ICAT 2002
December 4-6, Tokyo, JAPAN

An Immersive Programming System: Ougi

Noritaka OSAWA[†*], Kikuo ASAI[†*], Motofumi SUZUKI[†], Yuji Y. SUGIMOTO[†] and Fumihiko SAITO[‡]

[†]National Institute of Multimedia Education, JAPAN
[*]The Graduate University for Advanced Studies, JAPAN
[‡]Solidray Co. Ltd, JAPAN
{osawa,asai,motofumi,yuji}@nime.ac.jp saito@solidray.co.jp

Abstract

The *Ougi* immersive programming system employs abstract data visualization and manipulations utilizing virtual reality technologies. Its multimodal interfaces allow one to edit a program, to control its execution and to debug the program by direct manipulation and hand gestures in an immersive virtual environment. *Ougi* uses both textual and graphical representations and supports a subset of the Java programming language as the target language. Its graphical jigsaw-puzzle-like representation shows grammatical constraints visually. *Ougi* enables us to edit nested hierarchical structures and to perform layout of graph structures using direct manipulation. *Ougi* lets beginners focus on learning the essence of programming without requiring familiarity with the keyboard and commands of a conventional text editor.

Key words: immersive programming system, jigsaw-puzzle-like representation, nested-board representation, Java, Java3D

1. Introduction

It usually takes beginners a long time to make and debug a program. This is partly because they need to understand the new concepts, syntax, and semantics of a programming language. Moreover beginners often spend a lot of time editing a program and correcting typos and misspellings, according to our observation, when a text programming language is used because many novice programmers are not used to touch typing (especially those in non-western countries) or the operations of a text editor or conventional programming environment system such as Emacs or integrated development environments (IDE). We think that this situation adversely affects their programming efficiency as well as their understanding of programming concepts. Multimodal interactions in immersive virtual environments can ameliorate this situation since the interactions can be better modeled after reality than a keyboard interface or GUI.

We think that three-dimensional space should be used in order to fully utilize the body's freedom of movement. To view objects in 3D, a 2D display is insufficient because it cannot give one an adequate sense of depth in 2D. Therefore a stereoscopic view is necessary to help one to touch, grasp, and move virtual objects in an immersive virtual environment.

We have developed a prototype system for immersive programming called *Ougi*[*], whose work-in-progress status was partly reported in [12]. *Ougi* allows one to edit and manipulate programs through direct manipulation and hand gestures in an immersive environment. It naturally employs 3D visualization. The current implementation supports a subset of the Java programming language [4] as the target language.

The main goals of the *Ougi* system are full utilization of the range of motion of one's body and the enhancement of a person's understanding of programming. Although the target users are both beginners and expert programmers, the current system is focused on beginners. Therefore, this paper explains the features and functions designed with beginners in mind. The utilization of body motion has a possibility to enhance the study of programming of beginners who cannot learn the essence of programming from a conventional programming environments or only textbooks. This paper also reports comments from students who used the system for a short period.

This paper is organized as follows. Related work is discussed in Section 2. Section 3 briefly explains *Ougi* system. Section 4 and Section 5 describe visualization and interactions in *Ougi*. After that, we show screenshots in several views of *Ougi*, and explain functions and features of *Ougi* in Section 6. Section 7 explains the implementation of the prototype system and Section 8 discusses the prototype system. A summary is given in Section 9.

[*] <http://www.nime.ac.jp/~osawa/research/ougi/>
Ougi is a Japanese word that has two different meanings. One is "folding fan", which is a symbol of expansion and prosperity in Japan. The other is "secret". It is our hope that this system can help people to learn the secrets of programming.

2. Related Work

There has been considerable research on visual 3D programming and algorithm animation [2][5][7][14][15], or visual object-oriented languages [3]. However, an object-oriented programming system for immersive environments has not been implemented. *Ougi* supports object-oriented programming through one's body motion in an immersive environment.

Moreover, interactive multimodal interfaces for programming (or abstract and discrete information structures) in immersive environments have not been fully studied. Direct manipulation such as a grab-move-release (or get-and-put) operation is basic in 3D, but simply replacing the drag-and-drop operation in a GUI with a grab-move-release operation in 3D is insufficient. The design of 3D interactive interfaces is not a trivial problem because the degrees of freedom in a 3D space larger than on a 2D plane.

3. Ougi

As stated above, *Ougi* is an immersive programming system. It utilizes 3D visualization and multimodal interfaces. Specifically, a user wears 3D glasses that give him or her a stereoscopic view of the virtual environment and a sensor glove with a position tracker for detecting motions of his hand and fingers in the virtual environment. The user can 'write' a program by moving his body or hands. Figure 1 shows a snapshot of a user programming in the system. *Ougi* is also called "a system for a dancing programmer" because as the programmer uses the system, the programmer appears to be dancing in the virtual environment.

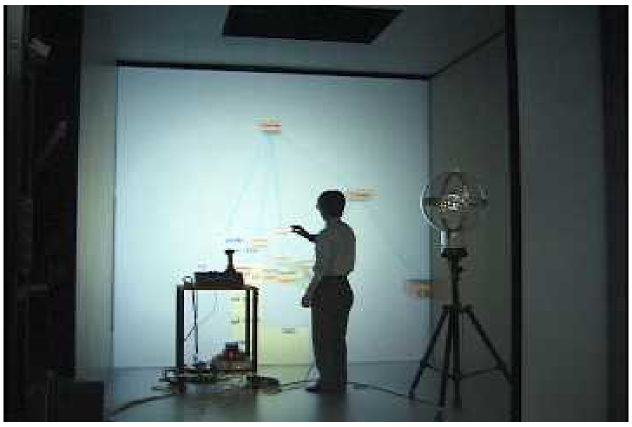


Figure 1: A snapshot of *Ougi*

4. Visualization

In general, the layout of a program in *Ougi* is automatically computed on the basis of layout rules such as force-directed layout and flow layout. In addition to automatic layout, we can move elements of a program manually.

4.1. Nested Structures

It is reasonable to represent language elements by nested structures according to the programming language's syntax. That representation is based on nested boards. Although translucent nested-box representations such as [7] were proposed and are interesting, those representations are not suitable for direct manipulation because it is difficult to select an inner box. When a user wants to choose an inner box, the user's hand intersects with outer boxes and thus the selection is ambiguous. In [7], direct manipulation of structures is neither discussed in an immersive environment nor in a desktop environment.

Therefore we used nested-board representation shown as in Figure 2. Nested-board representation is better than nested-box representation for direct manipulation in an immersive environment since the inner structure in the nested-board representation can be accessed without intersecting with the outer structures. Unfortunately, even if we use the nested-board representation, it is sometimes difficult to manipulate directly an inner element using a hand because the hand often touches or collides with unintended elements other than the intended inner element. A simple way to avoid this collision is to increase both the size of boards and the space between elements. This can solve the problem but the number of elements in a fixed space is limited and thus this is not desirable.

We therefore developed handle representation for direct manipulation of nested structures as shown in Figure 10. Each region has a handle shaped like a cylinder. Direct manipulation of the handle is used to move and copy the region. The nested structures and handles are not always shown in the *Ougi* system since they may limit the visibility within a structure or otherwise become visual clutter. When the virtual hand intersects with the regions of program elements, the handles of the regions appear.

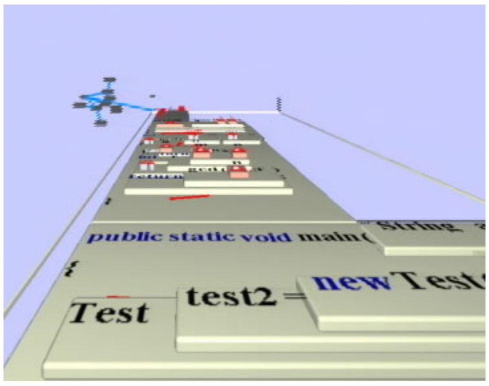


Figure 2: Nested boards

4.2. Jigsaw-puzzle-like Visualization

Although conventional visual programming uses visual objects to represent values, variables, or elements of a language, we do not think that the conventional visualization effectively visualizes information about classes, types, or signatures of methods. *Ougi* represents them by using 3D jigsaw-puzzle-like representation or 3D glyphs, which are based on 2D glyphs [9][11], in order to show grammatical constraints visually. In this paper, a glyph is a shape representing a relationship.

In the previous work[11], we demonstrated that all classes of the Java 2 Standard Edition (J2SE) could be given unique glyphs that satisfy the class inheritance relationships. Since J2SE has more than 1500 classes, jigsaw-puzzle-like visualization must be practical.

We also conducted experiments to determine the user's accuracy and speed to determine the inheritance relationship between two classes[11]. The experiments show that one can recognize inheritance relationships faster when the proposed glyphs are used rather than when an ordinary textual representation is used.

In jigsaw-puzzle-like visualization, a constraint is represented by an inclusion of shapes. For example, if a convex shape is included in a concave shape, they satisfy a constraint. "Jigsaw-puzzle-like" does not mean an exact match of shapes like an actual jigsaw puzzle.

As an example of jigsaw-puzzle-like visualization, let us explain representations of a type constraint in an assignment in the following. Unconstrained assignments are shown in Figure 3. An assignment has placeholders which are empty. The placeholders can include any type variable or value.

If a variable is put in the left-hand side of an assignment, the shape of the placeholder at the right-hand side is changed to the shape which can include a type of the left-hand side. A variable has a concave shape on the front side of its shape and a convex shape on the backside of it. The front side can hold a value of a type which is a subtype of the variable. The backside can be put in the concave placeholder which has a super-type of the variable. Figure 4 shows an example of constraint representation when the left-hand side of an assignment is specified.

If a value is set in the right-hand side of the assignment and the left-hand side is not specified, the wire frame shape of the right-hand side is displayed in the left-hand side. A variable at the left-hand side should not hide the wire frame shape in order to satisfy the type constraints in the assignment.

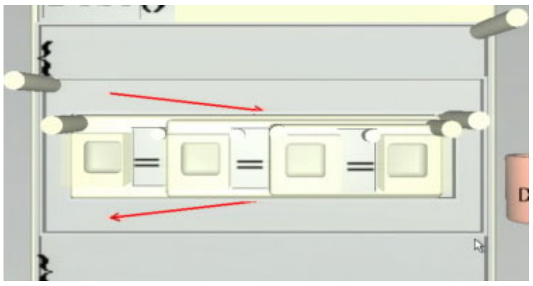


Figure 3: Unconstraint Assignments

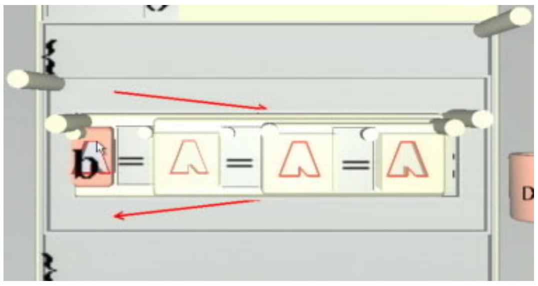


Figure 4: An example of constraint representation when the left-hand side of an assignment is specified

4.3. Textual and Graphical Visualization

Although *Ougi* is an immersive and multimodal programming system, it does not eliminate textual representations but utilizes both textual and graphical representations. The textual representations are retained as useful because we have all learned written languages for many years and understood the meanings of many words.

Hence we are not sufficiently trained in understanding the meanings of graphical symbols such as icons. It is difficult to understand abstract concepts only from graphical symbols because we do not have standard symbols for abstract concepts.

Furthermore, since most people are familiar with mathematical formulae, the usual mathematical expressions are also used in the system.

5. Interactions

We need to both manipulate program elements and navigate in an immersive program space in order to make a program and to debug it. To do that, we use direct manipulation and hand gestures, which are useful for such manipulation and navigation.

5.1. Direct Manipulation

In immersive virtual 3D space, direct manipulation by hand is intuitive because direct manipulation in a 3D environment is similar to that in real life. We do not need a mouse or a keyboard to pick up and move a virtual object, only our hands. In manipulating program elements in a 3-D space, the following operations are possible. Snapshots of direct manipulation are shown in Figure 5.

- Touching an element: A touch of a forefinger on an element selects it. When an element is selected, an action specific to the touched element is performed.
- Copying an element: One can copy an element by pinching it between his thumb and forefinger and moving it.
- Moving an element: One can arrange an element by pinching it between his thumb and middle finger and moving it.

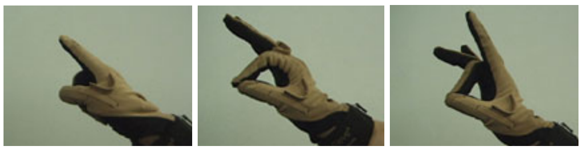

(a) touch (b) copy (c) move

Figure 5: Examples of direct manipulation

5.2. Hand Gestures

Although direct manipulation is intuitive in immersive virtual space, it is impossible to control invisible functions using direct manipulation. Therefore, in addition to direct manipulation, we need something magical to control programs in virtual space. To do so, the current prototype system uses hand gestures and virtual 3D widgets such as buttons. Hand gestures and operations on virtual 3D widgets can control a program. Figure 6 shows snapshots of hand gestures.

- Moving in a virtual space: One can move the virtual space by clenching one's fist and moving it. Rotation is not performed by this gesture.
- Resetting the position: One can move to the starting position by touching one's thumb with one's little finger.
- Menu panel control: One can switch a menu panel display on and off by bending both one's middle and ring fingers.
- Undoing operations: One can undo operations by bending both one's ring and little fingers. Multiple undo operations are possible. The number of undo operations is limited only by the amount of memory available.
- Redoing operations: One can redo undone operations by bending both one's forefinger and middle finger.

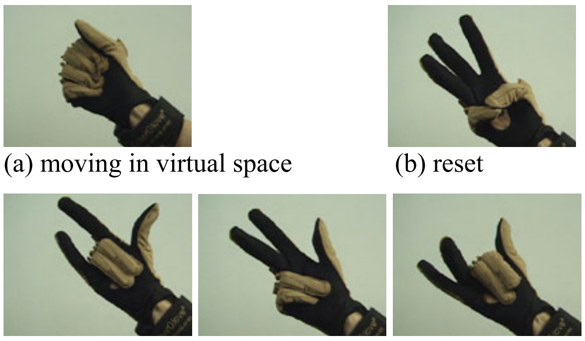

(a) moving in virtual space (b) reset
(c) menu control (d) undo (e) redo

Figure 6: Examples of hand gestures

6. Functions and Features

This section explains the functions and features of *Ougi* system and shows some screenshots.

6.1. Class Hierarchy

A class hierarchy is represented by a 3D graph. A class can be moved by direct manipulation using the hand. One can pick up a class with one's thumb and index finger, and then move the class. One can place the classes where one wants. Figure 7 shows a screenshot of a manual layout of a class.

The layout can also be arranged automatically by a force-directed technique[5]. Pushing the layout button on initiates the automatic layout. Figure 8 is a screenshot when an automatic layout is performed.

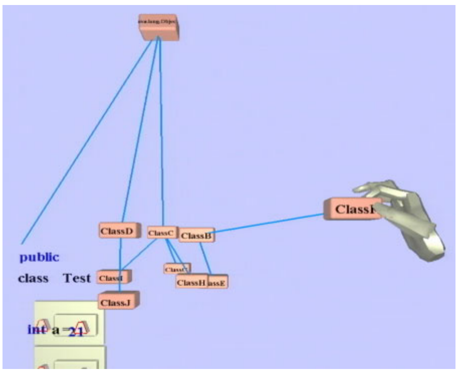


Figure 7: Manual layout of a class

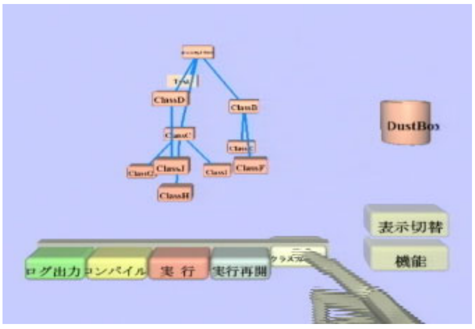
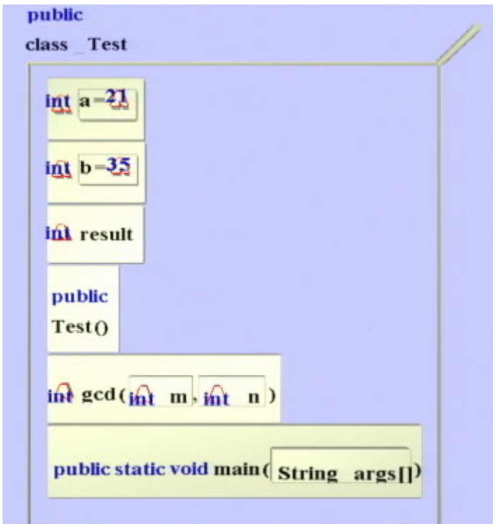


Figure 8: Automatic layout of a class hierarchy

6.2. Class

A class touched with the index finger is unfolded. When a class is unfolded, fields and methods of the class are shown. Figure 9 shows an example of an unfolded class. By touching a method in the class, the touched method is then unfolded. A folded method has only a heading part of it as shown in Figure 9.



```
public
class  Test
int a=21
int b=35
int result
public
Test()
int gcd( int  m , int  n )
public static void main( String  args[])
```

Figure 9: An unfolded class with folded methods

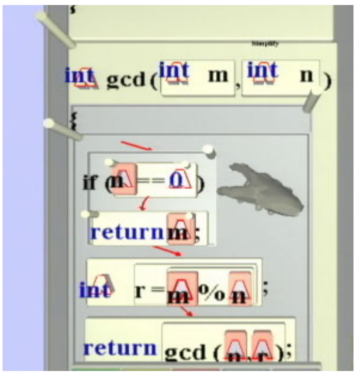
6.3. Method

When the virtual hand intersects with the regions of program elements, the nested structure and handles of the regions are shown as in Figure 10. This reveals the syntax structure of the regions. As explained, a handle is necessary for ease of operation. If it were not used, the nested structures would have to be large enough to be separated from the outer and inner structures, otherwise it would be difficult to choose a middle structure without touching neighboring ones.

One can copy an element by pinching its handle between one's thumb and index finger. When the element is pinched, the virtual hand turns yellow in color and the message of the copy is displayed in front of the hand. Figure 11 shows a copy operation.

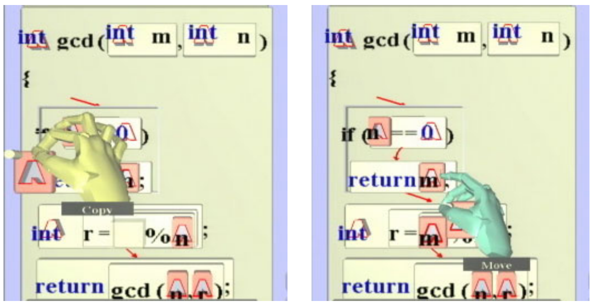
One can also move an element by pinching its handle between one's thumb and middle finger (instead of the index finger). Similar to the copy operation, the virtual hand is green in color while a move is performed. The message of the move is also displayed. Figure 11 also shows a screenshot of a move operation.

Although both textual representation and graphical nested structures are usually shown in a method, one can also turn off almost of the graphical representation. A screenshot of a textual view is shown in Figure 12. This textual view is similar to a usual Java program. Therefore one can learn textual programming using *Ougi*.



```
int gcd( int  m , int  n )
{
if (m == 0)
return m;
int  r = m % n ;
return gcd ( n , r );
```

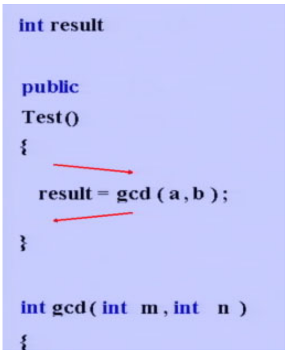
Figure 10: Handles



```
int gcd( int  m , int  n )
{
if (m == 0)
return m;
int  r = m % n ;
return gcd ( n , r );
```

(a) copy (b) move

Figure 11: Copy and move operation using a handle



```
int result

public
Test()
{
  result = gcd ( a , b );
}

int gcd ( int  m , int  n )
{
```

Figure 12: Textual view of a class (with a flow representation)

6.4. Debugging Support

Several debugging functions are supported. One can set a breakpoint in a program. The breakpoint has to be on an arrow sign between statements. Figure 13 shows a screenshot when a breakpoint is put on a control flow.

One can execute a program by using a button on a 3D panel. When an instance is created while the program is running, the instance can be displayed in the same virtual space. Figure 14 shows a screenshot when an instance is created after the program starts. The small box at the right of the figure is a newly created instance. As the execution progresses, the live fields or activation frames of a called method are displayed in the box (Figure 15 and Figure 16).

When a program is stopped at a breakpoint, the values of variables on statements adjacent to the breakpoint are automatically displayed. Although the values of all variables can be displayed, such a display is often misleading because the target language is a procedural language. Therefore we show only values of the variables on statements adjacent to the stopped point.

One can also inspect the value of a live variable manually. Figure 17 shows a screenshot of a variable inspection. When you touch a variable on the activation frame with your index finger, the value of the variable is shown. Touching the variable again turns off the display of the value.

When a method is called recursively, multiple activation frames can be displayed as shown in Figure 18. Variables on activation frames have their own values. One can investigate the changes on activation frames by navigating the program space and touching them.

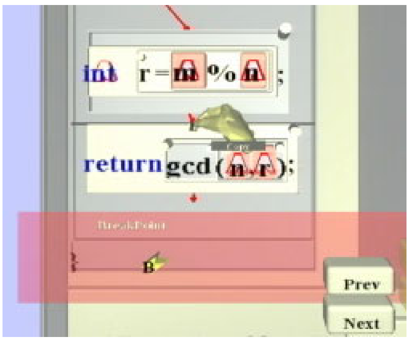


Figure 13: Setting a breakpoint on a control-flow arrow between statements

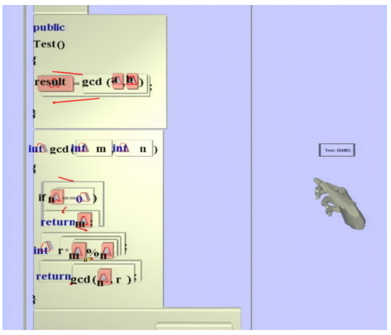


Figure 14: Program execution and a created instance

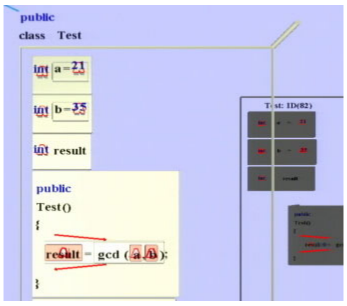


Figure 15: A class and an instance

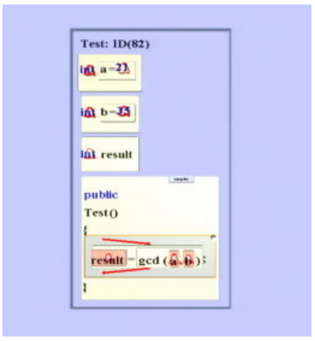


Figure 16: An instance with live fields and an activation frame of a method

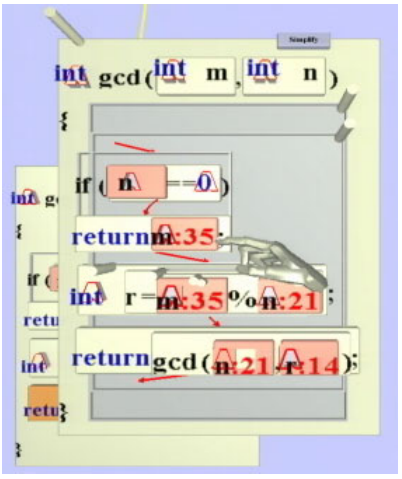


Figure 17: Inspection of a variable in an activation frame

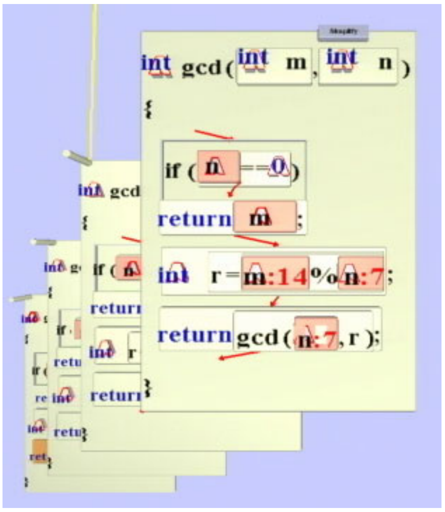


Figure 18: Activation frames when a method is called recursively

7. Implementation

The prototype system works in a virtual reality environment called TEELeX (Tele-Existence Environment for Learning eXploration) [1] at the National Institute of Multimedia Education in Japan.

A prototype system runs on a PC workstation (Compaq AP550 with dual 1-GHz processors and an Elsa Synergy III graphics board supporting dual displays). A Six-DoF position tracker (Polhemus Fastrak) and sensor gloves (Virtual Technologies CyberGlove) are used to detect the position and motion of the user's body and hands. The hardware configuration of the prototype system is given in Figure 19.

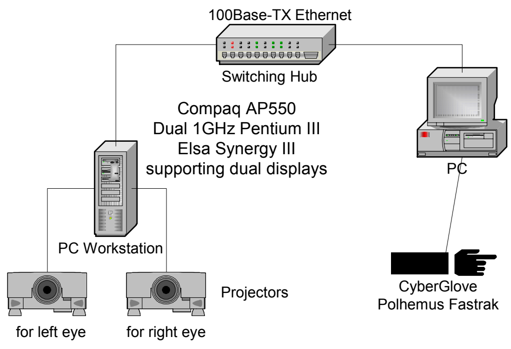


Figure 19: Hardware configuration of prototype system

The prototype system has been developed using the Java programming language[4], the Java 3D class library[16], and the *it3d* library (Interactive Toolkit library for 3D applications)[13]. *It3d* is briefly explained in subsection 7.2. The use of Java enhances the portability of the system, which should work on a wide range of computer systems.

The basic software structure of the system is target language-neutral. In other words, it does not depend on the Java as the target language because the internal software structure is based on a design pattern of model/view/controller, and basic view elements are prepared. A new language can be easily accommodated by developing the models of the language and their views which are specific to the language.

7.1. TEELeX

TEELeX is a kind of surround display system. It has a large cubic screen for immersion. Each face is 3 meters by 3 meters. Circular polarization is employed to give a stereoscopic view to users, who only need to wear lightweight stereo glasses. Two video inputs are used to give a stereoscopic view on each side. One video input is for the right eye and the other for the left eye. The prototype system uses one stereoscopic face.

Figure 20 shows a schematic diagram of TEELeX. A snapshot when one uses *Ougi* in TEELeX is given in Figure 1.

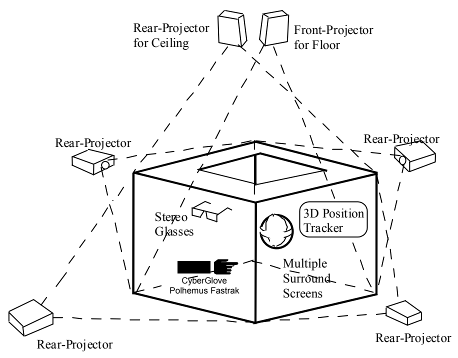


Figure 20: Schematic diagram of TEELeX

7.2. It3d

It3d[1] is an interactive toolkit library for 3D applications utilizing artificial reality (AR) technologies[13]. It was implemented using the Java language and the Java 3D class library to enhance portability. *It3d* makes it easy to construct distributed applications that are portable and adaptable. It consists of three sub-libraries: an input/output library for distributed devices, a 3D widget library for multimodal interfaces, and an interaction recognition library.

8. Discussion and Future work

Direct manipulation in an immersive 3-D space is more intuitive than on a 2-D plane since direct manipulation in 3-D is nearer to that in real life than GUI operations in 2-D. We can understand the location of a virtual object and manipulate it with stereoscopic view in an immersive space. Handle representation of nested structures enables us to edit complex hierarchical structures using direct manipulation.

The visualization and manipulation methods used in *Ougi* are not limited to a Java program. We will extend our immersive editing system to a cooperative editing system for general structured information such as XML and UML. We think that high-level editing functions will support the cooperative design of experts.

Eighteen students used the prototype system for a short period of time (about 1 hour). Since the number of hand gestures is small, students could learn direct manipulation and gestures easily. They said that programming in an immersive virtual environment is interesting and they were willing to try using it to make a program. Thus we believe that *Ougi* can enhance motivation to study programming. The students used the system without a calibration of the sensor glove. Since the size of a hand depends on a student, the virtual hand does not match the real hand for some students. Therefore gestures recognition failed in some occasions

[1] *it3d* can be accessed through <http://www.nime.ac.jp/it3d> (in Japanese) or <http://www.nime.ac.jp/it3d/index-e.html> (in English).

for some students. We need sensor glove calibration before the use of the system although it takes a little time to perform the sensor glove calibration.

We will conduct formal experiments to evaluate the effectiveness of our 3D visualization method, manipulation method and learning of programming using physical movement in an immersive environment. We also plan to design and develop a new immersive programming language that is more suitable for multimodal interfaces in immersive environments than a subset of Java.

Furthermore, we are currently implementing multiple-focus visualization and navigation functions[8] for Focus+Context views in *Ougi* by using heat models[10] to support visualization and manipulation of large-scale programs for experts. If the functions are implemented, the system can help a programmer investigate and debug the large-scale program in an immersive program space.

9. Summary

We have developed a prototype immersive programming system utilizing multimodal interfaces. It uses both textual and graphical representations and takes advantage of the merits of both. Moreover, it features unique 3D jigsaw-puzzle-like graphical representation for complex semantic constraints such as inheritance hierarchy. This feature enables users to understand grammatical constraints merely by looking at their shapes. We also developed nested-board representation with handles for direct manipulation of program elements in an immersive environment. This representation can be applied to immersive editing of structured documents such as XML. Since the interaction techniques such as direct manipulation, hand gestures, and handles of regions, which are employed in the system, are easy to learn, a beginner can start to learn the essence of programming without time-consuming study of the operations of a system.

Acknowledgements

The authors would like to thank Mr. Norio Takase and Mr. Takashi Tohyama for working with us to develop *Ougi*.

This research was partially supported by a Grant-in-Aid for Scientific Research (14380090 and 11358002) in Japan.

The toolkit library "*it3d*" was developed with funding by the Support Program for Young Software Researchers in 2000, which was implemented by the Research Institute of Software Engineering (RISE) commissioned by Information-technology Promotion Agency (IPA) in Japan.

References

[1] Kikuo Asai, Noritaka Osawa, and Yuji Y. Sugimoto, “Virtual Environment System on Distance Education,” *Proc. of EUROMEDIA '99*, pp. 242-246, 1999.

[2] M.H. Brown and M.A. Najork, “Algorithm animation using 3D interactive graphics,” *ACM Symp. on User Interface Software and Technology*, pp. 93-100, 1993.

[3] Margaret M. Burnett, Adele Goldberg and Ted G. Lewis, *Visual Object-Oriented Programming: Concepts and Environments*, Manning, 1995.

[4] James Gosling, Bill Joy and Guy Steele, *The Java^{TM} Language Specification*, Addison-Wesley, 1996.

[5] Eades, P. "A Heuristic for Graph Drawing," Congressus Numerantium, Vol.42, pp.149-160, 1984.

[6] H. Lieberman, “A three-dimensional representation for program execution,” *IEEE Workshop on Visual Languages*, pp. 111-116, 1989.

[7] Marc A. Najork, “Programming in Three Dimensions,” *Journal of Visual Languages and Computing*, Vol. 2, No. 7, pp. 217-242, 1996.

[8] Noritaka Osawa, Kikuo Asai, Yuji Y. Sugimoto, “Immersive Graph Navigation Using Direct Manipulation and Gestures,” *Symposium on Virtual Reality Software & Technology 2000* (VRST 2000), pp. 147-152, 2000.

[9] Noritaka Osawa, “Generation and Evaluation of Glyph Representing Superclass-subclass relationships,” *Proc. of IEEE Symp. on Visual Languages*, pp. 81-82, 2000.

[10] Noritaka Osawa, "A Multiple-Focus Graph Browsing Technique Using Heat Models and Force-Directed Layout," 5th International Conference on Information Visualisation (IV2001), pp.277-283, 2001-7.

[11] Noritaka Osawa, "Visualization of Inheritance Relationships By Using Glyphs", IEICE Trans. on Information and Systems, IEICE Trans. on Information and Systems, Vol. E85-D, No.1, pp.275-282, 2002. <http://search.ieice.org/2002/files/e000d01.htm#e85-d,1,275>

[12] Noritaka Osawa, Kikuo Asai, Yuji Y. Sugimoto, and Fumihiko Saito, "A Dancing Programmer in an Immersive Virtual Environment," Symposia on Human-Centric Computing Languages and Environments (HCC2001), pp.348-349, 2001.

[13] Noritaka Osawa, Kikuo Asai, and Fumihiko Saito, “An Interactive Toolkit Library for 3D Applications: it3d,” Eighth Eurographics Workshop on Virtual Environments (EGVE2002), pp.149-157, May 2002.

[14] F. Van Reeth and E. Flerackers, “Three-dimensional graphical programming in CAEL,” *IEEE Symp. on Visual Languages*, pp. 389-391, 1993.

[15] J. Stasko and J. Wehrli, “Three-dimensional computation visualization,” *IEEE Symp. on Visual Languages*, pp. 100-107, 1993.

[16] Henry Sowizral, Kevin Rushforth and Michael Deering, *The Java 3D API Specification,* Addison Wesley, 1998.