Hyper Mochi Sheet: A Predictive Focusing Interface for Navigating and Editing Nested Networks through a Multi-focus Distortion-Oriented View

Masashi Toyoda and Etsuya Shibayama
Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Oookayama, Meguro-ku,
Tokyo 152–8552 JAPAN
+81-3-5734-3870
{toyoda,etsuya}@is.titech.ac.jp

ABSTRACT

Multi-focus distortion-oriented views are useful in viewing large information on a small screen, but still have problems in managing multiple foci during editing. The user may have to navigate information space by focusing and defocusing multiple parts to obtain multi-focus layouts that change according to various editing situations. As a result, it becomes haphazard to navigate and edit large nested networks such as hypertexts. We propose a user interface for quickly obtaining desirable layouts. The interface uses two techniques: focus size prediction and predictive focus selection. These techniques are based on a user test and experiences in applications. We also describe two example applications.

Keywords

distortion-oriented view, multi-focus, editing, navigation

INTRODUCTION

Multi-focus distortion-oriented views [14, 10, 1, 4] are useful in viewing large information on a small screen. These views provide more flexible layouts of focused parts while preserving the overall context compared to single-focus distortion-oriented views[8, 13, 12, 7, 5]. As such they seemingly have the potential for scalable *editing* of large networks such as visual programs and hypertexts. In reality, however, they have not yet supported efficient editing interfaces of large networks due to problems in managing multiple foci. More flexibility and freedom of multi-focus layouts often require more work by the user than in single-focus views, in which a change of layout involves only focus movement and change in magnification factor. Rather, in multi-focus views, the user may have to perform boring focusing and defocusing operations on multiple parts of the screen to obtain a layout suitable for a particular editing situation, which changes frequently during editing.

Before discussing the problems in detail, we show our objective applications, which are editors that handle hierarchically nested networks with hyperlinks such as visual programs, hypertexts, and file systems. In Figures 1 and 2, we show two concrete applications.

Figure 1 shows the KLIEG visual programming environment [15], which addresses the scalability problem. KLIEG allows the programmer to edit multiple modules in one view and to construct nested data-flow networks for programming in the large. In Figure 1, there are four modules at the top level (**nqueens**, **combiners**, **master_worker_nqueens**, and **dispatchers**). In this case, to edit **master_worker_nqueens** referring **combiners** and **dispatchers**, the user magnifies these modules and shrinks **nqueens**. In addition, a program includes invisible hyperlinks from components to their definitions and documents, and a document is also a hypertext. When the user follows a hyperlink, KLIEG automatically focuses its destination and defocuses unnecessary foci. Using KLIEG, the user can easily drag-and-drop components between modules and can navigate a program with hyperlinks.

Figure 2 shows a novel presentation tool, which can handle hierarchically structured slides with hyperlinks. In effect, it can be used as a 2D visual outline processor for hypertexts. It allows the creator to edit a presentation through multi-focus views. It can also simultaneously show multiple slides and their overview during presentation. Each picture in Figure 2 is a different view of the same presentation. The top view is an overview, and the bottom view is a focus+context view in which a slide titled "Structure of Diagrams" is being focused. With a single mouse operation, the presenter can follow a hyperlink from a slide to the next one, and the system automatically moves the focus to the next slide and adjusts slide sizes. The creator of the presentation does not have to explicitly designate these sizes during editing, rather, the system predicts the sizes from a history of editing operations.

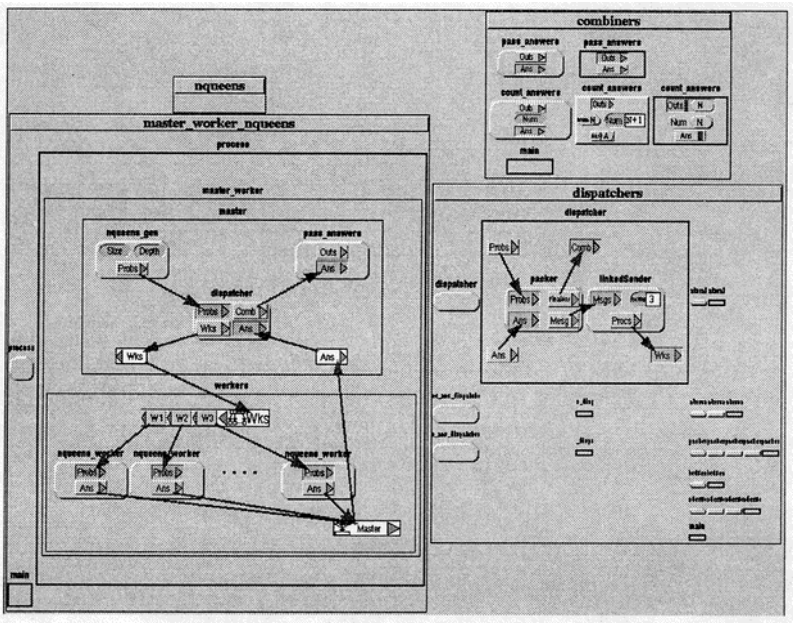

Figure 1: A visual programming environment KLIEG

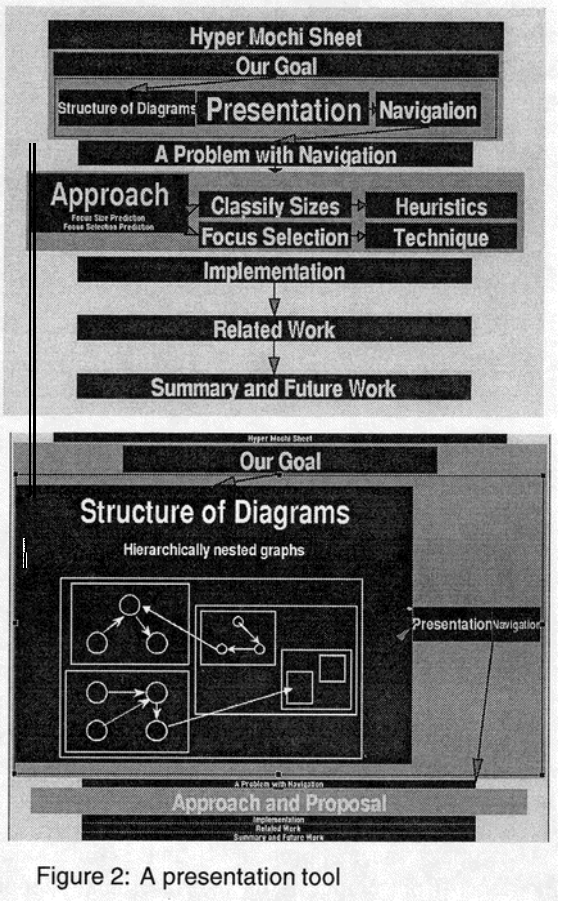

Figure 2: A presentation tool

To be more specific, our algorithm is intended to address the following problems:

- The user can resize a node to arbitrary sizes during editing, but it is tedious to resize the node for focusing and defocusing every time on editing its contents. It might be useful if the system allows the user to focus and defocus a node in simple operations. However, since a set of appropriate node sizes for focusing and defocusing is different from each node, it is a tedious task to explicitly designate these sizes for each node.
- A desirable layout after following a link may vary according to the current editing situation. When the user follows a hyperlink, it is always necessary to focus the destination of the link, but the source and other foci may or may not be necessary. On one hand, if the user still wants to edit the source, both the source and the destination should be focused. On the other hand, after the user finished editing the source, it is not necessary to retain the focus on the source.

To address these problems, we propose a user interface that allows the user to obtain easily desirable layouts for various editing situations. We implemented this interface as a library; Hyper Mochi Sheet. The interface uses the following predictive techniques.

- ***Focus*** size prediction automatically determines a pair of node size, one being used when the node is focused and the other being used when the node is defocused. It is not necessary for the user to explicitly set these sizes, rather, our technique predicts appropriate sizes of nodes using a history of editing commands.
- ***Predictive focus selection*** automatically selects necessary foci and discards unnecessary foci during navigation with hyperlinks. When the interface focuses and defocuses nodes, it uses sizes predicted by the focus size prediction. Since necessities of foci may depend on application semantics, Hyper Mochi Sheet provides a default focusing behavior that can be customized by the application programmer.

The next section discusses related work. Then we describe the basic interface of Hyper Mochi Sheet. This is followed by explanations of prediction techniques, and an evaluation. Finally, we conclude.

RELATED WORK

There have been various user interfaces that handle hierarchical networks with multi-focus distortion-oriented views [10, 14, 1, 4]. There have been, however, little research supporting automatic multi-focus management and hyperlink navigation. Our approach is a new attempt to support them using predictive techniques that are mainly used in PBD (programming-by-demonstration) systems such as [9, 6].

Layout-independent Fisheye View[10] shows an algorithm that can be used for navigating nested networks. However, it describes merely an algorithm and there are few discussions about the way to construct the user interface.

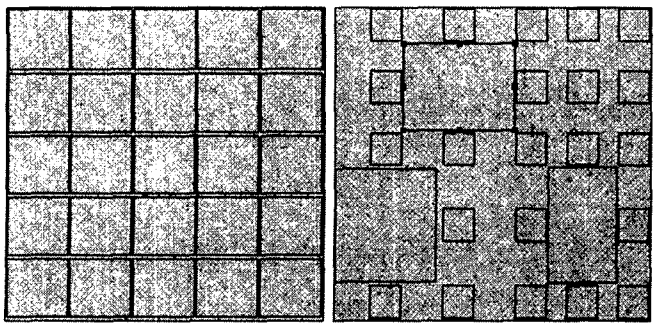

Figure 3: A distortion view in Hyper Mochi Sheet

The Continuous Zoom (CZ)[1] uses smooth animation during zooming, and supports efficient navigation through the hierarchy of a nested network. However, it neither supports navigation with hyperlinks nor editing graphs. The Intelligent Zoom[2] combines the CZ with intelligent supports in network supervisory control systems. It suggests opening (magnifying) a node in an alarm condition, and when the node is opened it automatically selects an appropriate representation from several aspects of the node such as a bar chart and a trend diagram. However, since it merely suggests opening and closing nodes, it does not reduce explicit focusing and defocusing.

The rubber sheet approach[14] and 3-dimensional pliable surface[4] put emphasis on interfaces that support flexible selection of a shape as a focal region. It is, however, difficult to access nodes deep in a hierarchy. The user must specify focal regions and stretch them repeatedly for accessing deep information. It is a tedious and boring task.

Pad++[3] is a single focus and pan/zoom based interface. It supports multiple foci by multiple windows called portals. Since it does not perform automatic portal management, the user has to create, delete, and arrange multiple portals manually. Pad++ also supports hyperlink navigation in a single focus view, but it does not address multi-focus issues in hyperlink navigation.

PBD systems, such as Metamouse[9] and Eager[6], predict operations that the user will perform next. These systems automatically extract patterns of recurring operations from a history, and create macros by generalizing these patterns. However, in our approach, the purpose is to reduce explicit designation of sizes and necessary foci. Therefore it is often necessary to predict operations that have never been performed.

BASIC INTERFACE OF HYPER MOCHI SHEET

To make distortion views, we use an approach similar to the Continuous Zoom[1]. Figure 3 displays an application of our approach to a 2D grid graph. When some nodes are magnified in the left view, it becomes impossible to display all nodes in their desirable sizes on the screen. In this case, all nodes are compressed uniformly in the horizontal and vertical directions keeping relative positions of nodes as the right view.

In addition to the continuous zoom algorithm, our algorithm avoids overlapping of nodes by simply aligning nodes in the horizontal and vertical directions during moving and resizing nodes. To use screen space more efficiently, it also meshes adjoining rows or columns together. For example, in the right view of Figure 3, two columns in the left are meshed together.

The user can focus and defocus nodes by stretching and shrinking them with handles that are shown as small black rectangles in Figure 3. The width and height of a node can be stretched independently to each direction. In addition the user can move nodes by dragging.

We also use semantic zooming [11], which changes an amount of information of a node according to its size. For example, in Figure 2, when a slide is small, we can see only its title. When a slide is large enough, we can see details of the slide.

FOCUS SIZE PREDICTION

Focus size prediction automatically determines a pair of node size[1] in the following.

- *Small size* is used when the node is defocused. The node area of this size is smaller than the large size. It is possible to edit inside roughly in this size.
- *Large size* is used when the node is focused. The user can edit inside details of the node. The node area of this size is larger than the small size.

The system predicts these sizes from a history of editing commands. It is not necessary for the user to explicitly set these sizes during editing. Once the small and large sizes of a node are determined, the user can easily select one of these sizes by clicking a mouse button or by using a popup-menu. Changes to the large size and the small size perform instant focusing and defocusing, respectively. These commands are useful when the user edits one node repeatedly and when the user navigates edited networks.

We do not provide any other intermediate sizes for the prediction, although they are useful in some situations. This decision simplifies size changing commands and makes the prediction easy but useful. Note that the determination of sizes is not trivial, because the user can resize nodes to arbitrary sizes in arbitrary orders during editing. For example, when the user stretches a node from its small size, it is difficult to distinguish whether the user want to modify its small size or its large size.

Preliminary User Test

We performed a preliminary user test to investigate when the user determines small and large sizes during editing. We use the editor in which the user must set these sizes of each node explicitly. By tracing command histories, we tried to find out typical sequences of commands around size setting.

[1] In the following, a size stands for a pair of width and height of a node.

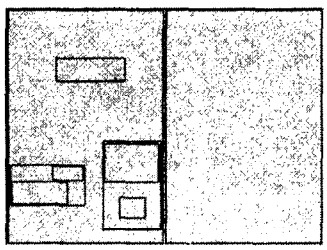

Figure 4: The diagram used in the user test: all rectangle sizes are set to small

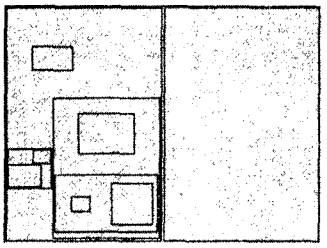

Figure 5: The diagram that shows the detailed view of the bottom-right rectangles

Table 1: Command Sequences around SetSmall

Initial size	Command Sequences around SetSmall	Subjects 1	2	3	4	5	6	7	8
small	Shrink+.**SetSmall**.except Shrink	5	6	5	5	3	5	9	2
	–	2	2	2	1	1	1		
small	Expand.**SetSmall**.Expand or Large	2	1	1	1	1	2	2	2
	Small	1	1	1					
	–				1	1	1		
large	Shrink+.**SetSmall**.Small							2	
	Large	1							
small	Expand.Shrink.**SetSmall**.Any						2		
Others								3	3

Shrink: resize to a smaller size, Expand: resize to a larger size
Small: change to the small size, Large: change to the large size
+: one or more execution of the command
–: the node was left

Table 2: Command Sequences around SetLarge

Initial size	Command Sequences around SetLarge	Subjects 1	2	3	4	5	6	7	8
small	*.Expand.**SetLarge**.Small	6	7	3	5	2	2	2	3
	Shrink					3			
large	Expand+.**SetLarge**.Small				1	1	6		
	Shrink+.**SetLarge**.Small				1		1	1	
Others		1		3		1	3	2	

*: alternative sequence of commands that may be empty

Method

- *System*: We used a simple editor for drawing nested nodes. The editor provides typical editing commands such as adding, removing, resizing, and moving nodes. For node size setting, it provides SetSmall and SetLarge commands that store the current node size as the small size and the large size, respectively. The editor also provides Small and Large commands for changing a node to the corresponding size.
- *Subjects*: Seven student volunteers and an instructor of computer science served as subjects in the user test. All subjects were familiar with typical window-based GUIs.
- *Task*: Subjects were required to draw a diagram, which is shown in the left hand side of Figure 4, on the right blank area, and to set small and large sizes of all nodes. This diagram consists of 13 nested rectangles[2], and sizes of each node have been set. The default size of each node is its small size. Figure 5 shows the diagram in which bottom-right rectangles are changed to its large size. Each subject was instructed to set sizes immediately when he decided sizes, and to edit without hurry. In addition, we did not limit the time for the task.
- *Procedure*: Before performing the task, subjects were given an explanation of the system and a practice trial on a part of the diagram. We spent about 10 minutes on this session.

Result and Observations Tables 1 and 2 show patterns of command sequences around SetSmall and SetLarge respectively, and the number of times each pattern was used by each subject. A pattern begins when the node was in its small or large size after its creation[3] or changing its size. This is the initial size in the pattern and is followed by a command sequence performed on the node before the execution of set command. The pattern also includes a command performed after the set command.

We considered only resize related commands such as resizes (Shrink and Expand) and size changes (Small and Large), because we could not find distinctive regularity from other commands. Note that we treated consecutive resizes on a single node as a single resize command, since such a sequence stands for a fine tuning of the size.

We show observations of the result in the following.

1. SetSmall occurs after repeated Shrink from the small size in most cases (See the first pattern in Table 1).
2. SetSmall also occurs after a single Expand from the small size, and is mostly followed by Expand or Large (See the second pattern in Table 1). In some cases, SetSmall is followed by Small, but SetLarge occurs more frequently between Expand and Small (See the first pattern in Table 2).
3. SetLarge mostly occurs after Expand and before Small (See Table 2). Both SetSmall and SetLarge may occur after Shrink from large size and before Small (See the third pattern in Table 1).
4. Among six SetLarge commands performed by the subject 5, three of them occur before Shrink. The subject first set the large size then shrunk and set the small size, though most subjects set the small size first. In the fourth pattern in Table 1, we can see this sequence before SetSmall.

Prediction Algorithm

Based on the above observations, we designed and implemented a size prediction algorithm. Our design policies are (1) to give a higher priority to patterns used by most subjects, (2) to satisfy subjects as fairly as possible, and (3) to keep the algorithm simple.

[2]Some nodes are not displayed in Figure 4, because their parent nodes are too small

[3]A node is in small size at the creation time

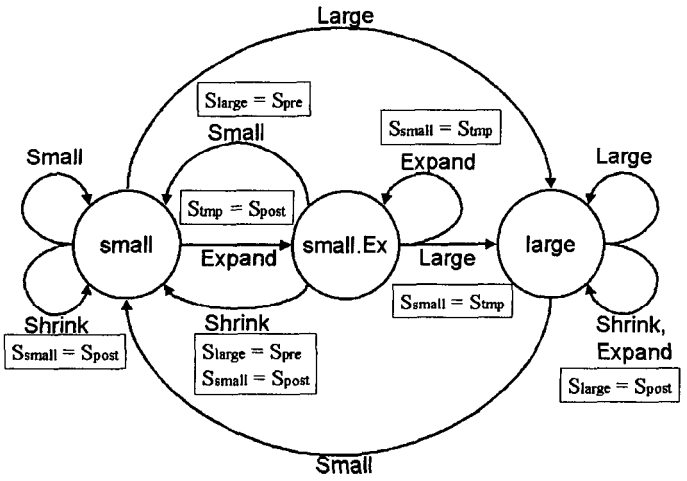

Figure 6: A state transition chart for predicting size

The algorithm is based on state transitions on each node that are shown in Figure 6. The state is changed when the user performs a command on the node, and a label on an arrow represents the command. There are three states: small, small.Ex, and large. The small and large states represent that the node is in the corresponding sizes, and small.Ex represents that the node has been expanded repeatedly from the small size. The small.Ex state is necessary, since the size is uncertain when the node is expanded from the small size (See the observation 2).

When a transition occurs, the small and large sizes (S_{small} and S_{large}) may be changed. In Figure 6, rectangles include actions performed after the transition. S_{pre} and S_{post} represent the sizes before and after the transition, respectively, and S_{tmp} represents the temporal store of a size. We describe the reason for each action in the following.

- *Shrink from small*: According to the observation 1, S_{small} is changed to S_{post}.
- *Expand from small*: Since S_{post} may be either size, S_{post} is stored temporary into S_{tmp}.
- *Expand or Large from small.Ex*: According to the observation 2, S_{small} is changed to S_{tmp}.
- *Small from small.Ex*: According to the observation 3, S_{large} is changed to S_{pre}.
- *Shrink from small.Ex*: We don't ignore the observation 4 to satisfy subjects fairly (This is policy 2). In fact, there are few conflicts with other observation. In this case, S_{large} and S_{small} are changed to S_{pre} and S_{post}, respectively.
- *Shrink or Expand from large*: According to the observation 3, S_{post} may be the small size in the case after Shrink from large. In this algorithm, S_{large} is changed to S_{post}, because there are three subjects who used SetLarge and is only one who used SetSmall. This decision follows the design policies 1 and 3.

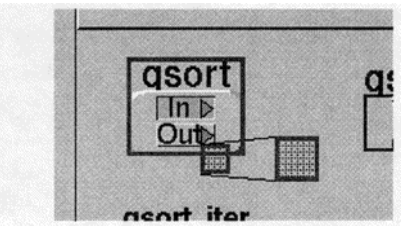

Figure 7: Size correction interface

Size Correction Interface

Since prediction may be error-prone, manual correction is necessary. We provide an interface to correct a size by choosing a size from the size history of the node. When the user performs a size changing command on a node, two buttons appear near the node (Figure 7). The smaller button changes the size to the next smaller size in the history and the larger button the next larger size. If the predicted size is acceptable, the user can ignore these buttons. This interface allows the user to correct size precisely to a past size rather than using handle interface.

PREDICTIVE FOCUS SELECTION

During navigation with hyperlinks, the system predicts foci that will be unnecessary, and automatically discards these foci. Focus selection enables the user to obtain almost desirable layout only by following hyperlinks.

Hyperlink Navigation Examples

As an example, we show a simple navigation using a presentation tool in Figure 8. In case of viewing slides one after another, the focus on the current slide will become unnecessary when the user follows a hyperlink. In Figure 8 (b), the system automatically shrinks the title slide when the user follows the hyperlink to the slide "Our Goal." In Figure 8 (c), the slide "Our Goal" is shrunken in the same way.

Figure 9 shows another navigation example in a visual programming environment. In case of editing visual programs, it is necessary to retain foci on nodes that are in the middle of editing. In Figure 9 (a), the user is editing a data-flow diagram **master** at the center of the bottom-left module, and intends to check the behavior of the **pass-answers** component by following the hyperlink to its definition part. In this case, the system can predict that **master** is still necessary because there are unconnected components in the network. Therefore, the system retains the focus on **master** when the user follows the hyperlink (Figure 9 (b)).

Prediction Method

To realize such automatic focus management, Hyper Mochi Sheet library provides each node with a boolean function f(F), which returns true if the focused (large size) node *F* is still necessary. Programmers can reflect application semantics in their applications by defining customized f(F) for each node. For example, in visual programming editor,

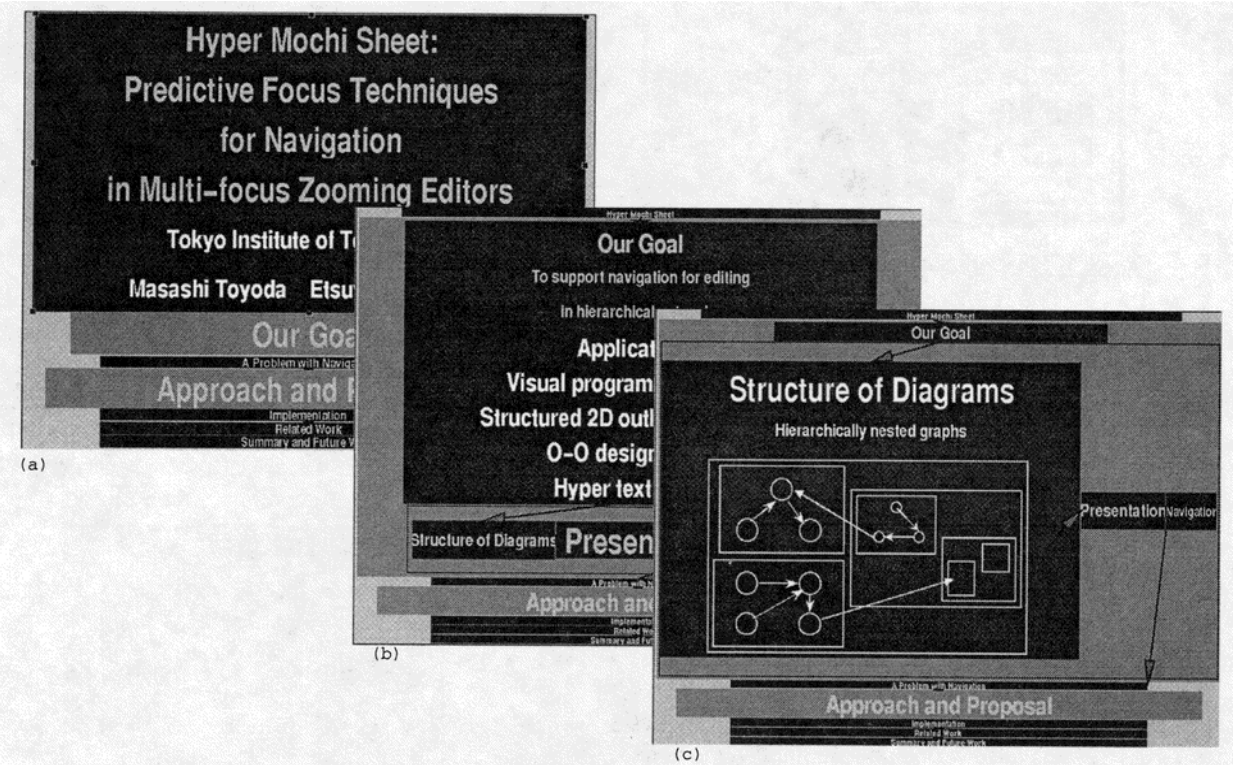

Figure 8: A hyperlink navigation in our presentation tool

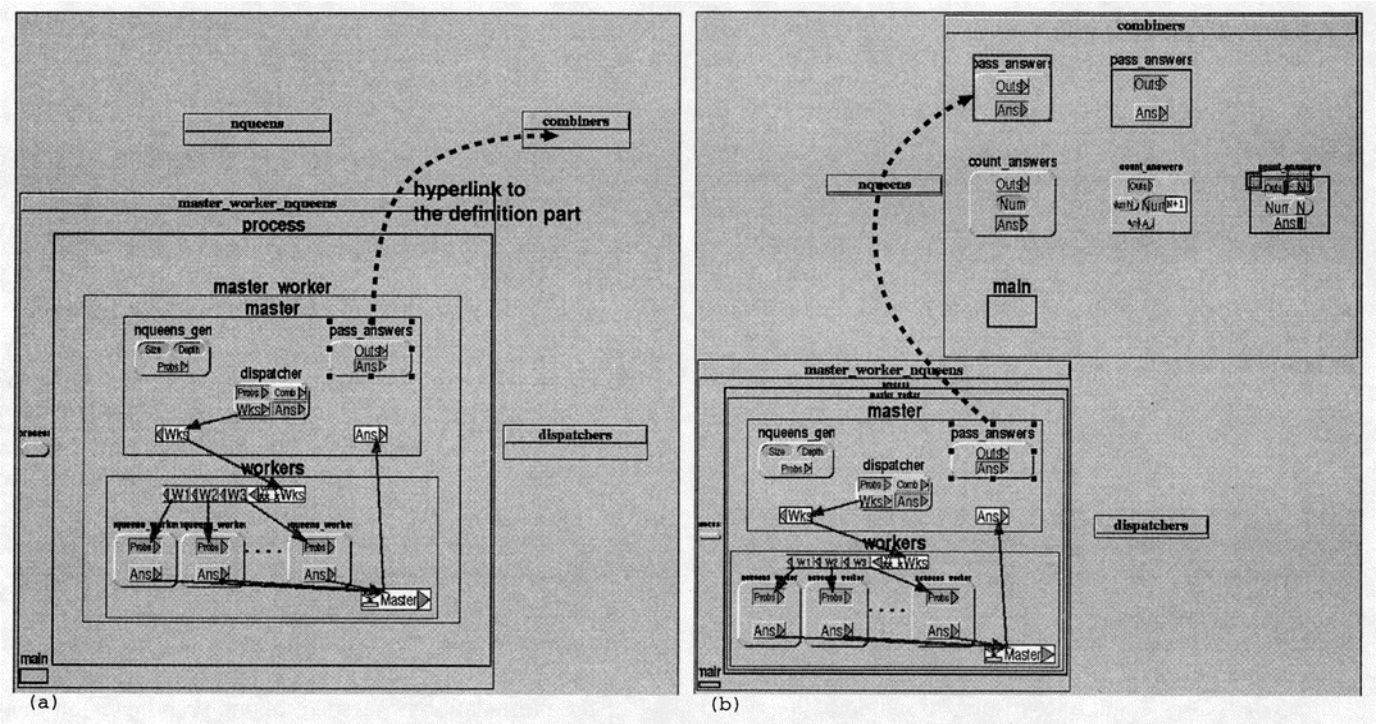

Figure 9: A hyperlink navigation in a visual programming environment

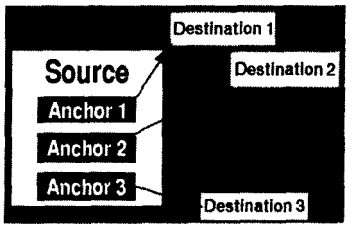

Figure 10: Hyperlink structure

f(F) returns true if there exist unconnected ports in the node *F*. The default implementation of f(F) returns false if all child nodes in the *F* are in the small size.

When the user follows a hyperlink from an anchor (See Figure IO), the system changes the size of the destination to its large size, and stores the source and the destination in the focus *list*. Simultaneously, the system changes the sizes of all the ancestor nodes of the destination to their large size in parent-to-child order, so that the destination will be visible. After magnifying the destination, the system checks whether each node except the destination in the focus list satisfies *f*. If *f* returns false with a node in the list, the node size is changed to its small size. Then the system changes the sizes of ancestors in child-to-parent order. Before changing the size of an ancestor *A*, the system checks *f(A)*. If *f(A)* returns false, A is changed to its small size, and if not, the system stops changing sizes of upper ancestors.

In addition, the system animates transition from one layout to another, so that the user is not confused even if the layout drastically changes during navigation.

EVALUATION

In this section, we describe an experiment to evaluate the feasibility of the focus size prediction technique. We leave evaluations of the predictive focus selection and integration of two techniques for future work because of difficulties that are caused by their application dependent characteristics. The number of implemented applications is not enough to formally evaluate all techniques, even though we consider that they seem fine so far.

Method

- *System*: We used a simple presentation editor with the focus size prediction function. Differences from the editor in the preliminary user test are that a node has one line editable text inside, and that the editor does not provide size setting commands (SetSmall and SetLarge). A text in a node is not displayed when the node has child nodes and the node is large enough[4] to display its children.
- *Subjects:* Ten student volunteers served as subjects. Four of them were also ones of the preliminary user test. All subjects were familiar with typical window-based GUIs.
- *Task:* Subjects were required to edit a simple presentation based on the table of contents shown in Figure 11. Each subject was instructed (1) to represent the presentation hierarchy as nested nodes like Figure 12, (2) to put some empty text boxes as contents of each leaf section such as "1 Introduction" and "2.2.1 Basic Usage," (3) to arrange nodes as you like, and (4) to edit without hurry and we did not limit the time for the task, In addition, we did not force for subjects to check node sizes during editing.
- Procedure: Before performing the task, subjects 'were given an explanation of the system and a practice trial on a part of the presentation. We spent about 10 minutes on this session. After each subject performed task, we checked whether sizes of each section are along to the subject's intention. In this session, we asked subjects about correctness of sizes using Large and Small commands..

[4] A node is large enough if the width and height of the node are larger than 70 pixels.

Figure 11: The table of contents used in the experiment

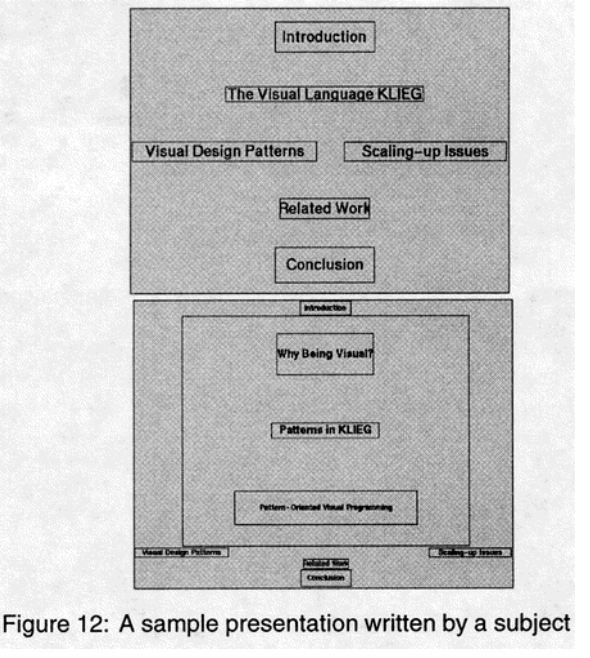

Figure 12: A sample presentation written by a subject

Table 3: The number of use of the size correction interface and the number of prediction errors checked after the task

		Subjects									
		1	2	3	4	5	6	7	8	9	10
# of size corrections on 17 sections	Small	1	4	1	0	3	1	0	3	4	3
	Large	1	0	1	0	2	1	0	4	0	0
# of errors in 17 sections	Small	2	0	0	0	1	0	0	0	0	0
	Large	2	0	3	0	0	3	3	0	3	3
total	Small	3	4	1	0	4	1	0	3	4	3
	Large	3	0	4	0	2	4	3	4	3	3

Result and Discussion

Table 3 shows the number of the use of the size correction interface, and the number of prediction errors. The use of the size correction interface means that a subject found and corrected a wrong node size, which was not along to the subject's intention, during editing. An error was counted when a wrong node size was found during the check session after the task. Each number was counted for each size. Subjects 1 to 4 were also ones of the preliminary user test, but there were no significant differences in the result from other subjects.

In spite of the fixed algorithm, error rates are significantly small. The average error ratio after the task is 6% (the best is 0% and the worst is 11%). Even in total error ratio, the average is only 14% and the worst is 20%.

Note that the prediction algorithm almost suits all subjects, though they edited the presentation in various manners. Some subjects resized nodes without using Small and Large command, and some subjects used Small and Large command on about half of the nodes. In addition, some subjects first decided a large size of a node, and other subjects decided a small size first.

CONCLUSION

We have proposed two prediction techniques for managing multiple foci of distortion-oriented views during navigation and editing nested networks. The focus size prediction automatically determines appropriate sizes of nodes. We showed reasonable accuracy of this technique with an experiment. The predictive focus selection automatically defocuses unnecessary foci during navigation with hyperlinks. Application programmers can reduce the error rate of this technique by customizing prediction methods for their applications.

It is shown that our techniques are useful for a visual programming editor and a presentation tool. We believe that the techniques can be applied to other applications, such as hypertexts, file systems, and object-oriented software designs, with appropriate heuristics. We plan to implement these applications and to construct a framework that allows the programmer to introduce more application semantics as prediction keys.

ACKNOWLEDGEMENTS

We would like to thank Satoshi Matsuoka, Shin Takahashi, and the TRIP meeting members for their helpful advice. We also thank our test users for their participation.

REFERENCES

1. L. Bartram, A. Ho, J. Dill, and F. Henigman. The Continuous Zoom: A Constrained Fisheye Technique for Viewing and Navigating Large Information Space. In *Proceedings of UIST '95*, pages 207–215, November 1995.
2. L. Bartram, R. Ovans, J. Dill, M. Dyck, A. Ho, and W. S. Havens. Contextual Assistance in User Interfaces to Complex, Time Critical Systems: The Intelligent Zoom. In *Graphics Interface '94*, pages 216–224, 1994.
3. B. B. Bederson and J. D. Hollan. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. In *Proceedings of UIST '94*, pages 17–26, November 1994.
4. M. S. T. Carpendale, D. J. Cowperthwaite, and F. D. Fracchia. 3-Dimensional Pliable Surfaces: For the Effective Presentation of Visual Information. In *Proceedings of UIST '95*, pages 217–226, November 1995.
5. W. Citrin and C. Santiago. Incorporating Fisheying into a Visual Programming Environment. In *Proc. 1996 IEEE Symposium on Visual Languages*, pages 20–27, 1996.
6. A. Cypher. EAGER: Programming Repetitive Tasks by Example. In *Proceedings of ACM CHI'91*, pages 33–39, April 1991.
7. J. Lamping and R. Rao. Laying out and Visualizing Large Trees Using a Hyperbolic Space. In *Proceedings of UIST '94*, pages 13–14, November 1994.
8. J. D. Mackinlay, G. G. Robertson, and S. K. Card. The Perspective Wall: Detail and Context Smoothly Integrated. In *Proceedings of ACM CHI'91*, pages 173–179, 1991.
9. D. L. Maulsby, I. H. Witten, and K. A. Kittlitz. Metamouse: Specifying Graphical Procedures by Example. In *Proceedings of SIGGRAPH '89*, volume 23, pages 127–136, July 1989.
10. E. G. Noik. Exploring Large Hyperdocuments: Fisheye Views of Nested Networks. In *ACM Conference on Hypertext and Hypermedia*, pages 14–18, 1993.
11. K. Perlin and D. Fox. Pad: An Alternative Approach to the Computer Interface. In *SIGGRAPH 93 Conference Proceedings*, pages 57–64, 1993.
12. G. G. Robertson and J. D. Mackinlay. The Document Lens. In *Proceedings of UIST '93*, pages 101–108, November 1993.
13. M. Sarkar and M. H. Brown. Graphical Fisheye Views of Graphs. In *Proceedings of ACM CHI'92*, pages 83–91, 1992.
14. M. Sarkar, S. S. Snibbe, O. J. Tversky, and S. P. Reiss. Streching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens. In *Proceedings of UIST '93*, pages 81–91, November 1993.
15. M. Toyoda, B. Shizuki, S. Takahashi, S. Matsuoka, and E. Shibayama. Supporting Design Patterns in a Visual Parallel Data-flow Programming Environment. In *Proc. 1997 IEEE Symposium on Visual Languages*, pages 76–83, September 1997.