Eighth Eurographics Workshop on Virtual Environments (2002)
S. Müller, W. Stürzlinger (Editors)

An Interactive Toolkit Library for 3D Applications: it3d

Noritaka OSAWA[†*], Kikuo ASAI[†], and Fumihiko SAITO[‡]

[†]National Institute of Multimedia Education, JAPAN
[*]The Graduate University of Advanced Studies, JAPAN
[‡]Solidray Co. Ltd, JAPAN

Abstract

An interactive toolkit library for developing 3D applications called "it3d" is described that utilize artificial reality (AR) technologies. It was implemented by using the Java language and the Java 3D class library to enhance its portability. It3d makes it easy to construct AR applications that are portable and adaptable. It3d consists of three sub-libraries: an input/output library for distributed devices, a 3D widget library for multimodal interfacing, and an interaction-recognition library. The input/output library for distributed devices has a uniform programming interface style for various types of devices. The interfaces are defined by using OMG IDL. The library utilizes multicast peer-to-peer communication to enable efficient device discovery and exchange of events and data. Multicast-capable CORBA functions have been developed and used. The 3D widget library for the multimodal interface has useful 3D widgets that support efficient and flexible customization based on prototype-based object orientation, or a delegation model. The attributes of a widget are used to customize it dynamically. The attributes constitute a hierarchical structure. The interaction-recognition library is used to recognize basic motions in a 3D space, such as pointing, selecting, pinching, grasping, and moving. The library is flexible, and the recognition conditions can be given as parameters. A new recognition engine can be developed by using a new circular event history buffer to efficiently manage and retrieve past events. Development of immersive AR applications using it3d demonstrated that less time is needed to develop the applications with it3d than without it. It3d makes it easy to construct AR applications that are portable and adaptable.

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Software support

1. Introduction

Distributed processing has been increasingly put to practical use as computers and network equipment have become less expensive and more capable. However, the main contents currently used in distributed processing are text documents, 2D still images, 2D video, and audio. As computers and networks continue to become commodity devices, however, distributed processing will become even more widely used and easier to implement. This will encourage the distribution of interactive 3D content. First, though, we need to make 3D information visualization and manipulation easier and more flexible to facilitate the adoption of such rich and advanced interactive content.

Many artificial reality (AR) technologies (including virtual reality and augmented reality) have moved to the production level from the laboratory level, and there are several multimodal input/output devices on the market. However, developing AR applications is more difficult than developing 2D or desktop applications because appropriate toolkit libraries are not sufficiently available for many platforms. More useful and portable toolkits for developing interactive applications are therefore needed to facilitate the research and development of AR applications and the rapid prototyping of advanced applications. We have developed an interactive toolkit library for 3D applications, that we call *it3d*[*]. It is implemented with the Java programming language[6] and the Java 3D class library[18] in order to improve the portability of applications and *it3d* itself. *It3d* has high-level functions such as 3D widgets and gesture recognition, and it uses multicast communication for efficient and flexible data exchange.

Related work is discussed in Section 2. Section 3 gives an overview of *it3d*; the features and functions of the *it3d* sub-libraries are described in Sections 4, 5, and 6. Section 7 explains how we used *it3d* to develop practical AR applications and why it is particularly useful. Our plans for future work are given in Section 8, and Section 9 summarizes this paper.

[*] Refer to http://www.nime.ac.jp/~osawa/research/it3d/ (in English) or http://www.nime.ac.jp/it3d (in Japanese, as of writing). The library can be accessed through http://www.nime.ac.jp/it3d

© The Eurographics Association 2002.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

2. Related Work

There are several 3D graphics libraries and toolkits. Some of them are low-level 3D graphics libraries such as OpenGL[17] and DirectX[8]. Java3D[18] and Performer[15] are scene-graph-based 3D graphics toolkits at a slightly higher level than OpenGL and DirectX. Although low-level 3D graphics libraries and scene-graph-based 3D graphics libraries support fine rendering control, they are too low-level and unsuitable for rapid prototyping of interactive AR applications because it takes too long to develop practical AR applications using only their rendering toolkits. They do not provide high-level functions such as widgets in a 2D GUI. Therefore, programmers have to develop similar high-level functions for each application. This means the user interfaces of AR applications often lack uniformity and are confusing.

Libraries such as Open Inventor[16], CAVE library[4], World Toolkit (WTK)[13], MR Toolkit[14], Avocado[19], and VR Juggler[2] have higher-level functions than do OpenGL, DirectX, Java3D, and Performer.

Open Inventor works on SGI IRIX and Linux operating systems. There are also its ports to the Windows operating system. The CAVE library requires the CAVE hardware or some other variation. At present, these are not platform-independent toolkits.

WTK is a cross-platform system for building 3D applications for scientific and commercial use. However, customization requires programming in C or C++, re-compilation, and re-linking. This impedes the quick modification of user interfaces. In contrast, *it3d* supports dynamic customization based on prototype-based object-orientation.

MR Toolkit has many useful packages. Although it supports C, C++ and FORTRAN applications, it does not use a language-neutral IDL such as OMG IDL to define communication interfaces. It used unicast communication.

Avocado is a framework for distributed virtual reality applications. It is based on C++ and the Performer API. Since it focuses on the data distribution, it does not have high-level 3D widgets.

VR Juggler is a C++ class library and a platform for virtual reality applications. It has a runtime configuration management tool. It does avoid rendering functions intentionally to enable it to work with various rendering libraries. Therefore, it does not have high-level 3D widgets for an interactive user interface. In contrast, *it3d* has various useful 3D widgets and they can be used to quickly develop applications.

Since many conventional 3D toolkits do not have interaction-recognition libraries, programmers need to develop a recognition engine or combine an existing recognition engine with their applications. This may stand in the way of establishing a uniform user-interface for AR applications. On the other hand, *it3d* has a basic gesture-recognition library.

3. It3d

It3d is an interactive toolkit library for developing 3D applications utilizing AR technologies. In this section, we explain our goals in developing *it3d* and its overall structure.

3.1. Goals

The main purpose of *it3d* is to enable rapid prototyping for the research and development of interactive 3D applications, especially for information visualization[3] and the manipulation of abstract information. Our goals for *it3d* are as follows.

- Rapid prototyping

  Development of a toolkit library for multimodal interactive 3D applications in distributed environments.

- Enhanced portability

  Implementation using the Java programming language and Java 3D class library. Unification of device interfaces at the network level.

- Customizable high-level function components

  Development of highly customizable 3D components, or widgets such as 3D buttons and combo-boxes. Development of customizable interaction recognition functions.

3.2. Structure

It3d is composed of three sub-libraries: an input/output library for distributed devices, a 3D widget library for multimodal interfacing, and an interaction recognition library (The relationships between them are illustrated in Figure 1). We will briefly overview the features and functions of the sub-libraries in this subsection, and then describe them in more detail in the following sections.

© The Eurographics Association 2002.

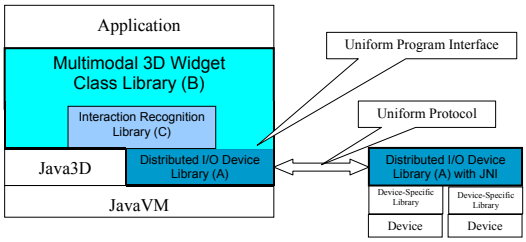


Figure 1: Relationships between *it3d* sub-libraries

The input/output library for distributed devices has a uniform programming interface style suitable for various devices. The device interfaces are defined with OMG IDL[20], so the events and data of various devices can be exchanged in a uniform and efficient form. The library uses multicast peer-to-peer communication to enable efficient device discovery for distributed plug-and-play and the exchange of events and data.

The 3D widget library for multimodal interfacing has useful 3D widgets that support efficient and flexible customization based on prototype-based object orientation, or a delegation model. The attributes of the widgets are used to dynamically customize them. The attributes constitute a hierarchical structure. The 3D widget library includes useful 3D user-interface components and a 3D layout manager for automatic placement of 3D components. Rendering of 3D objects is performed by Java 3D library, which uses OpenGL or DirectX.

The interaction-recognition library has recognition functions for 3D direct manipulation and gestures. It recognizes basic motions – such as pointing, selecting, pinching, grasping, and moving – in a 3D space. The library is flexible, and the recognition conditions can be given as parameters.

As mentioned, we developed *it3d* using the Java programming language[6] and Java 3D class library[18] in order to enhance its portability.

4. Distributed input/output device library

The various multimodal input/output devices have different native application-program interfaces (APIs). The operating environment in which their device drivers will work depends on specific computers and operating systems. In other words, they are not portable. It is very difficult to make all of them work on one computing platform. Therefore, a distributed I/O device interface is required. The programming interfaces of these devices should be unified at the network level. A uniform interface for the same kind of devices is also needed to enable efficient programming and to enhance application portability.

It3d absorbs the diversity in device interfaces and provides programmers with a uniform interface. In other words, this library not only wraps native programming interfaces of input/output devices, but also gives programmers a uniform application interface for the same kind of functions.

The input/output library we developed for distributed devices is based on a model in which applications are separated from devices at the network level and the interfaces of the different devices are integrated. This model enables applications to work in different environments that have different devices. The library uses multicast communication to permit efficient distribution and exchange of events and data.

4.1. Multicast peer-to-peer communication

We use multicast communication between devices and applications because multicasting enables efficient device discovery and management. Unlike with unicast communication, a central server is not needed for device.

First, multicasting is used to find appropriate devices in a network. A fixed multicast address is used for device discovery where an application can obtain information or events from devices of interest. Each device dynamically acquires a unique multicast address for its own use. A coordination protocol for dynamic multicast address allocation is implemented in the library. In addition to device discovery, an application communicates with devices through multicasting.

In a laboratory, classroom, or office, we can reasonably assume that devices can be connected with a fast- switching network such as a 100Base-TX or 1000BASE-T Ethernet LAN. A fast-switching network usually supports full-duplex communication and flow-control, thus it is collision-free unless the network traffic overflows the throughput of the network. Moreover, the data or events are often sent periodically from devices to an application. Therefore, we do not need to worry about error correction when using one-way multicasting in many cases. Since fast-switching network devices are now relatively inexpensive, it is also not important to consider legacy network devices such as 10Base-5 Ethernet LAN or dumb hubs.

Multicasting enables different computers to monitor the status of a device. This often helps to find a problem in the system. A distribution server or a packet-relay function is not needed.

Moreover, each computer can configure devices remotely. One computer can execute an application and another computer can be used to configure the devices. This eases the device configuration. Configuration updates are transmitted from configured devices to applications through multicasting as an event. Therefore,

© The Eurographics Association 2002.

the applications can be easily kept up to date on the configuration.

4.2. Uniform interfaces and protocols

The interfaces of distributed devices are defined using OMG IDL[20] which was designed for CORBA (Common Object Request Broker Architecture)[21].

Although we implemented drivers in the system using the Java programming language, the Java Native Interface (JNI), and some native codes, device-specific drivers may work only on a platform that does not support the Java runtime environment. Because of this, we developed an interface program written in a programming language other than Java. We used IDL because it supports multiple language bindings.

The standard Java libraries include CORBA functions, but do not support multicast communication. We therefore developed a multicast CORBA support library in which all calls are one-way and return values cannot be obtained automatically. Our distributed I/O device library has routines that wrap up CORBA functions and provide an interface for remote procedure calls that request an invocation, wait for a result, and then send the request again if a result is not returned within a certain time limit.

4.3. Supported Devices

As of this writing, the following devices are supported by *it3d*.

- Mouse (2 or 3 DoF + buttons)
- Joystick (more than 2 DoF + buttons)
- Microsoft DirectInput devices (Drivers that work on the Windows operating systems)
  - Mouse
  - Joystick (with force-feedback)
  - Game pad
  - Wheel (with force-feedback)
  - USB human interface devices
- Sensor gloves
  - CyberGlove
  - CyberTouch (with vibration)
  - CyberGrasp (with force-feedback)
- Position/orientation tracker (6 DoF)
  - Polhemus Fastrak
- 3D mouse (6 DoF)
  - Logitech Magellan
- Eye-movement tracker (2 DoF)
- Thermal feedback device (original)
- Remote MIDI file player

Data received from a device is reported to client applications using a delegation-listener model which is the standard programming style in the Java library. The application must specify the listener method of events that the application wants to receive in the model. A listener interface for each data type is specified and an I/O library class for clients usually implements the multiple appropriate listener interfaces. Thus, a client application can received all events through uniform event-listener interfaces. For example, an application can receive 2DoF events from a mouse and a joystick using the same listener method.

Each device has its own properties and the parameters of a device are usually stored in its properties. The names of properties for common parameters are organized and uniform among devices. Therefore, a client application can set the common parameters of similar devices in a uniform way.

5. Multimodal 3D widget library

The 3D widget library for multimodal interfacing includes several useful 3D widgets, some of which correspond to 2D GUI widgets. The 3D widgets can handle the events related to hand interaction, and thus support direct manipulation by hand in virtual 3D space.

The design of our multimodal 3D widget library is based on a prototype-based object-oriented model[9], or a delegation model. A widget's attributes are used to customize it, and they can be shared using a delegation mechanism. A prototype-based object-orientation model is suitable for dynamic customization. The multimodal 3D widget library is thus easily customizable.

5.1. Prototype-based object-orientation

In the prototype-based object-oriented model, there are no differences between instances and classes. All objects are instances and can be prototypes of new objects. A new object is created by cloning an existing object or a prototype. Needless to say, in a class-based object-oriented model, an instance is usually created from a class.

An attribute in an object is found through delegation. If an attribute is not found in an object, it is searched for in the parent (prototype) object. Thus, cloned objects can share attributes with their ancestors. The search for an attribute is done at runtime. Although this may degrade the responsiveness of an application, this has not been a problem in our experience (discussed in Section 7).

The prototype-based model is suitable for dynamic customization during runtime as stated. When an attribute of a parent or a prototype is changed, the attribute of its children may be affected by the change. Figure 2 shows an example. The top left toggle switch is a parent switch and the other switches are children cloned from the parent. In this case, when the parent switch is toggled, the children are also toggled. When a child switch is toggled directly, it gets its own state and

© The Eurographics Association 2002.

the state can be changed independently of the parent-switch state.

We can easily modify the natures of widgets, such as their appearances, by using a prototype mechanism. When the color attribute of a parent object is modified, all children objects have the same color as the parent object unless their color has been modified directly.

Sharing is done for each attribute. Even if a color of a child is modified directly, its behavior is still shared with its parent. Of course, when the state of the child's behavior is modified directly, the new behavior is not shared with its parent.

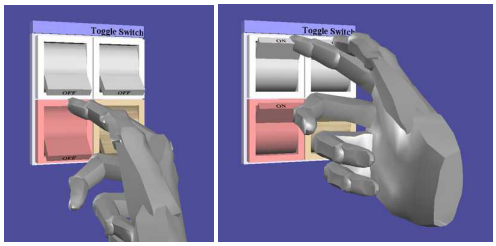


Figure 2: Attribute sharing through delegation

5.2. Hierarchical attributes

In our prototype-based model, 3D components have hierarchical attributes such as a hierarchical file system. In a programming language, simplicity is important, so the structure of attributes in an object is flat. However, 3D widgets generally need a number of attributes because of their many parameters, and a flat attribute structure complicates the management of attribute names. This is the same problem encountered in flat and monolithic file systems with a large number of files.

A hierarchical system is more suitable for non-trivial scale attribute structures and can easily handle a group of attributes in an orderly way.

An example of the attribute structure is shown in Figure 3. Leaf nodes usually have attribute values. Intermediate nodes usually work as directory nodes although they can have their own attribute values. The hierarchical components can also be accessed by a string as they can in a typical hierarchical file system.

The access mechanism is implemented by using a hash table internally. Although the hash table is a flat mechanism, the intermediate nodes of standard attributes are handled specially, which enables virtual hierarchical handling.

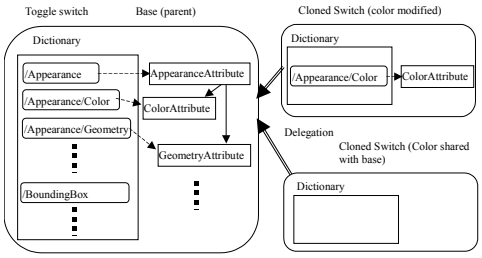
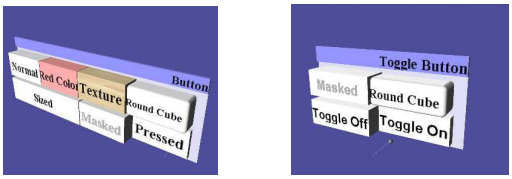
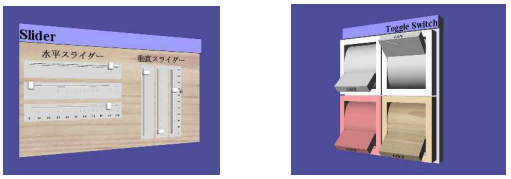


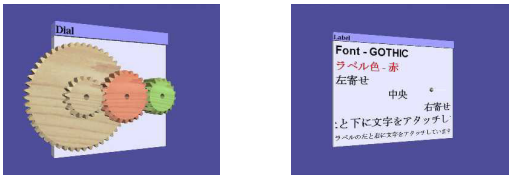
Figure 3: Hierarchical attributes and delegation



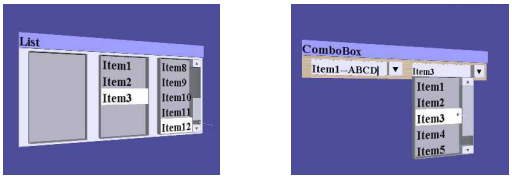
button toggle button



slider toggle switch



dial label



list combo box

Figure 4: Examples 3D widgets

5.3. 3D widgets

We think a toolkit library should have various components if it is to be useful. Without an adequate number of functions, prototyping can be tedious and

© The Eurographics Association 2002.

time-consuming. Therefore, we implemented various high-level 3D widgets in *it3d*. Some are 3D widgets corresponding to 2D GUI components and others are 3D-specific widgets. (Example 3D widgets are shown in Figure 4.) As discussed above, the shape and appearance of the widgets can be easily modified by changing the appropriate attributes.

5.4. 3D layout

A 3D container can contain 3D components or widgets. Moreover, it can have a 3D layout manager that controls the layout of the 3D components in the container. Although this layout approach is the same as that used in AWT and Swing in 2D, a 3D layout is more complicated than a typical 2D layout.

We have developed a line-up layout manager and a force-directed layout manager. The line-up layout manager is a 3D version of a flow layout manager and a grid layout manager in 2D. The force-directed layout manager places the 3D components in the 3D space on the basis of kinematical constraints between them[5,7,10]. When the constraints between the components are appropriately specified, the force-directed layout manager can automatically arrange the components in the 3D space. Most other toolkit libraries do not have this force-directed layout functionality. An example of the automatic force-directed layout of a graph where repulsive forces, elastic forces, and gravitational force were exerted is shown in Figure 5.

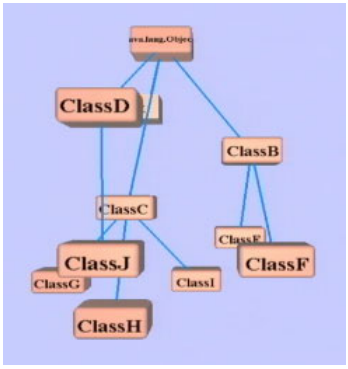


Figure 5: Automatic layout of a tree by a force-directed layout manager

6. Interaction-recognition library

The interaction-recognition library has functions for recognizing basic interactions and gestures – such as pointing, selecting, pinching, grasping, and moving – in a 3D space. The library generates high-level events as recognition output. This library relieves application programmers of the need to make a recognition engine for direct manipulation and gesture recognition of virtual objects and 3D widgets. The library is flexible, and the recognition conditions can be given as parameters. A new recognition engine can be implemented by combining the existing library functions with newly developed functions.

In this paper, basic interactions are categorized into direct manipulation and gestures. Direct manipulation is an interaction between virtual objects and pointers such as fingers. Gestures are interactions that do not relate directly with virtual objects.

Direct-manipulation functions of the interaction-recognition library allow one to manipulate virtual objects or 3D components using interface devices such as a sensor glove. In other words, get-and-put operations in 3D, which corresponds to drag-and-drop operations in a 2D GUI, can be easily used in the implementation of 3D applications. Since the model of events is similar to that in a 2D GUI, a 2D GUI can also control 3D components easily, although the degrees of freedom are often insufficient using a 2D GUI device such as a desktop mouse. To improve the accuracy of picking (get) and releasing (put) operations by a hand, the angles of the fingers used to pick up a virtual object can be used as well as the collision between the fingers and the virtual object.

The interaction-recognition library has a ring history buffer function. Gesture-recognition engines can use the ring history buffer to manage and retrieve past events (Figure 6). With the ring history buffer, a recognition engine does not need to receive all events; instead it needs to receive only key events and retrieve past events when necessary. This can reduce the event-management overhead.

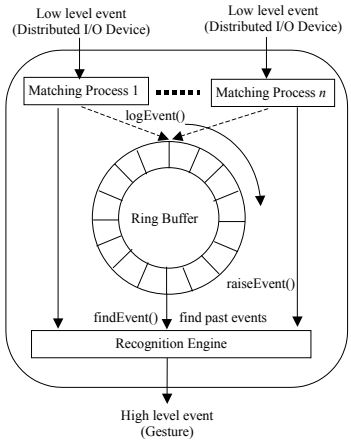


Figure 6: Gesture-recognition functions

© The Eurographics Association 2002.

In addition to the gesture-recognition engine in the library, we have developed a simple gesture-learning tool. Figure 7 shows screenshots of the tool being used. After the learning conditions are set up, a demonstrated gesture is learnt. Figure 7 show how an OK sign made with the right hand is learnt. After the gesture is learned, the tool can recognize it (Figure 8).

The conditions that can be specified in the current gesture recognition engine are the period of a gesture, the angles of the finger joints, the position and posture of the hand, and the track of the hand movement. These conditions can be learnt by using the gesture-learning tool.

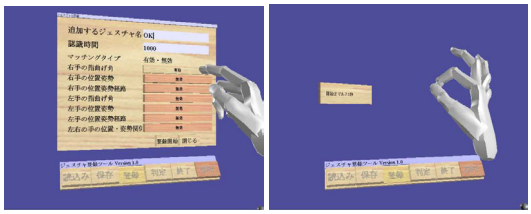


Figure 7: Screenshots of gesture learning

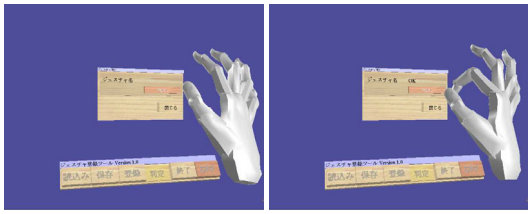


Figure 8: Screenshots of gesture recognition

7. Applications

We used developed *it3d* to develop interactive 3D applications for an immersive virtual environment called TEELeX[1]. These applications are working in the distributed I/O system configuration illustrated in Figure 9.

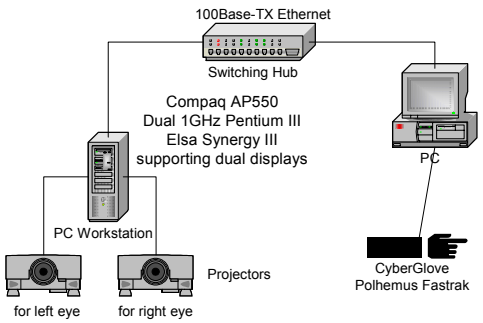


Figure 9: System configuration

7.1. Immersive music editing and playing prototype application: *Reijin*

To demonstrate the functionality and usefulness of *it3d*, we have developed an immersive application for editing and playing music that we call *Reijin*[11]. Figure 10 shows screenshots of *Reijin*.

The user can edit and play music using direct manipulation by hand. For example, notes can be manipulated with the hand. A note can be pinched by the middle finger and thumb, and moved to the desired position. When a note is pinched by the forefinger and thumb, the note is copied.

Reijin uses a number of the 3D widgets in *it3d*. A 3D combo box is used to select a file from which to load music data. Music playing is controlled with 3D buttons. The timbre can be selected from a 3D scrollable list. A 3D dial can be used to scroll through a score.

We were able to develop this application in considerably less time with the toolkit than would have been possible without it, thereby demonstrating the usefulness of the toolkit. The development also demonstrated that the distribution of devices and an application, customization using prototype-based object orientation, and interaction recognition of physical motions are useful for developing interactive AR applications.

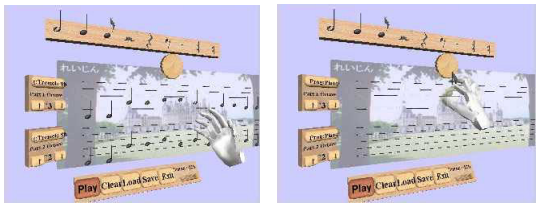


Figure 10: Screenshots of *Reijin*

7.2. Immersive programming system: *Ougi*

We have been developing an immersive programming system, called *Ougi*[12], which supports a subset of the Java programming language. It utilizes both textual and graphical 3D representation.

With *Ougi,* a user can write a program, control its execution, and debug it within an immersive environment. Implementation of *Ougi* showed that *it3d* can be used for non-trivial AR application development. Figure 11 shows a snapshot of *Ougi* being used in a TEELeX environment. Figure 12 shows some screenshots of *Ougi*.

© The Eurographics Association 2002.

Figure 11: Snapshot of *Ougi* in the TEELeX immersive virtual environment

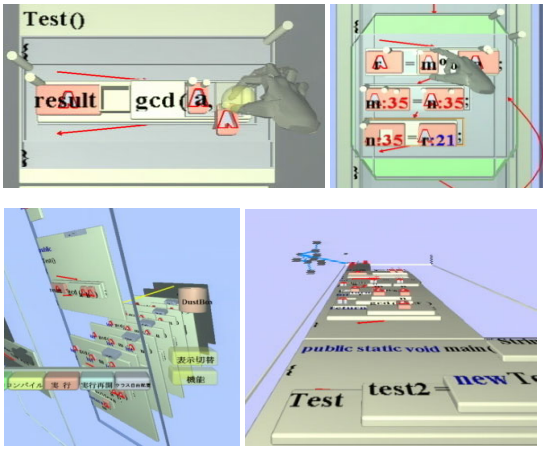


Figure 12: Screenshots of *Ougi*

8. Future Work

We will extend the supporting devices and implement more 3D widgets. We are developing distributed I/O interfaces for speech-recognition and speech-synthesis engines and for sampling sounds. We plan to integrate immersive 3D environments with portable devices such as PDAs. We also plan to develop highly functional 3D components such as circle menus, hierarchical displays (tree displays), 3D graph displays, and tabular displays.

Although *it3d* speeds up prototype development, the design and layout of the 3D widgets in the application still require a lot of time and work. This is because the design and layout have to be tuned with the programming through a 2D GUI, whereas the operating environment is three-dimensional. Hence, we plan to develop an immersive attribute editor for 3D widgets. The attribute editor will be used to customize 3D widgets through direct manipulation and gestures in an immersive 3D virtual space that is the same as the operating environment of the application. The attribute editor will make it easy to change the appearance and layout of a 3D widget.

We think that 3D applications should be component-based. Multiple functionalities or applications should coexist in a virtual 3D space. However, current 3D application frameworks do not support the coexistence of applications or the communications with them. The cooperation of 3D applications in a virtual 3D space needs a 3D user-interface management system. We will develop a system that will enable the coordination of multiple 3D applications in a virtual 3D space and a distributed 3D space. This 3D user-interface management system will contribute to the development of a standard user interface that is customizable for 3D applications and coordination between distributed cooperative applications.

9. Summary

The *it3d* interactive toolkit library for 3D applications supports various I/O devices in distributed environments and has a number of useful 3D widgets for interactive interaction. It uses multicast peer-to-peer communication and a prototype-based object-orientation model for dynamic customization. *It3d* is implemented using the Java language and the Java 3D class library, so it will work on a number of hardware and software platforms. *It3d* can shorten the development time of interactive 3D applications that are portable and adaptable.

Acknowledgements

We thank Mr. Norio Takase and Mr. Takashi Tohyama for working with us to develop *it3d*.

The *it3d* toolkit library 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 the 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. Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira, "VR Juggler: A Virtual Platform for Virtual Reality Application Development", *IEEE VR 2001*, pp. 89-96, 2001.

© The Eurographics Association 2002.

3. Stuart K. Card, Jock D. MacKinlay, and Ben Shneiderman, *Readings in Information Visualization - Using Vision to Think*, Morgan Kaufmann Publ., 1999.
4. C. Cruz-Neira, *Virtual Reality Based on Multiple Projection Screens: The CAVE and its Applications to Computational Science and Engineering*, doctoral dissertation, University of Illinois at Chicago, 1995.
5. P. Eades, "A Heuristic for Graph Drawing," *Congressus Numerantium*, Vol.42, pp. 149-160, 1984.
6. James Gosling, Bill Joy and Guy Steele, *The Java™ Language Specification*, Addison-Wesley, 1996.
7. Kamada, T., and S. Kawai, "Algorithms for drawing general undirected graphs," *Information Processing Letters*, Vol. 31, No. 1, pp. 7-15, 1989.
8. Microsoft Corporation, Direct X, <http://www.microsoft.com/directx/>.
9. James Noble, Antero Taivalsaari, and Ivan Moore, *Prototype-Based Programming: Concepts, Languages and Applications*, Springer-Verlag, 1999.
10. Noritaka Osawa, Kikuo Asai, and Yuji Y. Sugimoto, "Immersive Graph Navigation Using Direct Manipulation and Gestures," *Symposium on Virtual Reality Software & Technology 2000* (VRST2000), pp.147-152, 2000.
11. Noritaka Osawa, Kikuo Asai, Norio Takase and Fumihiko Saito, "An Immersive System for Editing and Playing Music on Network-connected Computers," *5th International Conference on Information Visualisation* (IV2001), pp.630-635, 2001.
12. Noritaka Osawa, Kikuo Asai, Yuji Y. Sugimoto, and Fumihiko Saito, "A Dancing Programmer in an Immersive Virtual Environment," *Symp. on Human-Centric Computing Languages and Environments* (HCC2001), pp.348-349, 2001
13. Sense8, WorldToolkit, <http://www.sense8.com/products/index.html>.
14. Chris Shaw, Mark Green, Jiandong Liang and Yunqi Sun, "Decoupled Simulation in Virtual Reality with the MR Toolkit," *ACM Trans. on Information Systems*, 11(3),
15. Silicon Graphics, Inc., OpenGL Performer, <http://www.sgi.com/software/performer/>
16. Silicon Graphics, Inc., Open Inventor, <http://oss.sgi.com/projects/inventor/>.
17. Dave Shreiner (Editor), *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.2*, Addison-Wesley, 1999
18. Henry Sowizral, Kevin Rushforth and Michael Deering, *The Java 3D API Specification,* Addson-Wesley, 1998.
19. Henrik Tramberend, "Avocado: A Distributed Virtual Reality Framework," *IEEE Virtual Reality '99*, pp.14-21, 1999.
20. Object Management Group, Java Language Mapping to OMG IDL Specification, <http://www.omg.org/technology/documents/formal/java_language_mapping_to_omg_idl.htm>
21. Object Management Group, *The Common Object Request Broker: Architecture and Specification*, John Wiley & Sons, (1992)

© The Eurographics Association 2002.