The Role of Another Spatial Dimension in Software Visualization

HIDEKI KOIKE
University of Electro-Communications

The primary objective of this article is to demonstrate the use of 3D-computer graphics in visualizing shapeless software information by focusing on performance monitoring of parallel/concurrent computer systems. Issues are addressed from two different perspectives: expressiveness of output media and user cognition. The former describes the limitations of 2D output media. The latter refers to a user's cognitive load when using 2D representations in a multiple-window environment. We show how these problems can be minimized by using a 3D framework. A prototype visualization system called VOGUE has been developed. A 3D framework is used to visualize the execution pattern of two parallel/concurrent computer systems: an electric power control system and a parallel manipulator system. Through these visualizations, we show the effectiveness of our framework. The applications of 3D frameworks to other kinds of software information are also described.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming—*distributed programming*; *parallel programming*; D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; D.2.7 [**Software Engineering**]: Distribution and Maintenance—*version control*; D.3.2 [**Programming Languages**]: Language Classifications—*concurrent, distributed, and parallel languages*, object-oriented languages; H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*virtual reality*; I.3.8 [**Computer Graphics**]: Applications

General Terms: Design, Human Factors

Additional Key Words and Phrases: Electric power control system, information visualization, parallel manipulator

1. INTRODUCTION

Information has no shape or color. Thus, one of the advantages of using diagrams is that virtual shape can be given to information to aid our comprehension and understanding. For example, a logical hierarchy itself is not a tree. The tree shape is created for our cognitive convenience. Once such mental models are created, we begin to think with these figures. Therefore, it is essential to create and portray diagrams which are designed from a human cognitive perspective.

This work was partially supported by the Tokyo Electric Power Company.
Author's address: Department of Communications and Systems, University of Electro-Communications, 1-5-1, Chofugaoka, Chofu, Tokyo 182, Japan; email: koike@cas.uec.ac.jp.

The purpose of software visualization systems [Chang 1988; Meyers 1986; Shu 1988] is to support software development by portraying such diagrams on computer displays. The diagrams which were once represented on paper media are now visualized on computer displays due to the advances in bitmap technology. Consequently, it became possible to store, recall, and reuse diagrams, and to even interact with the information in them. Currently, 3D computer graphics is the primary focus in the development of new output media. Mechanical CAD systems, molecular-modeling systems, and scientific-visualization systems have emerged since the introduction of 3D computer graphics. The reasons are as follows: (1) they can visualize objects which do not really exist (such as Van-der Waals surfaces), (2) they can visualize inherently 3D objects (such as molecular models), and (3) they can visualize dynamics through use of their animation capabilities. At present, 3D graphics are also being used in software visualization. However, the use of 3D graphics in software visualization raises an important question: How do we visualize shapeless software information in 3D?

The main goal of this article is to provide insight into this question by focusing on a performance visualization tool for parallel/concurrent computer systems. We address issues from two different perspectives: expressiveness of output media and user cognition. The former perspective discusses the limitations of 2D output medium, and the latter discusses the user's cognitive load when 2D representations are used in a multiple-window environment. As we stated earlier, consideration of the user's cognitive viewpoint is necessary when any visual representations are designed.

In this article we show how these problems can be reduced by applying a 3D visualization framework. A 3D framework represents one relation (e.g., the relation between processes) in 2D and assigns another meaning (e.g., the time) to the third axis. Thus, it is possible for the user to see two relations (e.g., process relations and their execution pattern) simultaneously as well as to focus on each single relation without changing mental models. A prototype visualization system called VOGUE (Visualization-Oriented Generic User Interface Environment) has been developed. Using VOGUE, we show how the 3D framework can be used to visualize the execution patterns of parallel/concurrent computer systems. We also compare the effectiveness of this approach to traditional approaches and discuss visualization of other software applications using the 3D framework.

2. RELATED WORK

Information visualization in 3D space remains an area with much to be explored. Particularly, in the visualization of parallel computers, only a few systems have used a 3D framework. Related work falls into two main categories: information visualization in 3D space and performance visualization of parallel/concurrent systems.

Pioneering work in information visualization in 3D space was done by SemNet [Fairchild et al. 1988]. SemNet was built to aid the maintenance of large knowledge bases by visualizing Prolog knowledge bases as 3D graphs.

The authors also experimented with automatic layout and navigation techniques. SemNet shows that link crossing, as it occurs with 2D graphs, can be minimized and that it is easier to trace each link. Other early work includes Lieberman's [1989] system, which represents the tree structure of programs as boxes and visualizes program execution with animation. The focus is on the animation capability of computer graphics. However, the author does not discuss the effective use of an additional spatial dimension. Information Visualizer [Card et al. 1991; Mackinlay et al. 1991; Robertson et al. 1991], developed at Xerox PARC, clearly demonstrates the importance of 3D visualization. For example, Cone tree [Robertson et al. 1991] visualizes hierarchical structures in 3D trees and demonstrates the effective use of the screen. Perspective wall [Mackinlay et al. 1991] visualizes relational information as 3D walls and shows the smooth integration of detail and context. The authors also described how interactive animation reduces the user's cognitive load. FSN [Silicon Graphics 1992] is a file navigation tool for UNIX and uses a 3D graphics library to implement directory hierarchies. FSN focuses on a perspective view of 3D graphics. Although all display 3D graphics, rotation about any axis does not reveal any new information. One perspective allows all information to be seen.

Much work has been done on performance visualization of parallel/concurrent systems. An early system was the PIE [Lehr et al. 1989] system developed at Carnegie-Mellon University, which is a performance monitor for the Mach Operating System. PIE visualizes the execution of threads in 2D graphs, where the threads are placed on the y-axis, and the x-axis indicates time. The authors reported that it became easier to recognize the kernel configuration. A similar system is ParVis [Linden 1990] developed at MIT. ParVis is a visualization tool for parallel LISP. ParVis visualizes process execution using a 2D process-time framework. The relation between the parent process and the child process is specified by a line drawn from parent to child. JED [Maloney 1990] is a visualization tool for the Cedar multiprocessor environment. It focuses on events which are visualized as different icons. There are many other examples [McDowell and Helmbold 1989]; however, it is noteworthy that these visualization systems use a common framework—a 2D process-time space. The Pavane System [Cox and Roman 1991] proposes another visualization framework for concurrent computations and also tries to introduce 3D concepts. The role of the additional dimension, however, is not discussed analytically.

3. 2D VISUALIZATION PROBLEMS

As computer systems become larger and more distributed, parallel/concurrent programming will play a larger role. However, the debugging techniques currently used in sequential programming are not sufficient for parallel/concurrent programming. McDowell and Helmbold [1989] reported that the visualization of control flow, or that of distributed data, is one effective approach for debugging such programs. He also concluded that the debugger for complex parallel/concurrent systems should visualize both the states of

the system at one moment as well as the running pattern for a certain time in a multiple-window environment. As described in Section 2, most of the performance-tuning tools use the process-time framework. If communication between processes takes place, they are represented as lines or arrows from the sender process to the receiver process in these diagrams. For such 2D visualization, we will examine two issues: expressiveness of output media and user cognition.

3.1 Expressiveness of Output Media

Two-dimensional output media such as paper or bitmap displays can generally represent two independent relations. This fact, however, limits the expressiveness of diagrams visualized in 2D media. The following examples illustrate these limitations:

—*The overflow of the y-axis due to the increase in the number of processes.* In current 2D tools, 10 or 20 processes, at most, are expected. However, in a larger system such as an electric power control system (described later), hundreds of processes are executed and need to be visualized. Moreover, in the near future, hundreds or thousands of processes will be running in parallel computer systems. Unfortunately, current 2D frameworks cannot visualize such a larger number of processes on the y-axis.

—*The representation of the relation between processes.* Consider, for example, the transputer system [Inmos 1988]. A transputer itself is a multiprocess computer, and several processes can run simultaneously within it. Moreover, by connecting four physical channels to other transputers, a multiple CPU system can be constructed. Thus, several processors run in parallel, while several processes run concurrently in each processor (see Figure 1). These processes can be classified corresponding to the processors they belong to. It is not natural to display this hierarchical structure in one dimension. Moreover, the request to visualize the physical connections between transputers can not be satisfied. Consider, as another example, geographically distributed computer systems. In such systems, the layout or location of computers is crucial. Thus, to recognize their cooperative computation, it is necessary to visualize both their physical placement and time relation simultaneously. These requirements, however, are not satisfied in a 2D framework.

—*The representation of the communication between processes.* As we described earlier, interprocess communication is often represented by lines or arrows. However, these lines must intersect the parallel lines for each processor (see Figure 2). If the number of processes or amount of interprocess communication (such as the communication between the parent process and the child process) increases, the diagram becomes difficult to understand.

As the above examples indicate, current 2D frameworks for the performance visualization of parallel/concurrent systems are not likely to accommodate future systems.

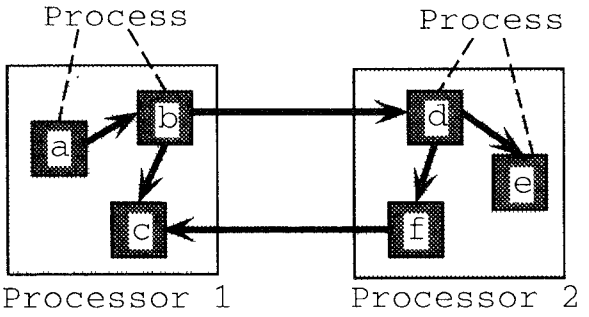

Fig 1. This diagram represents the relation between transputers and their processes. Several processes are running in each transputer. To represent this relation, two dimensions are needed.

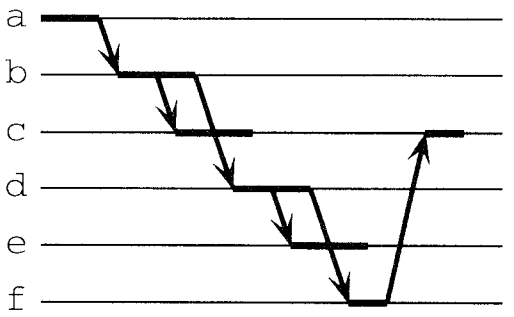

Fig 2. This diagram represents the execution pattern of each process and interactive communication. In this framework, it is difficult to represent hundreds of processes Moreover, the arrows which represent message passing between processes must intersect the horizontal lines for each process.

3.2 Cognitive Load of the User

As discussed earlier, designing from the user's cognitive perspective is essential in building effective visual representations. A system developed without such considerations may be beneficial for demonstrations, but never be very supportive in practical programming.

One of the important properties of the human visual system is its parallelism. We are able to rapidly process numerous pieces of information in an instant. Thus, in order to leverage these capabilities, it is desirable to display as many relations as possible in one diagram. However, this may cause an increase in complexity, and if the amount of information and complexity exceeds reasonable limits, the diagram becomes so complex that it does not help the users and may even cause further and unnecessary confusion.

To disperse the complexity, separated diagrams are generally used. Particularly, in multiple-window environments, diagrams showing different visual aspects can be shown in different windows. However, if a user's mental models are different, the same object is often represented in different shapes. For example, in visualizing transputer systems, the connection between transputers is represented as one diagram (see Figure 1), and its execution pattern is represented as another diagram (see Figure 2). In this case, the transputers placed in 2D space in Figure 1 and the transputers placed on the y-axis in Figure 2 are the same (i.e., *a* in Figure 1 corresponds to *a* in Figure 2; *b* in Figure 1 corresponds to *b* in Figure 2, etc.).

Such use of diagrams is sometimes quite confusing and counterproductive. The diagram may help users to construct their mental models, but in the same token, force them to do it in a way that may be unproductive. Hence, users obtain different mental models from each diagram. To thoroughly understand the relationship between processes and their running patterns, users must reconstruct one mental model which satisfies each constraint (see Figure 3).

3.3 3D Visualization as a Proposed Solution

The problems associated with the limitation of output media can be effectively reduced by using a 3D framework. Figure 4 represents the concept of a 3D performance visualization tool for parallel/concurrent systems. In this framework, two spatial dimensions (the xy-plane, for example) are used to represent the relations between processes, such as hierarchical structures or geographical locations. Another spatial dimension (the z-axis, for example) is used to represent time. If the current 2D tool can visualize n processes on the y-axis, approximately n^2 processes can be visualized in the xy-plane. In addition, this framework can represent the relations between processes because we have two dimensions to layout processes. Moreover, in general, arrows for interprocess communication do not intersect with lines for processes.

This framework can also considerably minimize the problems associated with user cognition. Using this framework, two 2D diagrams are integrated into one 3D diagram. Since only one representation is given to users, there is very little need for users to reconstruct their mental models. The 3D representation itself satisfies the constraints of each 2D diagram. In other words, this 3D representation itself can literally be a user's mental model. For this reason, it appears to reduce user cognitive load. Although experiments are necessary to provide statistical significance, the following example can provide support for our discussion. Consider, Apollonian curves as shown in Figure 5. Apollonian curves include circles, ellipses, and parabolas. When we see these curves, it may be difficult to understand the relations behind them. However, by introducing a conceptual model with a cone and a plane, which divides the cone, the relation between each curve is easily seen. The 3D model which conveys the information of 2D curves facilitates our understanding.

People may question whether or not current 2D diagrams are more desirable when focusing on the process-time relation. However, if we view the 3D representations from the direction perpendicular to the z-axis, we can obtain a process-time diagram. When we wish to focus on the relation between processes, we may rotate our viewpoint and look parallel to the z-axis. Ultimately, each 2D diagram is one of the viewpoints in the 3D representation, depending on which viewpoint we have chosen.

4. VISUALIZATION EXAMPLES

In this section, two applications of our 3D framework will be discussed. The first example is an electric power control system. The second example is a

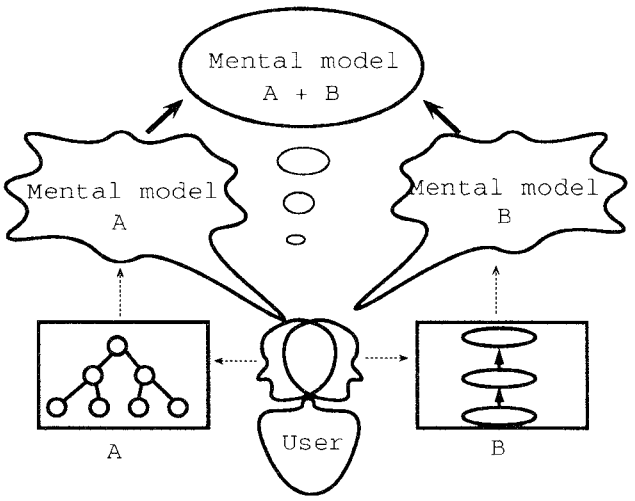

Fig. 3. Reconstruction of a mental model. The user obtains different mental models from each diagram. The user, then, has to reconstruct one mental model which satisfies each constraint.

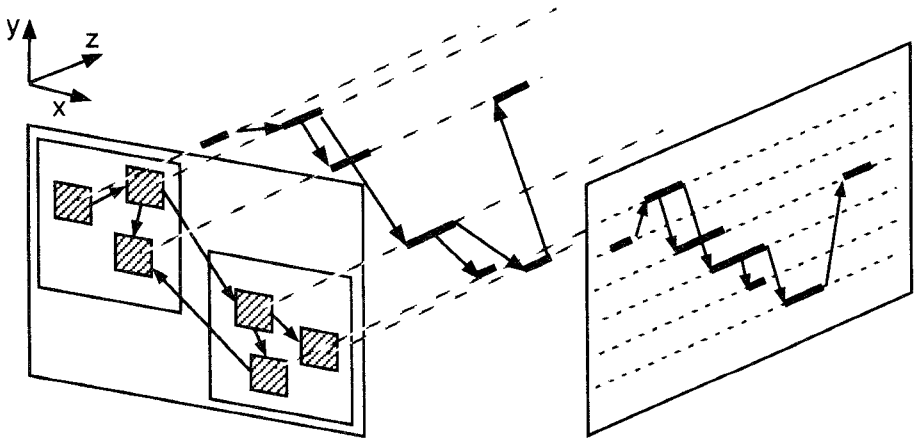

Fig 4 The concept of a 3D performance visualization tool for parallel/concurrent systems. Using three dimensions, the process relations (such as those shown in Figure 1) and their time flow (such as those shown in Figure 2) can be represented in one diagram.

parallel manipulator system for a robotic arm. Both visualizations were obtained using a prototype visualization system called VOGUE.

4.1 VOGUE Overview

There are two modules in VOGUE: an object-oriented database [Atkinson et al. 1990] module and a 3D grapher module. The object-oriented database was implemented by extending CLOS (Common Lisp Object System) [Steele

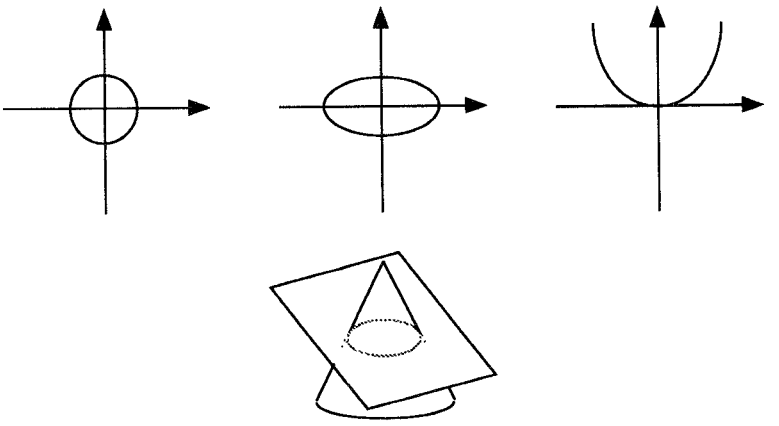

Fig. 5. Apollonian curves such as circles, ellipses, and parabolas. The relation between each curve is more easily understood when a 3D model with a cone and a plane is introduced.

1990], while the 3D grapher was implemented with 3D graphic libraries. (Initially, the 3D grapher was implemented on a Hewlett-Packard HP9000/350SRX [Hewlett-Packard 1988]. Presently, it is running on a Silicon Graphics IRIS workstation [Silicon Graphics 1991].) To obtain visual representations, users made models for their target software by defining classes as subclasses of the VOGUE node class or VOGUE link class, both of which are predefined in the object-oriented database. For example, processors or processes are defined as a subclass of the VOGUE node class, and their relation is defined as a subclass of the VOGUE link class. After each instance is created, it is mapped as a graphic node or a graphic link, respectively.

The system has some interaction capabilities. Sliders and buttons on the graphics display allow users to translate or rotate their viewpoints dynamically. Shepard and Metzler's [1991] experiments on mental rotation showed that the time needed for a user to recognize whether or not two separately represented pictures were of the same object was dependent on the angle of rotation between the two objects. Thus, fast and continuous movement of the viewpoint is necessary. Users can change their focused node by selecting with the mouse. If the selected node corresponds to a UNIX file, it is possible to open an editor window and edit the file. Further details of the VOGUE implementation are discussed elsewhere [Koike 1991].

4.2 Electric Power Control System

In the electric power control system for TEPCO (Tokyo Electric Power Company), a number of computers run in parallel and communicate with each other. Each computer is responsible for about a hundred tasks and these tasks are subgrouped according to their roles, such as man-machine tasks, analysis tasks, and so on. To debug the system, an exclusive process monitor has been developed and has been put into practical use. By executing the system, the process monitor produces trace files, such as that shown in Figure 6, containing some process information such as the time at which the

NO.	INF. TYPE	TIME	PID	PROCESS NAME	STATE	PRI	M	SYSTEM EVENT	PC	PFC
3351	EVENT	22:18:32.27	10081	MMOPEC	COM	24		WAKE	7FFEDF8A	
3352	EVENT	22:18:32.27	100A6	NTAACP	HIB	30		WAKE	7FFEDF8A	
3353	CTXED	22:18:32.27	10086	MMPBIN	CUR	25	K		80008956	0
3354	CTXST	22:18:32.27	100A6	NTAACP	CUR	30	K		7FFEDF8A	
3355	WAIT	22:18:32 27	100A6	NTAACP	HIB	30	K		7FFEDF8A	0
3356	CTXST	22:18:32.27	10086	MMPBIN	CUR	25	K		80008956	
3357	WAIT	22:18:32.28	10086	MMPBIN	HIB	25	U		7FFEDF8A	0
3358	CTXST	22:18:32.28	10081	MMOPEC	CUR	24	U		7FFEDF8A	
3359	EVENT	22:18:32 28	100A6	NTAACP	HIB	30		AST	7FFEDF8A	
3360	CTXED	22:18:32.28	10081	MMOPEC	CUR	24	U		7FFEDF8A	0

Fig 6. A trace file produced by the process monitor. Each column shows the information about one process Generally, one trace file contains thousands of lines.

process was executed and the state of the process. In general, the size of the trace file is much larger (e.g., thousands of lines). At present, programmers have to trace these files line by line.

The visualization of this trace data is an attractive idea. The difficulties, however, are how to visualize such large numbers of processes and how to represent the task groups. A 3D framework can be used to address these problems. Figure 7 is an example of a visualization. The nodes corresponding to each task are arranged on the xy-plane, which is parallel to a display screen, so that tasks belonging to the same group are placed physically near each other and are stretched along the z-axis, which is perpendicular to the display screen. In Figure 7, only the task nodes of one task, which is indicated by a yellow line, are shown. When a trace file is specified, the system starts to animate the execution of processes. The system reads a line from a trace file, creates a node for the process, and puts it at an appropriate position. Since there are 14 different states of processes, different colors are assigned to them. Some links are added, making it easier to recognize the time flow. The system is capable of displaying two or more diagrams and comparing the difference between them. In addition, if the user selects a node, an edit window opens and displays the corresponding line in the trace file.

This visualization application illustrates some of the advantages of 3D visualization. As we can see, in this diagram, about a hundred processes can be visualized. Furthermore, the tasks belonging to each task group are placed physically near each other. It is difficult for traditional 2D visualization methods to effectively represent this information. Moreover, the smooth integration of local and global information (also demonstrated by Perspective wall [Mackinlay et al. 1991]) was achieved in our 3D visualization system. When a trace file containing about two thousand lines is visualized in a 2D framework, the nodes corresponding to each state are small in size. Thus, when we view the global execution pattern of the graph, the local information such as process state, which is indicated by different colors, cannot be seen. To view the local information, part of the graph needs to be magnified. When this is done, however, the global information cannot be easily seen and recognized. In contrast, when observing with 3D visualization and using a 3D

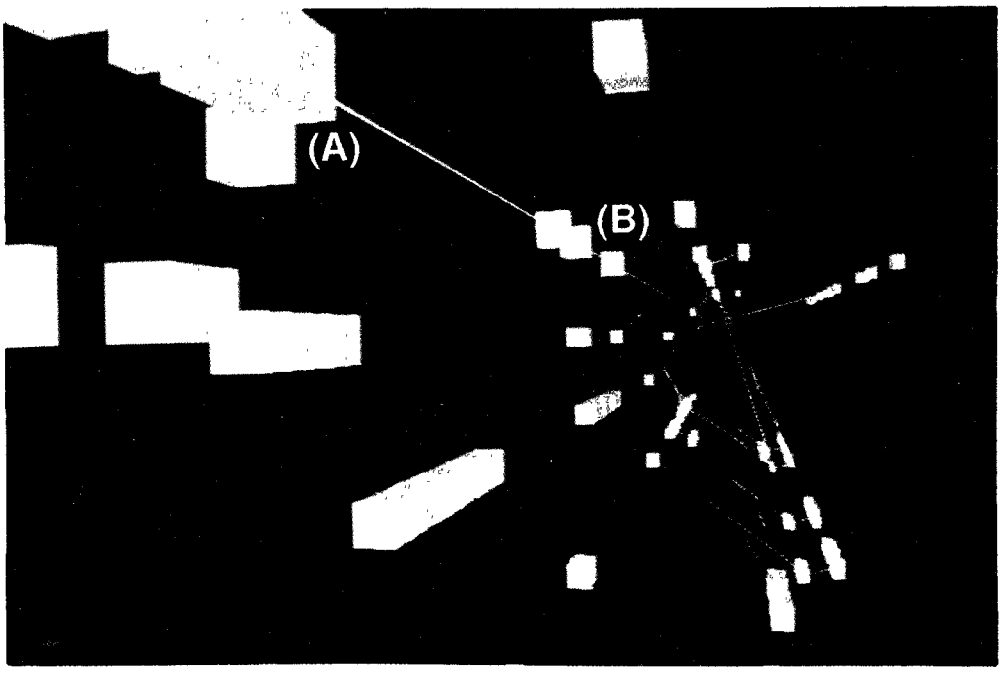

Fig. 7. A 3D visualization of the trace file produced by the process monitor when executing the electric power control system at Tokyo Electric Power Company.

perspective view, it is possible to observe the local information which is close to the viewer's eye while maintaining the rest of the execution pattern of graphs within our field of vision.

Due to the risks involved with experiments on a real electric power control system, some experiments were conducted using a TEPCO system simulator, which employed the same computers as the real system. Possible scenarios were made by patching existing programs. For example, a deadlock situation was created by altering the order in which processes obtained resources. The visualization of deadlock is similar to Figure 7, but the yellow node, which is indicated by (A) in the figure, is replaced by a green node, and following nodes, which are indicated by (B) in the figure, do not appear. This representation shows that the process is blocked and is waiting indefinitely for resources. Our experience shows it was more difficult for users, using 2D visualization, to identify the difference between the two diagrams. This is because it was more difficult, with 2D visualization, to note the change of color and the disappearance of nodes simultaneously.

4.3 Parallel Manipulator System

Ikei et al.'s [1990] parallel manipulator is a robot manipulator arm which contains a CPU for each joint. Each CPU runs in parallel and communicates

with other CPUs through message passing. When one joint is damaged, a given task is completed by using the other joints. In the experiment, 2D visualization was used to represent the process execution and the message passing. Figure 8 represents the message passing during a normal situation, and Figure 9 represents the message passing when the 3rd joint is damaged. However, the arrows which represent message passing must intersect the vertical lines for CPUs. Consequently, it is difficult to identify the differences between these diagrams.

This problem motivated the development of debuggers for parallel systems. When we debug parallel systems, it is crucial to identify where and when the trouble occurred. Using visual representations, this can be done by creating a visual pattern of message passing. The programmers may identify the failures by finding the communication or execution of processes which are not seen during the normal situation. It is, however, difficult in 2D due to the complexity of the diagram.

We applied our 3D framework to the parallel manipulator system. Figure 10 illustrates 3D visualization during a normal situation, and Figure 11 shows 3D visualization when the 3rd joint is damaged. A node for the network management processor is placed at the center of the display, and a node for the supervisory controller is placed to the left of it. The nodes for joint4, joint3, joint2, joint1, hand, and sensor are located clockwise from the top of the display to the bottom. When we observe Figure 11, we notice that communication exists between the 3rd joint and the 1st joint, which does not exist in a normal situation. Therefore, we can infer that some trouble might have occurred at the 3rd joint and that the 3rd joint might be sending messages to the 1st joint. Thus, the programmer can effectively identify hardware/software trouble. This is not only because the link-crossing problem (described in [Fairchild et al. 1988]) is minimized in 3D, but also because two aspects, time flow and the relationship between processors, are represented simultaneously in 3D while separating each relation.

5. DISCUSSION

5.1 Toward a Practical 3D Visualization Tool

The main objectives of this article are to discuss the role of an additional spatial dimension and to show the effectiveness of a 3D framework with practical examples. Therefore, it is not sufficient to develop a tool specific to a certain application, but necessary to develop a tool which can be applied to several applications. In VOGUE, the 3D grapher is equipped with basic operations, such as changing the position of nodes, scale, or color. Consequently, VOGUE is generic enough to be applied to different parallel/concurrent systems as well as different types of visualizations which will be described in the next section. However, to make 3D visualization practical, it is also necessary to discuss the use of such graphic attributes of nodes.

In addition, significant problems remain unsolved, these include: interaction with 3D objects and increase in the number of graphic elements.

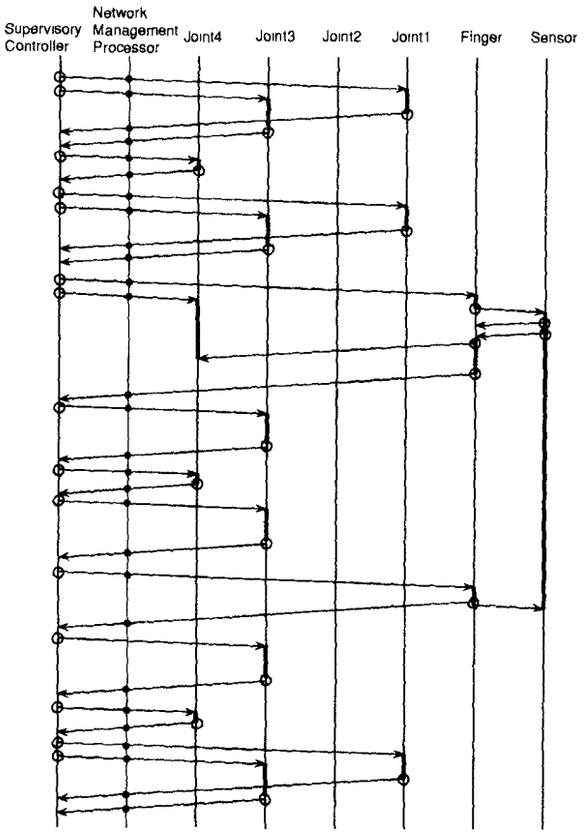

Fig. 8. Two-dimensional visualization of the message passing during a normal situation.

—*Interaction with 3D Objects.* There are two primary aspects associated with the interaction between 3D objects. One is the manipulation of 3D objects (i.e., holding/releasing, translating, and rotating the objects). Another is the changing of the user's viewpoint in 3D space. With 2D tools, the horizontal/vertical translations and zoom in/out are sufficient for changing user viewpoints. On the other hand, with 3D tools, in addition to the translations, both rotation around the user and rotation around the focal point must also be considered. Some research [Chen et al. 1988; Fairchild et al. 1988; Robertson et al. 1991] attempts to address these problems using 2D input devices, such as a mouse. In addition, virtual-reality techniques which use new input/output devices such as the DataGlove [Zimmerman et al. 1987], head-mounted display, and so on, are also being developed. For example, work by Feiner and Beshers [1990], Fisher et al. [1986], and Show et al. [1992] can be easily integrated with our 3D visualization techniques. We are also experimenting with the interaction techniques in virtual environments whereby the visualization in this study

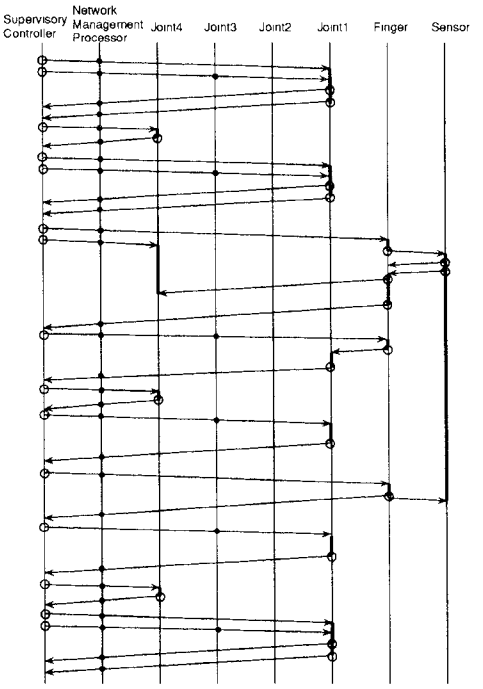

Fig. 9. Two-dimensional visualization of the message passing when the 3rd joint is damaged.

is being displayed on a video projector, and the nodes and links are manipulated directly with a DataGlove [Myoi et al. 1991]. These so-called VR techniques should be experimented with in parallel with visualization techniques as they are closely related.

—*Increase of Graphic Elements.* The update/refresh rate of graphic displays, in general, deteriorates with an increase in the number of graphic elements. When we visualize a large amount of data, it is difficult to obtain real-time interaction. Furthermore, the increase in the number of graphic elements disturbs user cognition. Thus, it is necessary to reduce the amount of displayed information. With VOGUE, we are experimenting with several methods, such as an object-oriented approach and a syntactic approach. The former uses node information which is held by an object-oriented database in VOGUE. For example, it can isolate and display the instances of a class and its subclasses. The latter focuses on a logical structure of the graph and displays a focus node and its neighbors. SemNet uses fisheye views [Furnas 1986] to reduce graphic elements. In VOGUE, fisheye views as well as a fractal-based technique have been implemented.[1] However, none of these methods is a total solution in itself. Further

[1]By using the technique described in Koike and Ishii [1992], users obtain a view similar to Furnas's fisheye views. This technique, however, can keep the total amount of displayed information nearly constant whichever node the user focuses on.

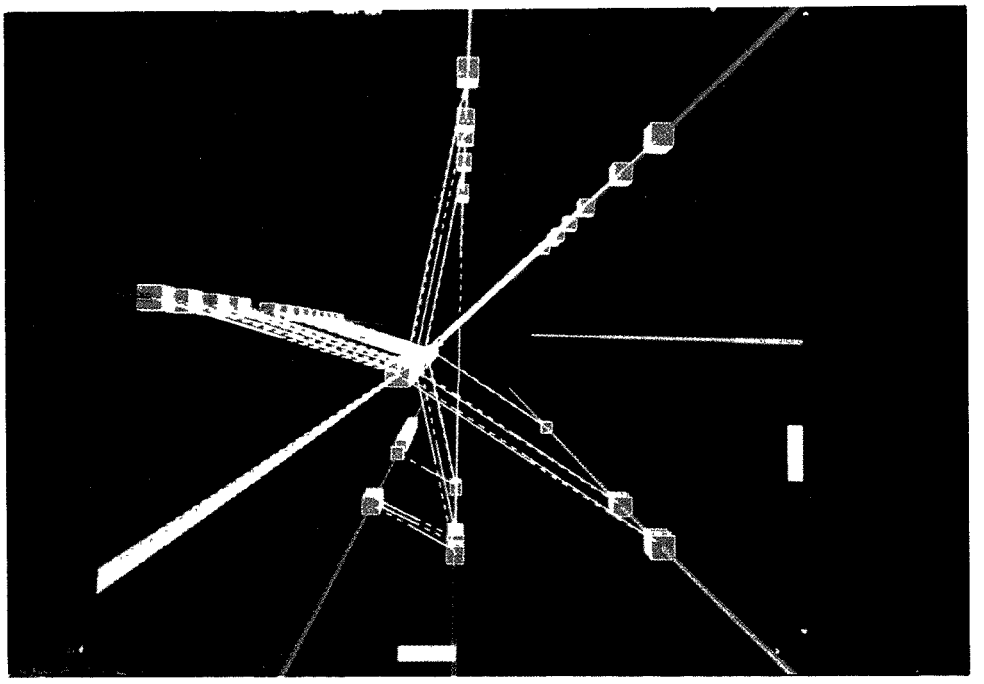

Fig. 10. Three-dimensional visualization of the message passing during a normal situation.

research will be critical to finding more complete solutions to this problem through the integration and fusion of various technologies and methods.

5.2 Other Applications

In the previous sections, we showed how the 3D framework was applied to improve the performance of visualization tools for parallel/concurrent systems. Another spatial dimension made it possible to visualize both the relations between processes and their time flow, while not disturbing user cognition. In software development, there are many other situations where it is natural (or necessary) to display two relations simultaneously. Two additional application areas will be described for 3D visualization.

5.2.1 *Class Libraries of Object-Oriented Languages.* Method inheritance in object-oriented languages is an efficient way of reusing program source code. The message which is sent to one instance of a certain class is not always processed by a method of the class. In this case, the method really executed belongs to one of its superclasses. Attention to this complex mechanism of method inheritance is not necessary when we run the program because it is done by the system automatically. However, when we debug the program, we must trace code line by line.

To support object-oriented programming visually, class hierarchy trees (see Figure 12) are generally used. However, the position of the method actually

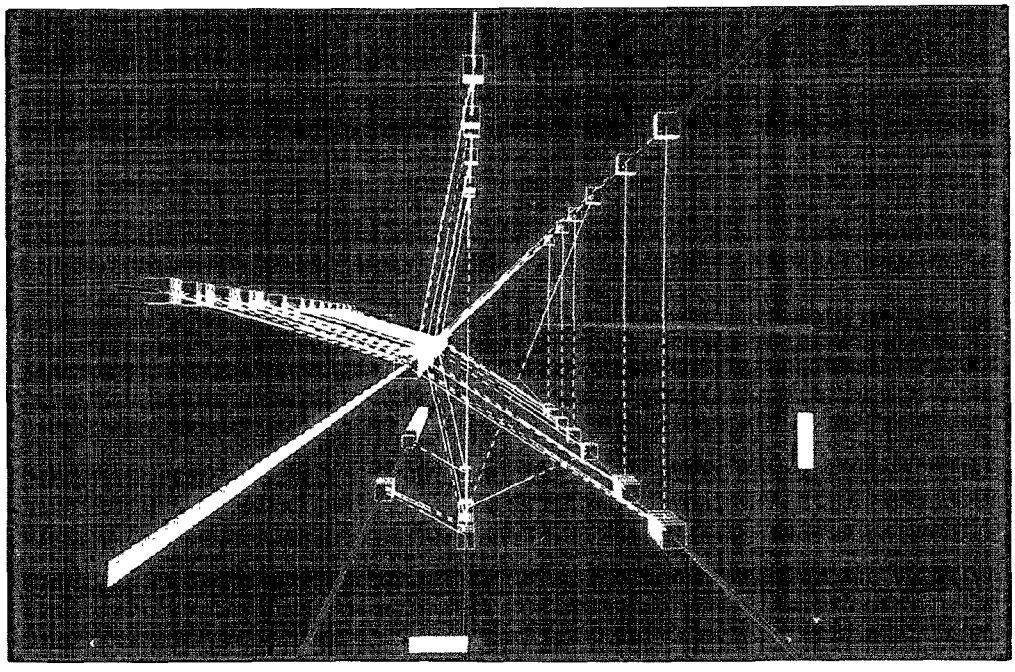

Fig. 11. Three-dimensional visualization of the message passing when the 3rd joint is damaged

executed is not clear just by observing the hierarchy tree. We need to know the list of classes (see Figure 13) that have the method of the same name. On the other hand, if we have only the method list, we cannot tell which method was really executed. Thus, we require both pieces of information. However, if these pieces of information are given by different visual representations, we must make efforts to construct mental models that satisfy each constraint.

Figure 14 shows a 3D representation of a class library. The class hierarchy is represented as a tree in the xy-plane. Each method node has the same xy-coordinates as the class node to which the method belongs, and the methods that have the same name have the same z-coordinate. If we look in the xy-plane, we realize the class hierarchy. If we look from the direction perpendicular to the z-axis, we realize the method list.

This visualization was also implemented in VOGUE (see Figure 15). In the figure, red nodes and white nodes represent classes and methods respectively. Users can rotate their viewpoints using sliders. If a node is selected, an editor window will open, and the source code is displayed.

Using this type of visualization, some experiments were conducted [Koike 1992]. For example, subjects were asked to answer the return value of several CLOS expressions. Results concluded that performance was much better with this type of visualization compared to a Prograph-style 2D visual browser [Cox and Pietrzykowski 1988] or a Smalltalk-style textual browser [Goldberg and Robson 1983]. Since the methods with the same name were positioned on

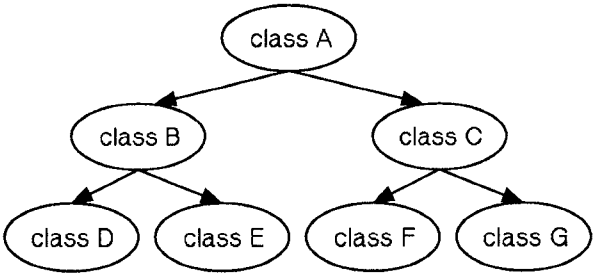

Fig. 12. The class hierarchy tree.

	Method 1	Method 2
class A	O	O
class B	O	
class C		O
class D		O
class E	O	
class F	O	
class G		O

Fig. 13. The list of methods which have the same name.

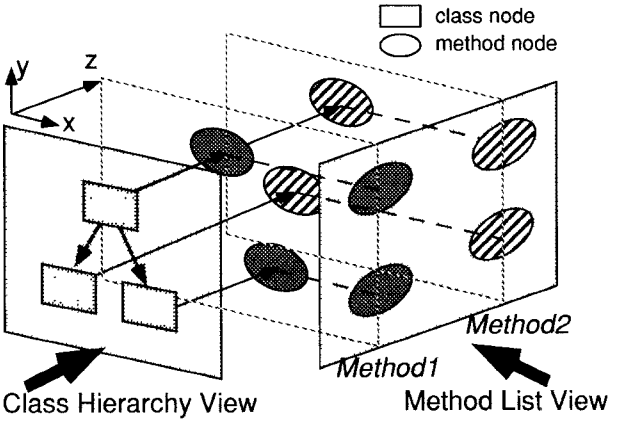

Fig. 14 The concept of 3D visualization of a class library. The class hierarchy and the method list are visualized in one diagram.

the same vertical line in a common xy-plane when viewing from the direction perpendicular to the z-axis, while the class hierarchies were also visible, users could easily identify the method which was really executed.

In practical object-oriented programming, class libraries are often modified by programmers. Methods can be deleted from subclasses and defined in a superclass. Such generalization processes must be done carefully because the change at the superclass effects its subclasses. Since programmers could

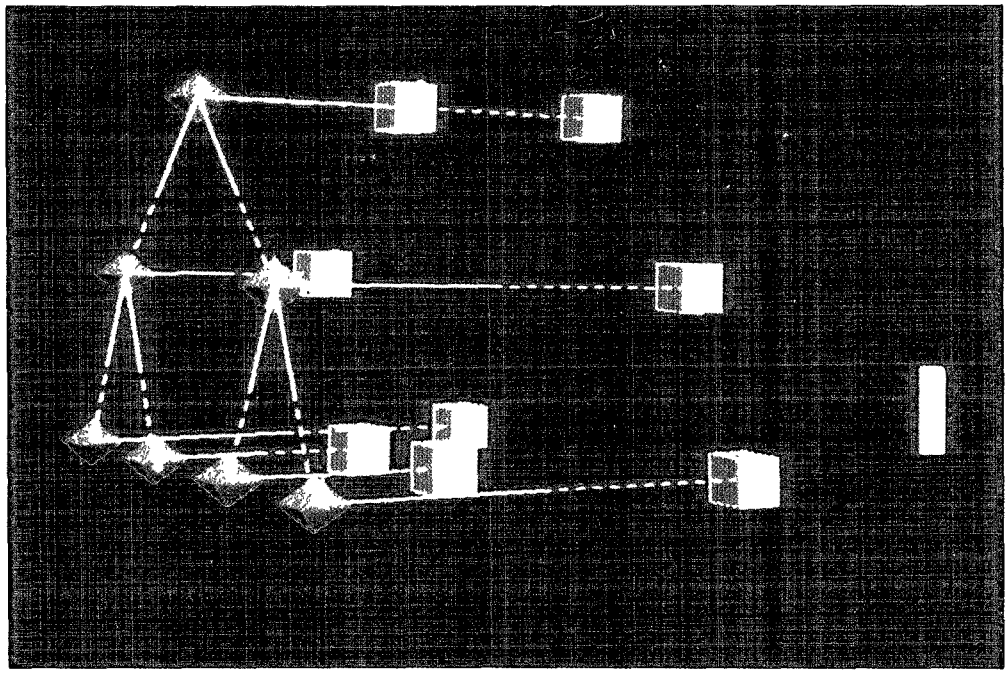

Fig. 15. Three-dimensional visualization of a class library implemented in VOGUE

easily recognize, with our 3D visualization framework, which classes were affected when they rewrote the method, unexpected errors were reduced. Also, 3D visualization was used to teach the concept of method inheritance in object-oriented programming to undergraduate students. From our experimental results, students learning with 3D visualization understood the concepts more quickly and easily.

5.2.2 *Version and Module Information.* In practical software development processes, version control and module management are very important. In UNIX, SCCS (Source Code Control System) or RCS (Revision Control System) is used as a version control tool, and Make is used as a module management tool. Unfortunately these tools do not have visual interfaces. On the other hand, CASE tools have more powerful capabilities as well as visual interfaces. In CASE systems, the version information is represented as trees along a time axis (e.g., Figure 16), and the module information is represented as trees (e.g., Figure 17) or bubble charts in 2D.

However, the version information and the module information are inherently unseparable. Consider, for example, a certain software system which consists of module A, B, and C (see Figure 17). In this example, when the system version is 1.0, the version number of each module is also 1.0. On the other hand, when the system version is 2.0, each version number is not the same. Module A is updated many times, while module C has no need to be

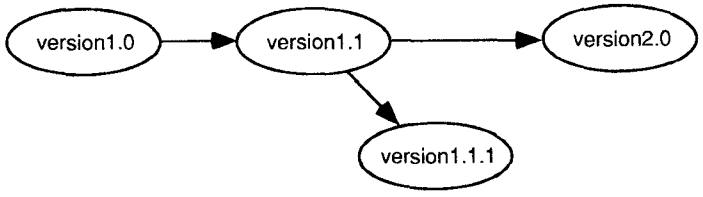

Fig. 16. A version tree.

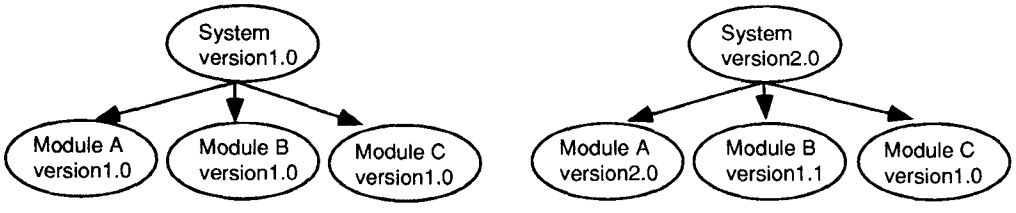

Fig. 17. A module tree. In general, each version number is not the same. In this example, Module C is not updated and is used both in System 1.0 and in System 2.0.

updated and is reused. The version history of module A is represented as shown in Figure 16. To install the software, it is necessary to know which modules it consists of as well as the version of each module.

It is possible to use a 3D framework to visualize both version and module information simultaneously, as shown in Figure 18. Using this representation, we can visually recognize the version/module structures. Although, it is possible to develop a system which retrieves, compiles, and links the needed versions, software engineers should still be aware of (i.e., have mental images of) this information.

6. CONCLUSIONS

This article describes how 3D space could be effectively applied as a framework for visualizing parallel/concurrent computer systems. The importance of 3D software visualization is emphasized from two different perspectives: expressiveness of output media and user cognition. To summarize, using another spatial dimension (the 3rd dimension which is perpendicular to the 2D plane) makes it possible to represent separately two (or more) relations in one 3D diagram in a way that makes it easy for users to quickly create their own mental models. Since only one representation is given to users, they have less difficulty in constructing mental models. As we discussed before, visualization systems must be designed from a user's cognitive perspective. In particular, efforts must be made to reduce the user's cognitive load, thereby improving performance.

We also discuss other application areas of 3D visualization. Both object-oriented programming and version/module management are very important

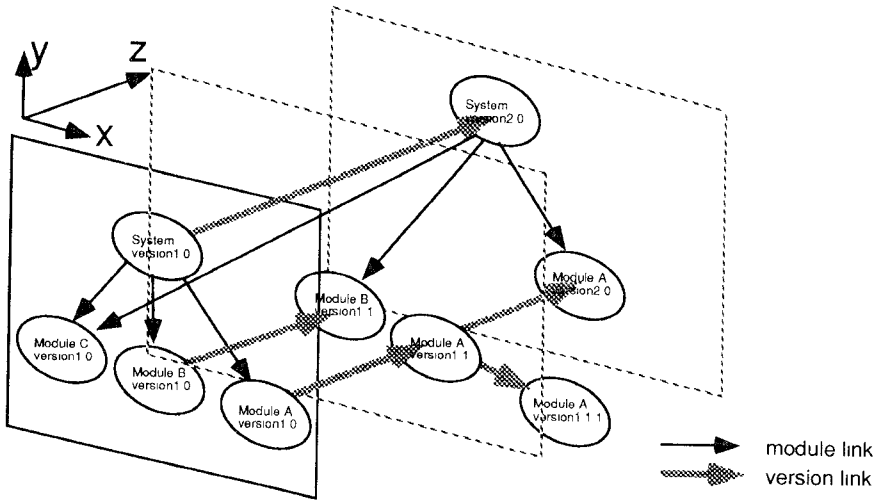

Fig 18 Concept of 3D visualization of both version and module information. The version tree and the module tree are represented in one diagram

in software engineering. Another spatial dimension plays a crucial role in these types of visualizations. Our long-term goal is to develop a visualization environment which integrates such visualizations.

ACKNOWLEDGMENTS

The author would like to express his deepest gratitude toward Takemochi Ishii of Keio University for his many helpful suggestions and toward Yasushi Ikei of Osaka University for providing trace data from his manipulator research. As well, the author would like to thank Simon Gibbs and the other reviewers for their valuable comments.

REFERENCES

ATKINSON, M., BANCILHON, F., DEWITT, D., DITTRICH, K. R., MAIER, D., AND ZDONIK, S. 1990. The object-oriented database system manifesto. In *Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases*

CARD, S. K., ROBERTSON, G. G., AND MACKINLAY, J. D. 1991. The information visualizer, an information workspace. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*. ACM Press, New York, 181–188.

CHANG, S. K. 1988. Visual languages. A tutorial and survey, *IEEE Softw. 4*, 1

CHEN, M., MOUNTFORD, S. J., AND SELLEN, A. 1988. A study in interactive 3-D rotation using 2-D control devices. *Comput. Graph. 22*, 4, 121–129

COX, K. C., AND ROMAN, G.-C. 1991. Visualizing concurrent computations. In *Proceedings of 1991 IEEE Workshop on Visual Languages*. IEEE Computer Society Press, Los Alamitos, Calif., 18–24.

COX, P. T., AND PIETRZYKOWSKI, T. 1988. Using a pictorial representation to combine dataflow and object-orientation in a language independent programming mechanism. In *Proceedings of the International Computer Science Conference*. IEEE, New York.

FAIRCHILD, K. M., POLTROCK, S. E., AND FURNAS, G. W. 1988. SemNet: Three-dimensional graphic representation of large knowledge bases. In *Cognitive Science And Its Applications For Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, N.J.

FEINER, S., AND BESHERS, C. 1990. Worlds within worlds metaphors for exploring n-dimensional virtual worlds, In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology* (UIST'90). ACM Press, New York, 76–83.

FISHER, S. S., MCGREEVY, M., HUMPHRIES, J., AND ROBINETT, W. 1986. Virtual environment display system. In *Proceedings of the ACM 1986 Workshop on Interactive 3D Graphics*. ACM, New York

FURNAS, G. W. 1986. Generalized fisheye views. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'86)*. ACM Press, New York, 16–23.

GOLDBERG, A., AND ROBSON, D. 1983. *Smalltalk-80: The Language and Its Implementation*. Xerox.

HEWLETT-PACKARD COMPANY. 1988. *Starbase Graphics Techniques HP-UX Concepts and Tutorials*. Vol. 1.

IKEI, Y., HIROSE, M., AND ISHII, T. 1990. Fault tolerant design of holonic manipulator. In *Proceedings of MSET21: The International Conference on Manufacturing Systems and Environment*. JSME, 117–122.

INMOS. 1988. *IMS B015 User Manual*.

KOIKE, H. 1992. An application of three-dimensional visualization to object-oriented programming. In *Advanced Visual Interfaces*. Proceedings of the International Workshop AVI'92. World Scientific, 180–192.

KOIKE, H. 1991. An application of three-dimensional visualization to software engineering. Ph.D. thesis, Univ. of Tokyo. In Japanese.

KOIKE, H., AND ISHII, T. 1992. A fractal-based method for information display control. *Trans. Inf. Process. Soc Japan 33*, 2, 101–109.

LEHR, T., SEGALL, Z., VRSALOVIC, D. F., CAPLAN, E., CHUNG, A. L., AND FINEMAN, C. E. 1989. Visualizing performance debugging. *IEEE Comput. 22*, 10 (Oct.).

LIEBERMAN, H. 1989. A three-dimensional representation for program execution. In *Proceedings of the 1989 IEEE Workshop on Visual Languages* IEEE Computer Science Press, Los Alamitos, Calif.

LINDEN, L. B. 1990. Parallel program visualization using ParVis. In *Performance Instrumentation and Visualization*, ACM Press, New York, 157–188.

MACKINLAY, J. D., ROBERTSON, G. G., AND CARD, S. K. 1991. The perspective wall: Detail and context smoothly integrated. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*. ACM Press, New York, 173–179.

MALONY, A. D. 1990. JED: Just an Event Display. In *Performance Instrumentation and Visualization*. ACM Press, New York, 99–116.

MCDOWELL, C. E., AND HELMBOLD, D. P. 1989. Debugging concurrent programs. *ACM Comput. Surv. 21*, 4.

MYERS, B. A. 1986. Visual programming. Programming by example and program visualization: A taxonomy. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'86)*. ACM Press, New York, 59–66.

MYOI, T., AMARI, H., INAMURA, K., KOIKE, H., KUZUOKA, H., HIROSE, M., ISHII, T., AND HAYASHI, T. 1991. A method of large-scale control system design aided by system visualization technology. *Trans. IEE Japan 111-C*, 5, 194–201. In Japanese.

ROBERTSON, G. G., MACKINLAY, J. D. AND CARD, S. K. 1991. Cone Trees: Animated 3D visualizations of hierarchical information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*. ACM Press, New York, 189–194.

SHEPARD, R. N., AND METZLER, J. 1971. Mental rotation of three-dimensional objects. *Science 171*.

SHOW, C , LIANG, J., GREEN, M., AND SUN, Y 1992. The decoupled simulation model for virtual reality systems. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'92)*. ACM Press, New York, 321–328.

SHU, N. C. 1988. *Visual Programming* Van Nostrand Reinhold, New York.

SILICON GRAPHICS, INC. 1992. *FSN: File System Navigator*. Online Manual Silicon Graphics, Mountain View, Calif.

SILICON GRAPHICS. INC. 1991. *Graphics Library Programming Guide*. Silicon Graphics, Mountain View, Calif

STEELE, G. L., JR. 1990. *Common Lisp the Language* 2nd ed. Digital Press, Cambridge, Mass.

ZIMMERMAN, T., LANIER, J., BLANCHARD, C., BRYSON, S., AND HARVILL, Y. 1987. A hand gesture interface device. In *Proceedings of the ACM Conference on Human Factors in Computing Systems and Graphics Interface (CHI + GI 1987)*. ACM Press, New York, 189–192.

Received November 1992; revised April 1993; accepted April 1993