Smooth Structural Zooming as a Tool for Navigating Large Inclusion Hierarchies

Kevin Pulo, Peter Eades, Masahiro Takatsuka
School of Information Technologies
University of Sydney
NSW, 2006, Australia

{kev,peter,masa}@it.usyd.edu.au

ABSTRACT

We present a new method for achieving Focus + Context visualizations called smooth structural zooming, which varies the level of detail of the data being visualized, rather than geometrically distorting the visualization. We apply a preliminary smooth structural zooming technique to the horizontal–vertical (h–v) inclusion tree layout convention, in particular Design Behaviour Trees (DBTs). We illustrate several advantages of this system, including the ability to navigate and explore inclusion tree layout data too large to be displayed at once, keeping good layouts at all times and preserving the user's mental map with animation.

Keywords

Focus + context, smooth structural zooming, inclusion tree layout convention, visual complexity, animation

1. INTRODUCTION

Visualization of relational data plays an important role in software systems, particularly their design and modelling, for example, call graphs, data-flow diagrams, Unified Modelling Language (UML) [8], flowcharts and Nassi-Shneiderman diagrams [6]. These visualization techniques work well with small amounts of data, but like many visualizations they are often less effective for large amounts of data.

One general solution is to use a visualization technique known as *Focus + Context* [1]. This involves displaying a central *focus region* at full magnification, so that details in the data may be easily seen, and a surrounding *context region* at lower magnification, so that a general, high-level structure of the data is seen, providing user orientation. In the past, Focus + Context techniques have been based on *geometric zooming* techniques, such as the fisheye lens, hyperbolic browser, and perspective wall [1]. We present an alternate method of performing Focus + Context called *smooth structural zooming*, where context information is summarised rather than distorted, giving a constant level of visual complexity during navigation.

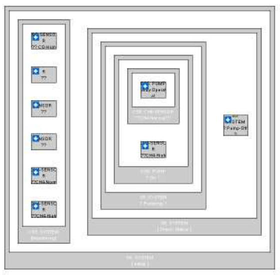
Figure 1: Part of the Mine Pump DBT as an inclusion tree.

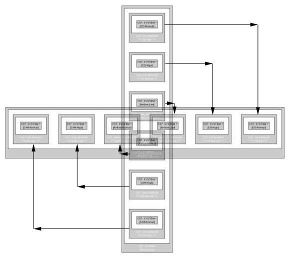
Figure 2: Changing a node arrangement.

We apply this new method to hierarchical tree structures in software engineering. We use the inclusion layout convention for displaying trees, as this is useful not only for strictly hierachical data, but is a good precursor for relational data which may be grouped according to a *cluster hierarchy* to give a *clustered graph*, such as call graphs and data-flow diagrams with nodes grouped by module or class. In particular, we illustrate the method using *design behaviour trees* (DBTs) [2], a technique for decreasing errors in the design and implementation of large software systems. Other tree-based data which the smooth structural zooming technique is suited to include the inheritance trees of object-oriented programming languages, and tree data structures, as may be used, for example, in interactive debuggers.

2. INCLUSION LAYOUT

The *inclusion tree layout convention* [3] is an alternate method of drawing trees where the parent–child relationship is visually represented by the child node being completely contained within the parent node. For simplicity, nodes are usually drawn as rectangles. A typical DBT is shown using the inclusion layout convention in Figure 1. This DBT describes the operation of a software system controlling a water pump in a mine. Inclusion trees are similar to *treemaps* [4], except treemaps generally have no margins around nodes and have some statistical data associated with the nodes.

Finding an inclusion layout for a tree is achieved by solving the *Minimum Inclusion Layout Problem* (MILP) [3] for each node, working upwards from the leaves. MILP is in fact NP-hard [5], however, we avoid this by allowing only two possible arrangments of the children of a node, horizontal and vertical, as seen in Figure 2, called *h–v arrangements*. In this restricted case of h–v arrangements (and integer node dimensions), we use a dynamic programming approach which solves MILP in polynomial time [3]. We define the 'size' of a node as the *minimum enclosing square*, as this appears to give good results empirically [7].

3. NAVIGATION TECHNIQUE

We say that a visualization has an intrinsic *level of detail* (or simply 'detail'), indicating the amount of data present, and a *level of visual complexity* (or simply 'visual complexity') indicates how many visual elements or attributes are being used to present this data. Different visualizations of the same data at the same level of detail may have different visual complexities, however, we consider the visual complexity of a higher detail visualization to always be greater than that of any lower detail visualization.

Our goal is a system which maintains an approximately constant level of visual complexity, while allowing the user to visualize and navigate inclusion trees that would ordinarily require a much higher visual complexity. Note that distortion techniques, such as the fish-eye lens, have higher visual complexity — rather than drawing the context data in a distorted fashion, it's better to summarise that context data, which gives a lower visual complexity.

The user has two *detail-increasing operations* available:

- expanding a node, revealing its child nodes and increasing its size to accomodate these children, and
- zooming out a level, allowing the user to see parts of the tree previously obscured by zooming in.

The system maintains the number of leaf and collapsed nodes visible as its detail measure. If it determines that this has risen unacceptably high as a result of user operations, it has two *detail-decreasing operations* available which it can perform in response to the user's stimulus:

- collapsing a node, hiding its children and returning to its original size, and
- zooming in a level, obscuring outer regions of the tree (containing collapsed nodes), allowing the user to concentrate on the central region of expanded nodes.

The response of the system to zooming out cannot include zooming in (as this would be idempotent), but may include the collapsing of any nodes. The response of the system to the expanding of a node cannot include the collapsing of that node or any of its ancestors, but may include collapsing of any other node and may include zooming in one or more levels. In addition, the user can choose to perform either of the detail-decreasing operations directly.

For these detail-reducing operations, the system must decide which nodes to collapse (if any), and how many levels to zoom in (if at all). If there are expanded nodes available to be collapsed, the system will consult a queue of nodes in order to find the least recently used (LRU) nodes. As many nodes are removed from the queue and collapsed as is necessary to reduce the detail to an acceptable level. After a node is expanded it is added to the end of the queue and its ancestors are moved to the end of the queue (in order), so that deeper nodes are considered first for collapsing. However, if there are no nodes available for collapsing (that is, all expanded nodes are ancestors of the currently expanding node) then zooming in is the only recourse, and so the system zooms in as many levels are necessary to sufficiently reduce the detail.

The transitions between different views are smoothly animated in order to preserve the user's mental map. Expanding and collapsing of nodes, as well as zooming in and out, are performed by simple linear interpolation of the node size and display clipping rectangle, respectively.

As the user changes their view of the visualization its layout must be updated in order to maintain the its quality. Since the system keeps the number of visible nodes constant, the new layout may be found simply by rerunning the layout algorithm. When zoomed in, only the layout of the inner visible region is updated and the layout of the outer obscured region remains fixed. This helps to preserve the user's mental map by preventing changes to hidden parts of the visualization.

Adjusting the layout requires changing the arrangement of one or more nodes from horizontal to vertical, or vice-versa. This is achieved by "rotating" the children nodes about the center of the node in question, as illustrated in Figure 2. Our system uses the *orthogonal* strategy of rotation, which interpolates the positions of the children along a "Manhattan path". Sometimes the system may need to perform more than one operation, in which case it must choose which animations will be performed *consecutively* and which will be performed *concurrently*. In general this is non-trivial, however as our system has a maximum of three possible animations it simply performs all animations concurrently.

An accompanying video [9] shows the animations used by the system on the Mine Pump DBT from Figure 1. It shows a typical navigation through the DBT, illustrating the main features of the smooth structural zooming system.

4. REFERENCES

[1] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization (Chapter 4)*. Morgan Kaufmann, 1999.

[2] R. G. Dromey. Genes, jigsaw puzzles and software engineering. In *Proceedings of the 9th Asia Pacific Software Engineering Conference (APSEC)*, page to appear. IEEE, 2002.

[3] P. Eades, T. Lin, and X. Lin. Two tree drawing conventions. *International Journal of Computational Geometry and Applications*, 3(2):133–153, 1993.

[4] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proc. IEEE Visualization '91*, pages 284–291. IEEE, 1991.

[5] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.

[6] I. Nassi and B. Shneiderman. Flowchart techniques for structured programming. *ACM SIGPLAN Notices*, 8(8):12–26, 1973.

[7] K. Pulo and M. Takatsuka. Inclusion tree layout convention: An empirical investigation. In *Proceedings of the Australian Symposium on Information Visualisation*, page to appear. CRPIT Vol 24, Tim Pattison and Bruce Thomas, eds, 2003.

[8] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modelling Language User Guide*. Addison-Wesley, 1999.

[9] http://www.it.usyd.edu.au/~kev/softvis03-kpulo-poster.wmv.