space request for each cluster node is propagated upward in this way until the root node is reached. Since the total amount of space is fixed (limited by the screen size), the size of the root node cannot change so the scale,  $S_i$ , of each interval throughout the hierarchy is divided by the scale factor of the root node,  $S_{\text{root}}$ . The zoomed length of an interval,  $x_i^*$  is just the normal length  $x_i$  multiplied by the modified scale factor:

$$x_i^* = x_i \left( \frac{S_i}{S_{\text{root}}} \right) \quad (3)$$

The length (and breadth) of the intervals containing a node constitute the total space available to the node (the *zoom hole*). The length of a node may not be equal to its zoom hole because of differences in scale factor or aspect ratio (see interval  $X_4$  and node C in Figure 5), so the final size of a node must be calculated separately:

$$L_i^* = L_i \left( \frac{S_i}{S_{\text{root}}} \right) \quad (4)$$

where  $L_i^*$  and  $L_i$  are the zoomed and normal lengths of the node. ( $S_i$  is the assigned scale factor for a leaf node and  $S_p$  is the computed factor for a cluster node.)

A cluster node can be resized by changing the scale factors of all the leaves within it uniformly. This produces a uniform change to the scale factors of all intervals, including

gap intervals.

After computing the sizes of the intervals and nodes, the nodes are repositioned according to the location of their center points. A node’s center stays at the same relative position in its interval as the size of the interval changes.

Note that because (4) is developed separately for X and Y, the zoom hole may not have the same aspect ratio as the original node (Figure 5). Often, however, it is desirable that the aspect ratio of a node should remain constant. To do this, we simply apply the smaller of the X and Y scale factors to both directions. On the other hand, a node not requiring a constant aspect ratio will simply occupy whatever space is

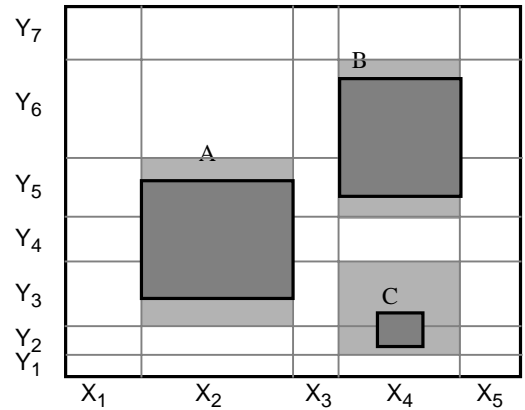


Figure 6. Node C scaled down from Figure 4 (zoom holes are shaded).

allocated.

One result of requiring a constant aspect ratio is occasional inefficient use of space, i.e. larger than necessary gaps between nodes. A second cause of such gaps is one node’s projection being contained in that of another (node C in Figure 5).

### Propagation

The algorithm as presented so far is *global* in nature: changing the scale factor of any one node changes the sizes of all nodes. Some users found the global change distracting and desired a more local effect. Our first response was to apply the algorithm only to a subtree of the hierarchy. By treating the parent of the node being scaled as the root of the hierarchy, only nodes below that point changed in size. A major drawback is that in order to enlarge a node beyond the size of its parent, the parent has to be manually enlarged first. Since the parent acts as the root of the hierarchy, it will not automatically grow to accommodate growing children. Thus the amount of user interaction required in the local variant is much greater than with the global algorithm.

We subsequently combined the global and local approaches into a *hybrid continuous zoom*. As in the local variant, the global algorithm is applied to the subtree rooted above the node being scaled. However, when the node grows

to some limit (in our current implementation, ninety-five percent of its parent’s area) the algorithm is restarted with the next highest ancestor as the root. That allows the node’s growth to be propagated to its parent, but still keeps the size changes limited to a subtree of the hierarchy. If the growing node grows enough, eventually the subtree root will get bumped all the way up to the true root of the hierarchy, beyond which no further growth is possible.

The ninety-five percent limit mentioned above is effectively an upper limit on node size. A lower limit on node size is also required, since it is useless to display representations that are too small to comprehend or to have nodes too small to pick with the mouse. The lower limit for each representation is set at runtime in pixels, with the smallest possible size being ten pixels square to support easy picking.

### Sizing Based on Degree of Interest (DOI)

The basic algorithm sizing behaviour has some shortcomings: it will stop as soon as any node becomes too small without checking the availability of space elsewhere in the network. For example, while enlarging node A (Figures 4 and 5), node C may have reached its minimum size, while node B is still larger than its minimum. In the basic algorithm any further attempts to enlarge A will fail since C cannot be shrunk any more. It makes more sense to use available space from B to satisfy A’s needs. In other words, we wish to “steal” space from any node that can afford it. Moreover, the basic version opens a node to some predetermined magnification factor; early user testing indicated a desire for more optimal sizing in which the node would open to its maximum size without extra user manipulation. Finally, given nodes with similar representations, we want their relative sizes to reflect their relative DOIs, visually cueing the user to their relative importance [10]. Our ultimate goal is to automate node sizing, to free the user from that task without in any way limiting the user’s freedom to manually resize as well.

We achieve the desired sizing effect by augmenting the basic algorithm with a two-stage calculation: we ensure that there is at least enough space to satisfy the minimum requirements of all requested representations, and if there is, we distribute any remaining unused space based on each node’s DOI.

To begin, the node/representation requests are sorted in descending order of DOI. (The DOI of a parent is the maximum of its children’s DOIs) Then all leaf node potential sizes are set to the minimum, or “closed” size, freeing up all the potential network space.

To calculate the minimum network size required, we go through the node list in descending DOI order. We set the requested size of each node  $Lreq_i$  to the minimum size of the desired representation. The leaf’s scale factor  $S_i$  is then simply:

$$S_i = \frac{Lreq_i}{L_i} \quad (5)$$

and the minimum length of the cluster  $Xreq$  is computed as in (1). Bottom-up recursion computes the new minimum network size. The final (root) minimum size request is compared to the root window size. If there is insufficient space, the algorithm stops, resetting the display to its previous configuration and prompting for external intervention from either the user or the system. (Such intervention may be a change in representation type or closing an open node, for example.)

If all minimum requirements can be met, any unused space is allocated in a top-down recursive fashion, beginning at the root, as follows. The amount of free space in the (cluster) node  $P$ ,  $Lfs_P$  is calculated as

$$Lfs_P = Lc_P - Lreq_P \quad (6)$$

where  $Lc_P$  is the node’s size calculated at the previous level of the top-down traversal (which starts with the fixed root). Space allocation to the node’s children is on the basis of the child’s DOI factor ( $doif_i$ ).

$$doif_i = \frac{DOI_i}{\sum_i DOI} \quad (7)$$

The amount of free space assigned to each child is

$$Lfs_i = doif_i \times Lfs_P \quad (8)$$

and the child’s new scale factor  $S_i^*$  reflects this new size:

$$S_i^* = \frac{Lreq_i + Lfs_i}{L_i} \quad (9)$$

Given the scale factor for each child, we find the scale factor for each interval that has nodes projecting on it by taking the maximum scale factor of the projecting nodes, as explained earlier. The length of those intervals is then

$$x_i^* = x_i S_i^* \quad (10)$$

When the interval sizes are added up, they may be less than the total space allocated to the parent node (due to overlapping children sharing the free space that was doled out). The leftover space is evenly distributed among the gaps, then the gap scale factors are calculated by plugging the desired gap size and original gap size in (10) and solving for scale. This differs from the basic algorithm where the gap scale factors are more or less arbitrary.

### Animation

An important goal of the continuous zoom is a smooth

transition between views. After determining that the minimum space requirements of each node can be met, and then calculating and redistributing the free space, we have a complete set of new scale factors for the nodes. Simply redrawing the network with the new scale factors would result in too abrupt a change in the appearance of the network. We therefore animate the change by linearly interpolating each scale factor between the initial and final values (computed as above) and redrawing the network at each step. About five steps seem to be sufficient to make the animation sufficiently smooth (in the opinion of our users).

## DISCUSSION

We are currently using the continuous zoom technique in a prototype interface to a simulated network management control system based loosely on power distribution and telecommunications networks. The zoom is part of a larger project to investigate the use of graphics and real-time reasoning techniques in intelligent user interfaces for supervisory control [1]. While the networks in our current domains of interest are typically very large (with thousands of nodes and links overall), the usual number of individual points and links with which an operator is closely concerned at any time is in the tens or hundreds.

The smooth animation of size changes proved to be a significant improvement over the abrupt transitions in our earlier zoom algorithms. The zoom algorithm is sufficiently fast (using a Silicon Graphics Indigo R3000 with entry-level graphics) to give the appearance of continuous change for modestly large networks (in which 50 - 100 nodes are currently visible, although the network may be much larger in its entirety). Speed depends mainly on how many nodes are visible, so it is possible to use networks with hundreds of nodes. Since there is little reason to have hundreds of nodes visible at once (they would be too small on a typical screen) network size should not be a limiting factor in applying the continuous zoom. The hybrid propagation scheme that resulted from combining the local and global approaches, essentially a "lazy" propagation method, proved quite satisfactory. The animation does suffer from certain discontinuities in motion; we are currently considering ways to correct this.

The free space allocation to nodes based on their DOIs can let some space go to waste. As stated earlier, overlapping nodes can result in leftover free space which is currently distributed to the gaps. In cases where it is not required for links, a better use of that space would be to somehow give it to nodes.

We have not yet fully explored the issues of *scalability* in this technique: i.e., how many nodes (and links) can be usefully managed. Scalability concerns not only screen space allocation, but also hierarchical structure and filtering or node visibility. We are planning studies on how many levels of hierarchy can be usefully employed to represent more complex information spaces. An interesting approach is the *semantic zooming* discussed in Furnas et al. [19] in which zooming affects not only the size of the area but also the choice of what view is displayed.

The algorithm has no inherent limitations on depth or breadth of hierarchy with respect to nodes. However, there is little provision for links; at present they are treated as an add-on, lines which are drawn between nodes. We are interested in how hierarchical structures may be used to represent links and how the algorithm may be extended to support space allocation and navigation issues in the *link space* as well as in the *node space*. Moreover, we need to examine network hierarchy. For example, how do users perform with broad shallow hierarchies as opposed to narrow deep ones? Also, is it possible to generalize the hierarchy from a tree to a directed acyclic graph?

Full filtering (in which nodes may be made invisible when they fall below a certain threshold) is currently not supported in the continuous zoom. Nodes are always accessible, in that even if they are "hidden" within a closed cluster, the path to them is always available. However, there are applications in which it may be appropriate to screen out this information on a dynamic basis (see the examples in [11]). We are considering extending the zoom algorithm to allow this by supporting dynamic network configuration, in which nodes and links can be added and deleted "on the fly" to the space allocation process and navigation structure. One issue of concern in dynamic network reconfiguration and filtering is the degree to which we can change the user's display without introducing disorientation, which may have undesired effects in a highly stressful, time-critical environment.

The continuous zoom has been developed in close collaboration with researchers and experienced network operators from the power distribution and telecommunications domains. Prototype testing on zoom variants with user groups over several iterations has guided the zoom development. Initial testing has involved both "naive" users and experienced network operators, and the response has been favourable, with most users commenting on the ease of navigating the large information space using the combination of detail-in-context views and direct manipulation interaction. Experimental validation is very important in our work, and it is something which is noticeably lacking from many of the promising techniques reported in the literature ([16] and [19] are notable exceptions.) Early experiments have shown the efficacy of a fisheye approach to a standard full-screen zoom approach in network monitoring and control tasks where navigating the network information space is an issue [16]. Simple experiments proved the desirability of the hybrid propagation technique over the local and global approaches. Experiments have been carried out which indicate that the continuous zoom in concert with intelligent assistance performs well in a simulated supervisory control system domain [2]. Formal studies comparing the current version of the zoom to more traditional interface techniques in the types of tasks our end users will be performing are in process at the time of this paper.

## CONCLUSION

The continuous zoom network viewing method provides multiple focus points in context with flexible control over node size. It supports efficient display reconfiguration with a

minimum of required user interaction but leaves the user in full control at all times. Disorientation is minimized because relative locations of nodes are preserved. This makes it superior to pan and zoom viewing, and other distorted-view techniques for large information spaces in time-critical applications. Hierarchical organization permits fast navigation through large networks, such as those found in supervisory control system domains. Because it partitions the network space into a hierarchy, the continuous zoom supports both the structural and associative thinking which Lai et al. point out are essential components of information searching [21]. In fact, Mukherjee et al propose a system of multiple hierarchical views as a means of effectively representing intricate networks [20]. While their approach is semantically based and does not support detail in context viewing, it holds promise for constructing *companion views* on additional screens which is the focus of our current research [22].

We believe this new method meets the requirements set out in the introduction for viewing/navigation tools for large hierarchical nets in a real time environment. It provides multiple focus points in context with flexible control over node size in a manner which reduces the interactive load on the operator. In particular, one of the goals for developing this method was to make better use of the space "left over" from the basic algorithm (i.e. the first step of the method we present here). We also wanted a visual enhancement of DOI. Both these goals are effectively achieved by the modifications described in this paper.

Other applications which have been brought to our attention as prime candidates for application of the continuous zoom are hypertext systems, software engineering visualization, and on-line training programs. While our development and application of this technique has been driven by problems related specifically to network management systems, there is no inherent characteristic of the algorithm itself which restricts it to network or graph structures. There is current interest in using it in tree-structured applications such as decision support systems and intelligent tutoring systems.

#### ACKNOWLEDGMENTS

We are indebted to our colleagues at MPR TelTech Ltd., the Alberta Research Council, TransAlta Utilities and the BC Telephone Co. for the chance to work closely with experienced supervisory control system developers and operators. The work described here was part of the Intelligent Graphic Interface project, made possible through the support of Industry, Science and Technology Canada, the British Columbia Ministry of Advanced Education, Training and Technology and PRECARN Associates.

#### REFERENCES

[1] Bartram, L., R. Ovans, J. Dill, J. Dyck, A. Ho and W. Havens. "Contextual Assistance in User Interfaces to Complex, Time-Critical Systems: The Intelligent Zoom." *Proceedings of Graphics Interface '94*, pp. 216-224, May 1994.

[2] Bartram, L. and T. Calvert. "Evaluating the Role of Intelligent Support in User Interfaces to Supervisory Control Systems." *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pp. 717-722, Oct. 1994.

[3] Bederson, B.B. and J.D. Hollan. "Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics," *Proceedings of UIST '94*, November 1994.

[4] Card, Stuart K., George G. Robertson and Jock Mackinlay. "The Information Visualizer, an information workspace." *Proceedings of ACM SIGCHI '91*, pp. 189-194, 1991.

[5] Dill, J., L. Bartram, A. Ho and F. Henigman. 1994. "A Continuously Variable Zoom for Navigating Large Hierarchical Networks". *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pp 386-390, Oct. 1994.

[6] Fairchild, K.M., S.E. Poltrock, and G.W. Furnas. "SemNet: Three-dimensional graphic representations of large knowledge bases." *Cognitive Science and its Application for Human-Computer Interface*, R. Guindon (Ed.), Elsevier, pp. 201-233, 1988.

[7] Furnas, G.W. "Generalized Fisheye Views." *Proceedings of ACM SIGCHI'86*, pp. 16-12, April 1986.

[19] Hollands, J.G., T.T. Carey, M.L. Matthews and C.A. McCann. "Presenting a Graphical Network: A Comparison of Performance Using Fisheye and Scrolling Views." In *Designing and Using Human-Computer Interfaces and Knowledge-Based Systems*, G. Salvendy and M. Smith (Eds), Elsevier, pp. 313-320, 1989.

[8] Mackinlay, J., George G. Robertson and Stuart K. Card. "The Perspective Wall: Detail and Context Smoothly Integrated." *Proceedings of ACM SIGCHI '91*, pp. 173-179, Apr. 1991.

[9] Mitta, D.A. "A fisheye presentation strategy: Aircraft maintenance data." *INTERACT '90*, pp. 875-880, IFIP, Elsevier, Aug. 1990.

[10] Noik, E. "Layout-independent Fisheye Views of Nested Graphs." *Proceedings of 1993 IEEE Symp. Visual Lang.*, Bergen, 1993.

[11] Noik, E. "A Space of Presentation Emphasis Techniques for Visualizing Graphs." *Proceedings of Graphics Interface '94*, pp. 225-233, May 1994.

[12] Perlin, Ken and David Fox. "An alternative approach to the computer interface." *Proceedings of ACM SIGGRAPH '93*, pp. 57-64, August 1993.

[13] Robertson, G.G., Jock Mackinlay and S.K. Card. 1991. "Cone Trees: Animated 3D visualizations of hierarchical information." *Proceedings of ACM SIGCHI '91*, pp. 189-194, April 1991.

- [14] Sarkar, M. and M.H. Brown. "Graphical fisheye views of graphs." *Proceedings of ACM SIGCHI '92*, pp. 83-91, April 1992.
- [15] Sarkar, M., S.S. Snibbe, O.J. Tversky and S.P. Reiss. "Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens." *Proceedings of ACM UIST*, pp. 81-92, Nov. 1993.
- [16] Schaffer, D., Z. Zuo, L. Bartram, J. Dill, S. Dubs, S. Greenberg and M. Roseman. "Comparing fisheye and full-zoom techniques for navigation of hierarchically clustered networks." *Proceedings of Graphics Interface '93*, pp. 87-97, May 1993.
- [17] Schaffer, D., Z. Zuo, S. Greenberg, J. Dill, L. Bartram, S. Dubs, and M. Roseman. "Comparing fisheye and full-zoom techniques for navigating large information spaces." *ACM Trans. Office Info. Sys.*, To Appear.
- [18] Spence, R. and Apperly, M. "Data base navigation: an office environment for the professional." *Behaviour and Information Technology*, 1(1), pp 43-54, 1982.
- [18] Spence, R. and Apperly, M. "Data base navigation: an office environment for the professional." *Behaviour and Information Technology*, 1(1), pp 43-54, 1982.
- [19] Furnas, G. and Bederson, B. "Space-scale Diagrams: Understanding Multiscale Interfaces." *Proceedings of CHI '95*, pp 234-242, 1995.
- [20] Mukherjee, S., Foley, J. and Hudson, S. "Visualizing Complex Hypermedia Networks Through Multiple Hierarchical Views." *Proceedings of CHI '95*, pp. 331-339, 1995.
- [21] Lai, Y. and Waugh, M. "The Effects of Three Different Hypertextual Menu Designs on Various Information Searching Activities". *Journal of Educational Multimedia and Hypermedia*, 4(1), March 1995.
- [22] Bartram, L., Henigman, F. and Dill, J. "The Intelligent Zoom as Metaphor and Navigation Tool in a Multi-Screen Interface for Network Control Systems", *Proceedings of IEEE Systems, Man & Cybernetics*, Oct. 1995 (to appear).