Nested User Interface Components

Ken Perlin, Jon Meyer
NYU Media Research Lab
Department of Computer Science
719 Broadway, 12th floor
New York, NY 10003
Tel: 1-212-998-3374
E-mail: {perlin,meyer}@cs.nyu.edu

ABSTRACT
Nested User Interface Components combine the concepts of Zooming User Interfaces (ZUIs) with recursive nesting of active graphical user interface widgets. The resulting system of recursively nesting interface components has a number of desirable properties. The level of detail of the view of any widget component and its children, as well as the responsiveness of that component to the user's actions, can be tuned to the current visible size of that component on the screen.

We distinguish between the interaction style of a component, and the semantic result that it produces. Only the latter is used to determine the geographic parameters for that component. In this way, very large and layered control problems can be presented to the user as a cohesive and readily navigable visual surface. It becomes straightforward to layout interaction semantics that are best handled by recursion, such as filters composed of nested expressions.

KEYWORDS: Zooming User Interfaces, Nested Interfaces, Widgets, Property Editing, Control Hierarchies.

INTRODUCTION

Over the last two decades, as Moore's law has continued at its remarkable pace, computer users have grown accustomed to increasingly powerful programs, utilizing progressively more sophisticated user interfaces. On the other hand, there has been only a modest increase in the size of display screens attached to most computers – applications must somehow pack more features into the same amount of space.

Software designers have attempted many strategies to deal with this increasing density. One of the most successful has been the adoption of nested hierarchies, such as those found in nested pull-down menus and control panels with multiple levels of options. The nested hierarchy strategy has become nearly ubiquitous because it embodies powerful semantics and manages complexity at each level.

Since nested hierarchies have become nearly ubiquitous, it is of great importance that nested hierarchy implementations possess high efficiency, ease of use, and user acceptance. If this goal is not kept in mind, many of the coming increases in application power will not be realized in their user interfaces. Already, many users aren't aware of their computer's more advanced features. Users lose patience while searching through scattered control panels, waste time switching between multiple representations because no single representation can contain the levels of information they need in a manageable space, and otherwise are hindered by current implementations of nested hierarchies.

We are developing what we believe to be a significantly more effective approach to representing and working with nested control hierarchies -- nested user interface components. Nested Components embody a constraint-based system for containing interface elements within one another, using a well defined and consistent geography, while allowing user customization to take place directly within the workspace. By allowing users to create arbitrarily nested semantic expressions through a controlled GUI, we offer the advantages of semantic flexibility found in language-based interfaces, with the error checking, easy learnability and ease of use features characteristic of GUIs.

ZOOMING USER INTERFACES

For almost ten years we have been investigating user interfaces which use zooming as a primary dimension of navigation (e.g. [26] [11] [1] [2] [4] [6]). Together with researchers at the University of New Mexico and the University of Maryland, we have built a series of zooming engines and created numerous examples of documents that utilize zooming. Over this timeframe other groups have also become interested in zooming and have built systems that take the zooming concept to new realms and applications (e.g. [38][39]). There are now several commercial implementations of user interfaces that use zooming (e.g. [28][22]). Paralleling these efforts, there is a growing body of research on multiscale algorithms - algorithms that allow information to be understood and manipulated over a large number of scales (e.g. [15] [20] [27]). More recently, our research colleagues at the University of Maryland have begun conducting user tests to determine in which situations zooming is most effective [5].

There are many examples of the use of "zooming" in human activities. For example, a person reading a newspaper brings the paper closer to read an article in which the reader is interested. Drawing the paper closer lets the reader focus more on a particular article - it allows them to comfortably read the small print, while eliminating distractions in the surrounding environment. When searching for an article to read, on the other hand, the reader holds the paper further away, so that the pages are easier to turn. The larger point sizes of article headings and sidebars facilitate this process. Scale is also used in information design. For example, schematic diagrams often include blow-ups that show a particular image at greater detail, or show a series of images to represent a zoom.

In previous research, we used zooming user interfaces primarily to create documents which model this detail+context metaphor – i.e. documents which present more detail as the user zooms in. This basic approach has been used for text documents, spreadsheets, file systems, calendars, and presentations. It is particularly effective for large hierarchical structures. One advantage of this zooming paradigm is that there is a strong sense of geography and object constancy.

In this paper, we extend zooming user interfaces to provide a novel style of graphical user interface (GUI) which we call Nested Components. We describe the Nested Components paradigm, illustrate that paradigm with an expression editor example, describe the toolkit used to create this example, mention related work and conclude with some notes on future directions.

NESTED COMPONENTS

Nested components offer two key contributions: Firstly, they provide a new approach for handling hierarchical control structures. Secondly, they offer a user-oriented mechanism for handling control structures that have a large branching factor – i.e. control hierarchies that offer many alternative controls situated at the same level in the hierarchy.

Our first observation is that many applications utilize a hierarchical control structure. For example, consider Microsoft Word. To edit the font of “heading” paragraphs in a document the user first opens the Style dialog box, then from within that the user accesses the Modify Style dialog box for the “heading” style, and from within that the user accesses the Font dialog box (see Figure 1). This multi-level nesting is typical of many modern applications.

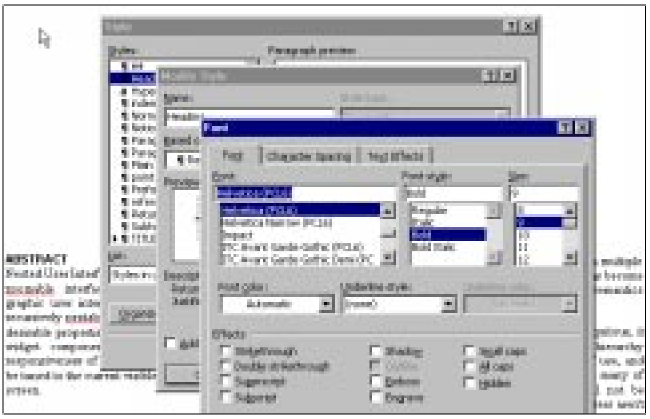

Figure 1: Multiple nesting dialog boxes used for editing the Font characteristics of “Heading” paragraphs.

To handle multi-level nesting in our Nested Components approach, the user is presented with a semantically zooming view of a set of controls. As the user zooms in on one area of the control structure, interface elements are displayed which offer a progressively greater level of control and detail. The user can then manipulate those controls, zoom in to gain access to even finer controls, or zoom out to perform a more coarse grain manipulation. This continuous style of interaction lets the user decide at what level they wish to work, without forcing them to interrupt their work in order to navigate between discrete windows.

Our second observation is that, in addition to hierarchical nesting, many applications also provide access to multiple controls at the same level in the hierarchy. For example, returning to the Font dialog box of Microsoft Word, we observe that the dialog box is divided into three distinct panels, accessed with tabs across the top of the dialog box labeled “Font”, “Character Spacing”, and “Animation”. These tabbed panels provide a convenient way of grouping related controls that are at the same nesting depth in the control hierarchy. They also allow the application to hide expert controls – making it easy for the user to access the commonly used features shown on the frontmost panel, while still giving access to the advanced or less frequently used controls tucked away on subsequent panels.

Our Nested Components approach offers a different mechanism for choosing between alternate controls at any given level in the control hierarchy. Initially we present the user with a default view of the controls for a particular level. We also provide buttons within that view which let the user customize the view in order to access more advanced controls. When the user activates one of those buttons, the more advanced controls are inserted in-situ, residing in the same geographical location in the control hierarchy, but at a smaller scale. We use an animation to perform this transition, so that the user retains a sense that they are editing the same object, only with a more advanced set of controls.

EXAMPLE: AN EXPRESSION EDITOR

To illustrate the use of Nested Components, we have created a tool for building and editing mathematical expressions. We chose mathematical expressions because they are hierarchical, they have a large branching factor, they are often difficult for novice users to create, and techniques that work for mathematical expressions are likely to generalize to other kinds of properties.

Many 2D and 3D graphics applications provide some mechanism for allowing the user to enter a parameter as a mathematical expression. The major difficulty in creating user interfaces for such parameters is that each parameter can be specified in many different ways (e.g. by using constants, keyframe curves, textual expressions, or a combined approach). Yet we want to present the user with a clear and uniform interface. Typically, the user must either type in a mathematical expression (hard for non-mathematicians), or select from a small set of pre-defined expressions (limiting), or pop up a visual graph editor (not as flexible as full mathematical expressions).

By utilizing Nested Components, we can allow the user to select from many different representations for a parameter, and yet still place the selected representation in the same visual slot in the user interface, at the appropriate scale. For example, the same visual slot could contain either a simple slider, or a keyframe graph (shown at a smaller scale). The user then zooms in or out to obtain a useful working view of the parameter. We can even allow the user to build entire hierarchies in-situ. The result is a visual tool that offers the same flexibility as a textual expression representation, but also many of the advantages of a visual graph editor.

Use of Nesting

In our expression editing system, compound expressions are represented using simple visual forms. For constants, the visual form consists of an interactive slider widget (see Figure 2). For keyframe curves, the form consists of a simple keyframe curve editor widget.

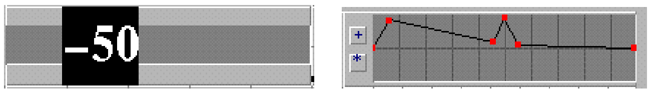

Figure 2: Basic visual components: a slider on the left for entering numeric constants and a keyframe curve editor on the right for drawing two dimensional function curves.

For compound binary expressions, the form is:

NAME	INPUT
MODIFIER	RESULT

In this visual form, the top-left NAME slot shows the name of the operator that is being applied. The INPUT slot shows contains a nested component representing the original value that the operator is modifying (either a keyframe curve, a slider, or a nested binary expression). The MODIFIER slot shows a nested component representing the value being used as the modifier (also either a keyframe curve, a constant, or a nested expression). The RESULT slot shows a visual display of the result of applying the operator to INPUT with MODIFIER as an operand.

For example, if an offset operator is being applied to a keyframe curve, INPUT is the original curve, MODIFIER is the translation factor, and RESULT shows the functional sum of INPUT and MODIFIER (Figure 3).

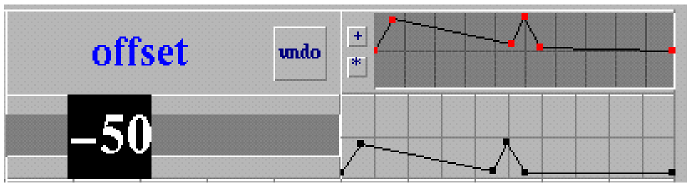

Figure 3: The “offset” binary operator, which in this case is taking an input curve (shown in the top right INPUT slot), translating it by –50, and displaying the resulting curve in the bottom right OUTPUT slot.

The user can directly edit either the value of the input or the modifier (by interacting with the slider or moving curve control points respectively). Alternatively, they can remove the offset operation by clicking on the “undo” button shown in the top-right NAME slot. This removes the binary operator and substitutes it with the nested expression in the INPUT slot. The user can also apply further operations to either the INPUT or the MODIFIER slots, substituting either value with a nested expression. Finally, the user can apply an operator to the result of the expression (Figure 4).

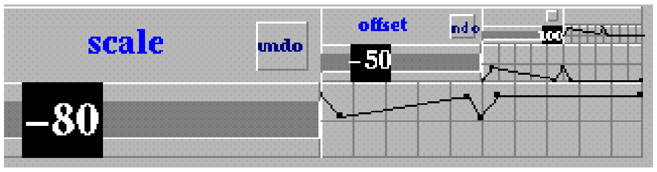

Figure 4: The “scale” binary operator. This operator is being applied to the same keyframe curve from Figure 3. You can see the offset expression nested in the INPUT slot in the top right corner. The scale operator is scaling the translated curve by a factor of –80%. The resulting inverted curve is displayed in the bottom right OUTPUT slot.

Use of Animation

The system uses smooth transition animations whenever elements of the expression are inserted or deleted, so that the user always sees a visually continuous view of the interface changing. During the transition for creating a binary expression, the user sees the original component smoothly shrink to the top right (the “input” slot), while the new components for the expression grow into place. If the user selects undo to remove a binary expression, then this visual transition repeats in reverse. Using animation helps cue the user to the roles of the new visual components. It helps users maintain relations between states in an application, as described by Robertson and his colleagues:

> "Interactive animation is used to shift some of the user's cognitive load to the human perceptual system. ... The perceptual phenomenon of object constancy enables the user to track substructure relationships without thinking about it. When the animation is completed, no time is needed for reassimilation." [31, p. 191].

Researchers including Robertson have demonstrated through informal usability studies that animation can improve subjective user satisfaction. There are a number of advocates of animation in user interfaces, e.g. [6] [24].

Use of Zooming

The most deeply nested controls shown in Figure 4 are too small to allow easy edition and manipulation. To solve this problem, a simple one-click zooming navigation mechanism is used. In our Nested Components implementation, clicking on a component causes the system to zoom and position that component so that it almost fills the view. Clicking on the edge of the view zooms out so you can see the enclosing hierarchy. It is also possible to pan and zoom manually using the mouse to obtain customized views. All traversals are animated.

In real-world applications, expressions are entered as parameters to control specific attributes of a mathematical model. To illustrate this, we have developed a prototype of a Nested Components interface for a 3D character animation system. Figure 5 shows a sequence of screen snapshots as the user zooms into a control panel for the actor "George", and then into the controls for George's pelvis properties. This series of static screen images does not capture the compelling effect of continuous animated zooming. When viewed on screen, the connection between each view is strikingly apparent. Once the user has zoomed in close enough for the parameter values to become important, they appear. The user can interact with the sliders and keyframe graphs to edit the parameters.

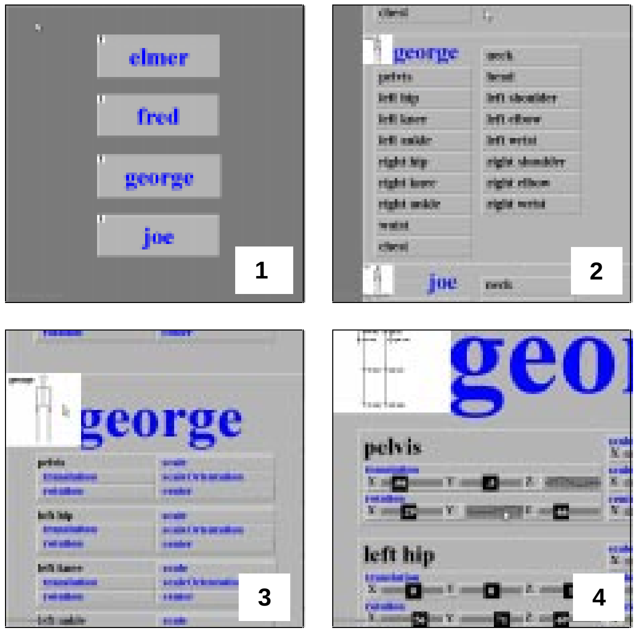

Figure 5: A sequence of screen snapshots showing a zoom into the properties for the “George” character in a 3D computer animation system. As the user zooms in, more details become visible.

Whether the user chooses to switch to curve editing or not, the result will play the same semantic role in the surrounding system: specifying transform attributes for this joint. In the GUI this is reflected in the fact that the curve fits into the same visual field as did the slider. From a user’s perspective, this is logical: when a parameter value is used, how it was created becomes less important than how it fits into the larger system.

A primary advantage of the Nested Interfaces approach is that the user can modify the expression at any level very simply. The user can zoom in or out to see the expression at different levels, and also edit, insert or modify the expression at any level. The screen is always updated to show the effect of the modifications both visually and numerically.

Use of Multiple Views

For complex control hierarchies, it is necessary to be able to view controls for multiple areas of the hierarchy at once. The user may need to compare the same property for two different objects, or to be modify a property on a fine-grain level but still be able to access higher level details, or to make a change in one part of the hierarchy and view the results in another.

To support this, we allow the user to create multiple views onto the same control space. See Figure 6. This enables the user to be controlling two or more different parameters at once, from multiple windows, each looking at a different part of the control space. Using a simple “bookmark” mechanism, users can quickly navigate to predefined views.

This lets users treat Nested Components in a more traditional multiple windows style, or alternatively use the novel features offered by the Nested Components paradigm.

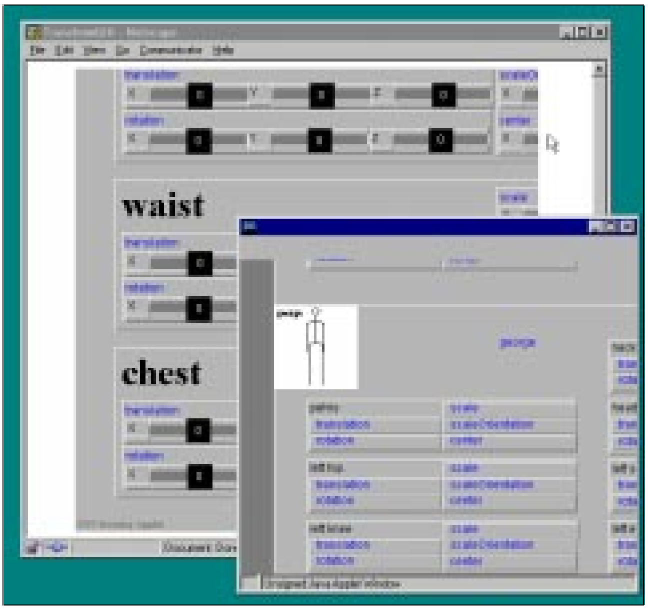

Figure 6: The system supports multiple views onto the same set of properties, so the user can look at one part of the control hierarchy and edit another.

ARCHITECTURE

The Nested Components system is implemented in Java 1.0 and consists of three layers. The bottom layer provides graphics support. The middle layer is a component object model. The topmost layer is a library of application widgets.

The bottom graphics layer consists of a wrapper class placed around the java.awt.Graphics drawing class. For each method in that class, we implement a corresponding method which takes floating point arguments, and internally does scaling and translation before calling the underlying method. In addition, we add methods to set the scale and offset for the context. In this way, we are able to use the Java 1.0 display methods included with current Web browsers. In the interest of responsiveness, we temporarily disable the display of text during zoom transition animations.

The middle layer is an extendable component object model, which is similar in many respects to a standard windowing toolkit, such as Java's AWT. In our implementation, each component object can serve as a container for child components; each such child has both an offset *and* a scale relative to its parent component, thereby defining its own local coordinate system (standard GUI toolkits provide an offset but not a scale). Each compound component is responsible for laying out its children. The toolkit provides methods to animate both layout changes and view changes. Components have extendable methods for rendering themselves and for handling events. Components are aware of the size at which they are being rendered, so they can provide appropriate rendering and event handling algorithms. For example, the graph editor widget may have several different appearances, from a very simple iconic representation to a much more complex representation which includes edit handles, grid lines and axes. Components with children can control what view magnification those children become visible at.

Mouse events begin at the innermost component visible at the screen pixel under the cursor, and move outward: if a component does not handle an event, the event is then routed to that component's parent. The default method given to components in response to a mouse click is ``zoom-transition the view so that you nearly fill the visible window." In this way, we are able to use containers for navigating in and out of detailed views.

The topmost layer in our implementation is a library that implements a useful variety of interface widgets, including buttons, labels, sliders, text editors, containers and curve editors. A subset of these are shown in the example figures. Application developers are free to add their own widgets to extend this set by extending the Component class.

RELATED WORK

Many research and product groups have explored the problem of how to create easy-to-use user interfaces that give the user more control but stay within the confines of a fixed-size display screen. In this section we highlight several alternative approaches to controlling objects.

Visual Programming

The visual programming community has investigated alternative models for visualizing the properties of large collections of data and objects. For example, the Alternate Reality Kit (ARK) [33] was an early system that provided users with the ability to call up a property box on any of the objects located on a large panning surface. Users could control general aspects of each object such as its response to physical forces (gravity, friction, etc.), as well as object-specific properties. However, there was no support in ARK filtering or organizing which controls were displayed for a given object. The Self system [34] has a more complex object editor. It provides a full instance-based programming environment and user interface builder. Users can call up property boxes on any object, and select from a number of property sheets. Self supports two views of these property boxes - a zoomed out view and a 1:1 full scale view. However, the interface panels in Self only operate at the full-scale view, and Self does not support nested property boxes.

Travers [36] describes a recursive prototype-based visual programming system in which objects are presented within rectangular windows on the screen. The user can open up child windows that hold details of sub-objects (and meta properties) within the parent object. The detail windows are presented within the parent window's frame. This approach lets users access increasing levels of detail on a given object. However, navigating between levels is slow, since the user is left to resize and reshape windows manually when more detail windows are opened. In our approach, we propose to use automated morphing and animated zooms to move between levels.

Data Visualization Systems

Many commercial systems make it is possible to pan and zoom over data visualizations. For example, AVS [37] and Khorus [30] provide many visualization widgets and tools, and also support a limited degree of visual programming. Database query systems such as DataSplash [9], combined with extensions such as VIQING [25], make it possible to perform database queries in a visual environment. However, these systems do not support views containing hierarchical nested user interface components.

In-Situ Controls

There have been a number of studies of "in-situ" interfaces. For example, Bederson et al. [3] explored local tools. In their approach, a collection of local tools sit on the work-surface along with the data. Each tool can be picked up (at which point, the cursor changes into the tool icon), used, and then put down anywhere on the worksurface. In [10], the authors propose a two handed user interface utilizing translucent tool palettes with push-through tools. One hand controls the location and orientation of the tool palette. The other is used for selection and drawing. This approach allows the user to work with tools and objects within the same space. Both of these approaches offer an interesting alternative to the popup property box, and may become valuable techniques for addressing the increasing screen-density problem.

Gestural Systems

In gestural, pen based systems (e.g. [16], [19], and the USRobotics Palm Pilot, to name a few), the user can interact with objects using gestures. Citrin [8] examined the effectiveness of gesture-based systems compared with palette-based systems, and found that users of gesture-based systems like the fact that they don't have to return to the tool palette each time they want to select a new tool. In our work we are interested in how nested interfaces and gestural approaches may complement each other. For example, marking menus [17] provide rapid access to large numbers of commands. We propose to integrate marking menus with our nested interface panels to quickly access specific levels of control.

CONCLUSION

This paper presents a new paradigm for managing hierarchical control structures. The paradigm is illustrated by a novel tool for entering and editing mathematical expressions. The tool is based on interactive widgets, and uses zooming to navigate nested hierarchies of user interface controls. The user can build new hierarchies in-situ to create arbitrary expressions.

The resulting system of recursively Nesting Components has a number of desirable properties. The level of detail of the view of any widget component and its children, as well as the responsiveness of that component to the user's actions, can be tuned to the current visible size of that component on the screen.

Expressions of components can easily be created and nested in-situ by the user upon the interface's virtual surface; the creation of such structures is accompanied by transition animations, which help the user to maintain a coherent view of the various semantic relationships. This produces a key feature of the approach: the distinction between the interaction style of a component and the semantic result that it produces; only a parameter's semantic influence on its surrounding context determines its geographic location in the interface. We have applied this approach to large and layered control problems, which the user can view as a single cohesive and easily navigable visual surface.

We have not yet carried out formal studies of this tool. However, informal feedback is encouraging. We plan to extend the tool to support more kinds of expressions, and also other semantic structures that readily lend themselves to hierarchical interaction. We would also like to explore nested component interfaces for other control hierarchies, such as editing the properties of a document or modifying a database or a program.

ACKNOWLEDGMENTS

Thanks to Clilly Castiglia and Noah Wardrip Fruin for editing and production assistance. Thank also to Ben Bederson and members of the Pad++ consortium for many enjoyable conversations and insights.

REFERENCES

1. [Bederson 94] Bederson, B. B., & Hollan, J. D. "Pad++: A zooming graphical interface for exploring alternate interface physics," *Proceedings of User Interface Software and Technology (UIST '94)* 17–26, New York: ACM. 1994.

2. [Bederson 96a] Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., & Furnas, G. "Pad++: A zoomable graphical sketchpad for exploring alternate interface physics," *Journal of Visual Languages and Computing*, 7, 3–31. 1996.

3. [Bederson 96b] Bederson, B. B., Hollan, J. D., Druin, A., Stewart, J., Rogers, D., and Proft, D., "Local Tools: An Alternative to Tool Palettes," *Proceedings of User Interface Software and Technology (UIST '96)* 169–170, New York: ACM. 1996.

4. [Bederson 97] Bederson, B.B, Hollan, J. D., Stewart,

J., Rogers, D., Druin, A., Vick, D., Ring, L., Grose, E., Forsythe, C., "A Zooming Web Browser," *Human Factors and Web Development*, Eds.: Forsythe, C., Ratner, J., and Grose, E., New Jersey: Lawrence Earlbaum, 1997.

5. [Bederson 98] Bederson, B., Boltman, A., "Does Animation Help Users Build Mental Maps of Spatial Information", submitted to CHI '99.

6. [Bederson 99] Bederson, B., McAlister, B., "Jazz: A Java Zooming Toolkit", CS-TR-4015, UMIACS-TR-99-24, May 1999. http://www.cs.umd.edu/hcil/jazz. Contact bederson@cs.umd.edu for details.

7. [Burnett 94] Burnett, M., Ambler, A., Interactive visual data abstraction in a declarative visual programming language, Journal of Visual Languages and Computing 5(1), pp.29-60, 1994.

8. [Citrin 93] Citrin, W., "Requirements for Graphical Front Ends for Visual Languages", *IEEE Symposium on Visual Languages*, pp. 142-150, Bergen, Norway, August 1993.

9. [DataSplash 98] The Tioga Database Visualization Research Group, "Tioga Project Home Page: http://datasplash.cs.berkeley.edu", Electronics Research Lab, University of California at Berkeley, Berkeley, California, 1998.

10. [Fitzmaurice 97] Fitzmaurice, G., Baudel, T., Kurtenback, G., Buxton, B., A GUI Paradigm Using Tablets, Two-hands and Transparency, in Formal Video Program, Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems v2, p.212-213, 1997

11. [Fox 97] Fox, D., "Tab: The Tabula Rasa zooming user interface system". URL http://www.cat.nyu.edu/fox/tab.html, New York University, 1997.

12. [Furnas 86] Furnas, G., Generalized fisheye views, in Proceedings of CHI '86, pp. 16-23, 1986.

13. [Furnas 95] Furnas, G. W. and Bederson, B. B. "Space-Scale Diagrams: Understanding Multiscale Interfaces," *Proceedings of Human Factors in Computing Systems (CHI'95)*, 234–241, New York: ACM. 1995.

14. [Furnas 97] Furnas, G., Effective view navigation, in Proceedings of CHI '95, pp.367-374, 1997.

15. [Hoppe 96] Hoppe, H., Progressive Meshes. Proceeedings of SIGGRAPH' 96, pp. 99-108, 1996.

16. [Kramer 94] Kramer, A., "Translucent Patches - Dissolving Windows", *Symposium on User Interface Software & Technology (UIST '94)*, pp. 121-130, ACM, New York, 1994.

17. [Kurtenbach 93] Kurtenbach, G., Sellen, A., Buxton, W., An Empirical Evaluation of Some Articulatory and Cognitive Aspects of Marking Menus Articles, Human-Computer Interaction v.8 n.1 p.1-23, 1993.

18. [Lamping 95] Lamping, J., Rao, R., Pirolli, P., A focus+context technique based on hyperbolic geometry for visualizing large hierarchies, in Proceedings of CHI '95, pp. 401-408, 1995.

19. [Landay 96] Landay, J., Interactive Sketching for the Early Stages of User Interface Design, Ph.D. Thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA., 1996.

20. [Lounsbery 97] Lounsbery, M., DeRose, T.D., Warren, J., Multiresolution Analysis for Surfaces of Arbitrary Topological Type, ACM Transactions on Graphics, vol. 16, no. 1, pp. 34-73. 1997

21. [Mackinlay 91] Mackinlay, J., Robertson, G., Card, S., The Perspective Wall: detail and context smoothly integrated, in Proceedings of CHI '91, pp. 173-179, 1991.

22. [Merzcom 97] Merzcom, Inc., `http://www.merzcom.com`, 1997.

23. [Misue 91] Misue, K., Sugiyama, K., Multi-Viewpoint Perspective Display Methods: Formulation and Application to Compound Graphs. In Proceedings of 4th International Conference on Human Computer Interaction, vol 2, pp. 834-838, 1991.

24. [Myers 96] Myers, B., Miller, R., McDaniel, R., Ferrency, A., Easily adding animations to interfaces using constraints, in Proceedings of UIST '96, 1996.

25. [Olson 98] Olson, C., Stonebraker, M., Aiken, A., Hellerstein, J., VIQING: Visual Interactive QueryING, , in Proceedings of VL '98, pp.162-169, 1998.

26. [Perlin 93] Perlin, K., & Fox, D. "Pad: An alternative approach to the computer interface," *Proceedings of Computer Graphics (SIGGRAPH '93)* 57–64, 1993, New York: ACM. 1993.

27. [Perlin 95a] Perlin, K., Velho, L., Live Paint: Painting with Procedural Multiscale Textures, in Computer Graphics; Vol. 28; No. 3. 1995.

28. [Perspecta 97] Perspecta, Inc., `http://www.perspecta.com`, 1997.

29. [Rao 94] Rao, R., Card, S., The Table Lens: merging graphical and symbolic representations in an

interactive focus + context visualization for tabular information, in Proceedings of CHI '94, pp.318-322, 1994.

30. [Rasure 92] Rasure, J., Young, M., "An Open Environment for Image Processing Software Development", The 1992 SPIE Symposium on Electronic Image Processing, San Jose, California, February 1992.

31. [Robertson 91] Robertson, G. G., Mackinlay, J. D., & Card, S. K. Cone Trees: Animated 3D Visualizations of Hierarchical Information. *In Proceedings of Human Factors in Computing Systems (CHI 91)* ACM Press, pp. 189-194. 1991.

32. [Shizuki 98] Shizuki, B., Toyoda, M., Shibayama, E., Takahashi, S.,Visual Patterns + Multi-Focus Fisheye View: An Automatic Scalable Visualization Technique of Data-Flow Visual Program Execution, in Proceedings of VL '98, pp. 270-27, 1998.

33. [Smith 87] Smith, R. B. "Experiences with the Alternate Reality Kit: An Example of the Tension Between Literalism and Magic," *Proceedings of the Human Factors in Computing Systems and Graphics Interface Conference*, 61–67, New York: ACM. 1987.

34. [Smith 95] Smith, R., Maloney, J., Ungar, D., The Self-4.0 user interface: manifesting a system-wide vision of concreteness, uniformity, and flexibility, in: OOPSLA '95. Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications, pages 47-60, 1995.

35. [Spenke 96] Spenke, M., Beilken, C., Berlage, T., FOCUS: the interactive table for product comparison and selection, in Proceedings of UIST '96, pp.41-50, 1996.

36. [Travers 94] Travers, M., Recursive Interfaces for Reactive Objects, Proceedings of ACM CHI'94, v.1 p.379-385, 1994.

37. [Upson 89] Upson, C., Faulhaber, T., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., VanDam, A., "The Application Visualization System: A Computational Environment for Scientific Visualization", IEEE Computer Graphics and Applications, 9(4), pp. 32-40, 1989.

38. [Woodruff 98a] Woodruff, A., Landay, J., Stonebraker, M., Constant Information Density in Zoomable Interfaces. Proceedings of Advanced Visual Interfaces '98, pages 57-65, L'Aquila, Italy, May 1998.

39. [Woodruff 98b] Woodruff, A., Landay, J., Stonebraker, M., Goal-Directed Zoom. In CHI '98 Summary, pages 305-6, Los Angeles, April 1998.