Shared Virtual Reality Interior Design System

Tomi Korpipää[123], Koichi Minami[3],
Tomohiro Kuroda[13], Yoshitsugu Manabe[13], Kunihiro Chihara[13]
[1] TAO Nara Research Center
[2]VTT Electronics, Oulu, Finland
[3] Nara Institute of Science and Technology
Tomi.Korpipaa@ele.vtt.fi, {koich-mi, tomo, manabe, chihara}@is.aist-nara.ac.jp

Abstract

Traditional methods in interior design usually lack depth and sense of realism, as well as require the designer and the client to meet in one place. These problems can be solved by utilizing shared virtual reality in the design process. This paper proposes an interior design system using remote heterogeneous virtual reality platforms. Using the system, the designer and the client can work together without the need to meet in the same place. The proposed system can be used to greatly enhance the feeling of presence.

To realize a useful shared virtual environment, a portable graphics engine is required to allow running the system in a number of hardware configrations, allowing using the resources available. Also, a smart network protocol is needed to ensure smooth operation and avoid unnecessary delays. This paper introduces a portable and configurable 3D engine developed for the system as well as a non-locking network protocol to realize the shared space.

Keywords: Shared, Virtual Reality, Network protocol, Interior design

1 Introduction

The idea for the system was initiated from the notion of hardships in modern interior design. Nowadays interior design is usually made by the designer and the client both being in the same place, which requires some traveling for at least one of them. Traditional methods also lack depth and sense of realism and require quite a bit of imagination to comprehend what the result actually would look like in the real environment.

To solve this problem, this paper proposes an interior design system using heterogeneous virtual reality platforms. Using the system, the designer and the client can work together without the need to meet in the same place. The designer can stay in his office and the client in a place convenient for him, for example nearest place offering a virtual reality platform, or even at his own home. Moreover, the designer can access extensive furniture database right in his office. This paper introduces a test arrangement of the proposed system.

The proposed system can be used to reduce the problems mentioned by greatly enhancing the feeling of realism. However, developing such a system has several aspects and a number of technical problems. The main topics in this paper are the graphics engine and the network protocol developed for the system.

2 System Overview

The system under development aims to realize a network protocol and a 3D graphics engine that allow the same virtual space be used in two or more remote systems even of significant performance difference. In a trial-and-error situation such as interior design, the network protocol has to be able to maintain the coherency of the virtual space dependless on what users are doing in separate environments. Graphics engine on the other hand has to be easily portable to different operating systems and visualization systems.

The system allows client users to perceive the same space at the same time, ie. in real-time. The users are also able to see each other as avatars, move inside the virtual space and communicate through the avatars. Users are also able to make modifications to the virtual space, such as add, remove and move the furniture, and the changes can be perceived by all clients almost simultaneously.

3 Test System Equipment

The test system arrangement uses two remote immersive Virtual Reality platforms. One end of the system is a CYLINDRA [1] at the Information Science Department building at Nara Institute of Science and Technology (NAIST) in Nara, Japan. The other end is an Immersive Multi-Display System in Telecommunications Advancement Organization of Japan Nara Research Center (TAO NRC) near Nara Institute of Science and Technology. These two Virtual Reality platforms are connected via a 150Mbps optical network. Picture of the test system arrangement on a whole can be seen in figure 1.

NAIST's CYLINDRA system consists of 6 CRT video projectors and an 8-CPU SGI Onyx2 with a 2-graphics pipe InfiniteReality2 graphics subsystem. Display is a

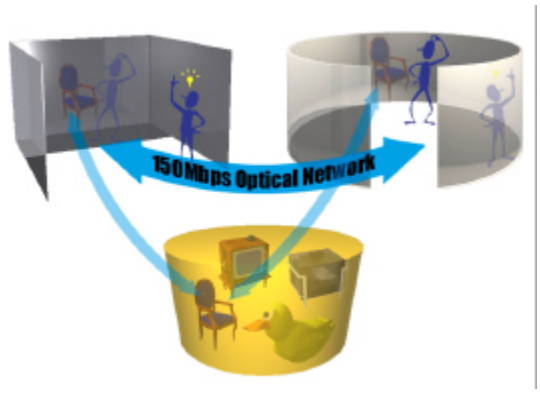

Fig 1. Test system arrangement

330-degree cylindrical wall, 6 meters in diameter and 2.4 meters high; see figure 2. CYLINDRA can produce stereo view for LCD shutter glasses by altering image for left and right eye. A simulated view from cylindrical display setting can be seen in figure 3 and an actual photo in figure 4.

The Immersive Multi-Display System consists of 8 LCD video projectors, 8 400MHz Pentium II PCs and a fast local network. Display consists of back wall, left and right side wall and a partial front floor; see figure 5. Stereo view can be produced for polarized glasses using 2 video projectors with polarized lenses for each screen, one computer handling the output of each projector. In the test arrangement only 3 projectors and 3 computers are used, as stereo view for this platform is not implemented and floor screen is not used. A simulated view from the Immersive Multi-Display System display setting can be seen in figure 6 and an actual photo in figure 7.

the graphical simulation and user interfaces, and uses the network protocol for sharing the virtual space.

The 3D Engine has to be portable, scalable and easily configurable to allow using it in several different systems and display configurations. Portability and scalability set serious restrictions for technologies that can be used, and easy configurability calls for ability to control almost all things through configuration files or at run-time.

At the moment, 3D Engine is programmed in C using OpenGL [4, 5] for graphics routines and GLUT (OpenGL Utility Toolkit) [6, 7] and GLX (OpenGL for X Window System) [8, 9] for window system dependent code used for rendering window and input device handling.

GLUT was originally chosen as the only window handling code to be used in the engine because of its availabity to several operating systems [7]. However, it has some serious restrictions, namely no support for

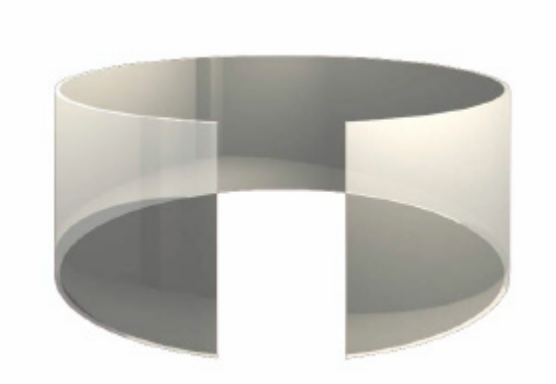

Fig 2. CYLINDRA display setup

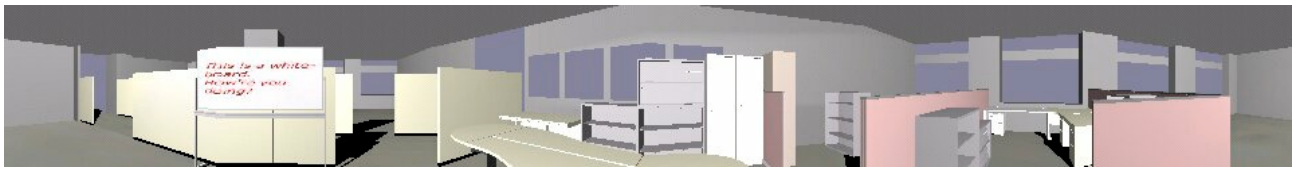

Fig 3. Simulated view from cylindrical display setting

4 3D Engine

Nowadays, there are several types of virtual reality platforms, consisting of different kinds of displays, computers and operating systems. To make an application, especially a shared application, really usable, it must be easily portable to several different platforms instead of having to engineer it separately for each platform. This kind of portability requires a special graphics engine that can be compiled and configured to almost any kind of system without too much work or extra investments for 3D simulation software such as IRIS Performer [2] or Sense8 WorldToolKit [3].

The main application of the system under development is the 3D Engine. It is the part of the system that handles

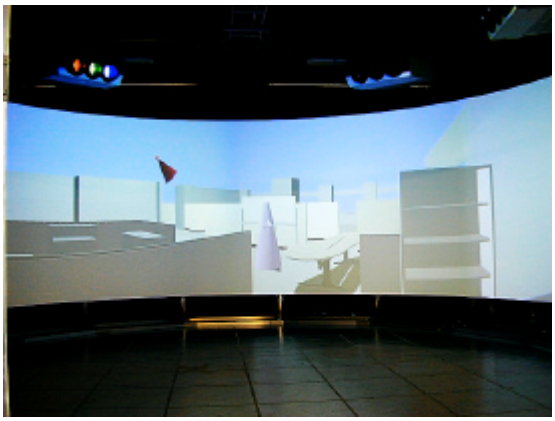

Fig 4. Photo of a scene in CYLINDRA

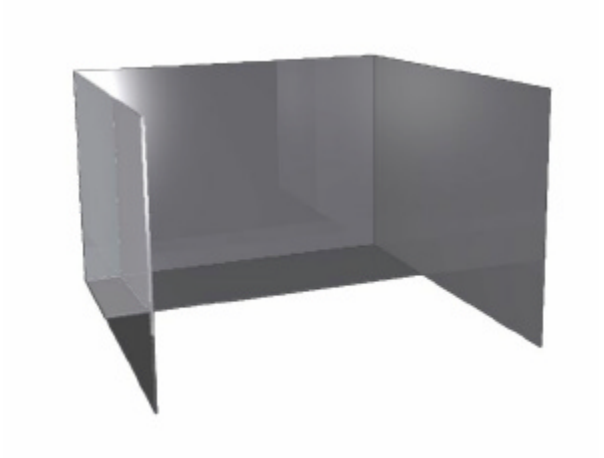

Fig 5. Immersive multi-display system display setup

Fig 6. Simulated view from immersive multi-display

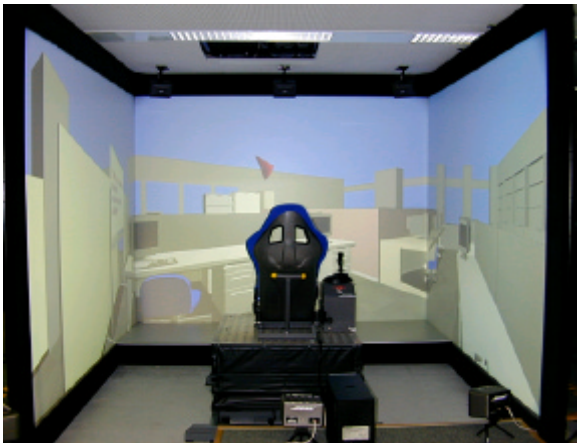

Fig 7. Photo of a scene in immersive multi-display system

multiple CPUs or graphics pipes. This makes GLUT unsuitable for certain systems. CYLINDRA, which is used in the test system arrangement, is one such system. As GLUT would be able to utilize only one of the eight available CPUs, the overall performance is poor. Also, the InfiniteReality2 graphics sybsystem has two graphics pipes, of which GLUT could utilize only one. In CYLINDRA platform this means that only 3 projectors could be used and only half of the display space covered. Due to these restricitions, native window handling code, namely GLX, was added to be used in such X Window Systems GLUT is not suitable for.

Depending on hardware and display configuration, different display handling methods have been implemented into the engine. In a system with a large virtual desktop, which is mapped into separate monitors or screens, one continuous window is created and divided to sections to fit the monitors/screens. Each section in the window can be adjusted as a whole through the configuration files, but the sections cannot be tuned individually. This method is usable with GLUT. Snapshots in figures 3 and 6 have been taken using this method.

In case of separate computers handling the drawing of each screen, the drawing is divided into main clients and help clients. Using this method, the main client handles all the actual work, including handling user input and communicating with the network server to handle scene sharing, and sends screen update commands using either UDP or TCP to the help clients, which only do drawing and nothing more. Whether to use UDP or TCP can be changed through the configuration files. In a normal case UDP provides better performance. Each screen, drawn by a separate client, can be adjusted individually using the configuration files. This method is the most flexible of all implemented methods and is usable with GLUT. Photo in figure 7 is from a setup using this method.

The third method is using GLX and is usable only in X Windows systems as such. It is meant to be used only in systems for which GLUT is unsuitable for, namely systems with more than one graphics pipe and/or multiple CPUs. In this method an X client is created for each separate screen and they can be adjusted as a whole, not individually. Currently the engine supports maximum of 3 graphics pipes and 9 window, but will be upscaled later. Photo in figure 4 is from a setup using this method.

Figure 8 depicts the test arrangement in terms of the display handling method used.

The 3D Engine has been succesfully tested in a number of different configurations, including IRIX 6.5, Linux, Windows 95, 98, 2000 and NT 4 in computers ranging from SGI O2 through several different laptop and desktop PCs to SGI Onyx2. Performance of the engine is, as expected, quite poor in non-3D-accelerated systems and ranging from adequate to very good in a properly 3D-accelerated up-to-date system. Some frame rates in different configurations can be seen in table 1.

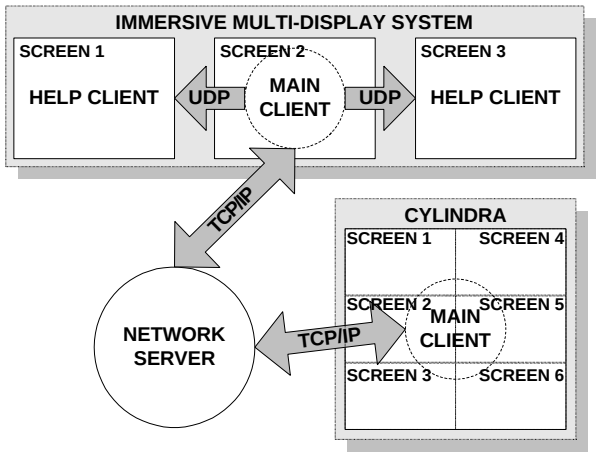

Fig 8. Heterogeneous system arrangement

Table 1. Some performance values

MONITOR - ONE WINDOW	fps : avg (peak)		
640 x 480, no textures, no shadows	**light load (1000 polyg.)**	**medium load (10000 p.)**	**heavy load (50000 p.)**
366MHz, Windows 2000, no 3D acc.	15.8 (20.4)	13.1 (20.4)	8.8 (13.0)
800MHz, Windows 2000, 3D acc.	105.6 (111.1)	86.6 (111.1)	81.7 (111.1)
180MHz MIPS R5000 O2, IRIX 6.5, 3D acc.	18.2 (25.8)	13.9 (25.1)	3.4 (5.1)
640 x 480, textures, shadows	**light load (1000 polyg.)**	**medium load (10000 p.)**	**heavy load (50000 p.)**
366MHz, Windows 2000, no 3D acc.	11.2 (14.5)	8.1 (13.1)	6.8 (10.9)
800MHz, Windows 2000, 3D acc.	80.6 (111.1)	64.5 (111.1)	30.4 (50.8)
180MHz MIPS R5000 O2, IRIX 6.5, 3D acc.	10.2 (12.8)	7.4 (12.8)	2.9 (4.1)
IMMERSIVE DISPLAY - MULTIPLE WINDOWS	**fps : avg (peak)**		
CYLINDRA (6 windows, stereo)	**light load (1000 polyg.)**	**medium load (10000 p.)**	**heavy load (50000 p.)**
1024 x 768, no textures, no shadows	93.2 (166.7)	91.7 (166.7)	11.9 (18.5)
1024 x 768, textures, shadows	36.9 (55.6)	25.4 (47.6)	3.5 (5.8)
Immersive Multi-Display System	**light load (1000 polyg.)**	**medium load (10000 p.)**	**heavy load (50000 p.)**
640 x 480, no textures, no shadows	68.2 (90.9)	51.1 (90.9)	26.9 (45.5)
640 x 480, textures, shadows	54.5 (90.9)	46.7 (90.9)	6.2 (10.8)

For single-window monitor tests used parameters are as follows : view angle 90°, field of vision 73.8° and medium draw distance. As can clearly be seen in the table, the performance is good as long as the complexity of the scene remains tolerable. The system as such, while usable, is not very user-friendly in very complex design situations, such as designing a large office with hundreds of complex desks and chairs.

5 Network Protocol

5.1 Overview

In sharing a virtual space through a network, coherency control, which means keeping consistency of the virtual space between multiple remote locations, is one of the most important subjects. Many different methods for coherency control have been proposed over time [10, 11, 12, 13, 14]. The methods can be categorized in two main types : methods using exclusion control and methods not using exclusion control.

Protocols utilizing exclusion control allow only one user to access the virtual environment at any one time. As locking and unlocking, before accessing the virtual environment and after the operation is finished, require a little time, exclusion control causes delays in system operation. Also, it is not a very user-friendly solution, as only one user can operate each locked object at a time, forcing others to wait. Protocols not using exclusion control result in a tag-of-war -situation when several users are accessing the same object at the same time [10].

In a trial-and-error situation like interior design, restrictions in both aforementioned types of concurrency control pose a problem. To address this problem, a protocol that realizes simultaneous and restriction-free access for multiple users has been developed. The base idea of the proposed protocol is not to prevent a conflict before it happens, but to resolve it afterwards by user's discussion. If separate users' operations for an object causes a conflict, the conflicting object is duplicated to tell the users there is a conflict to be resolved.

For example, a designer and a customer move the same piece of furniture at the same time. This causes the mentioned piece of furniture to be duplicated, indicating there has been a conflict. The designer and the customer can then discuss which choice is better, and either delete one or both, or leave them both as they are. Using this mechanism, user's operations are realized immediately locally, dependless of possible delays in the network.

5.2 Managing Virtual Environment

In the proposed protocol, virtual environment is expressed by a tree of objects. All objects, which will be called nodes in this context, in the tree have specific identifying information. The information in each node includes node identification number for identifying the node in the tree, data of the appearance and placement of the node, such as object name, position and orientation, and the information needed for coherency control, such as version number based on the value of the logical clock and name of the last modifier. See figure 9.

Node identification number is used by the 3D Engine for indicating the object being modified by a user. It is a unique number and is given to the node in creation. Position and orientation data are given as offset values from parent object's position or orientation. Version

number and last modifier of the node are updated when the node is modified by a local or remote user. Updating the version number and the last modifier is done not only to the modified node, but also all it's children and the node on the path from modified node to the root node. See figure 10.

Object name works as a link to the geometry data of the object. In the proposed protocol, all geometry data is stored in an object database and can be accessed using the object name.

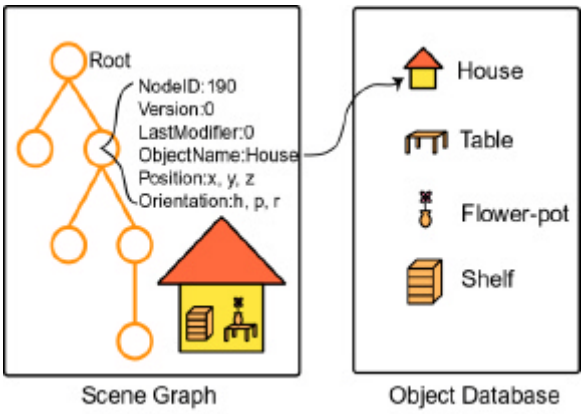

Fig 9. Expressing the virtual environment

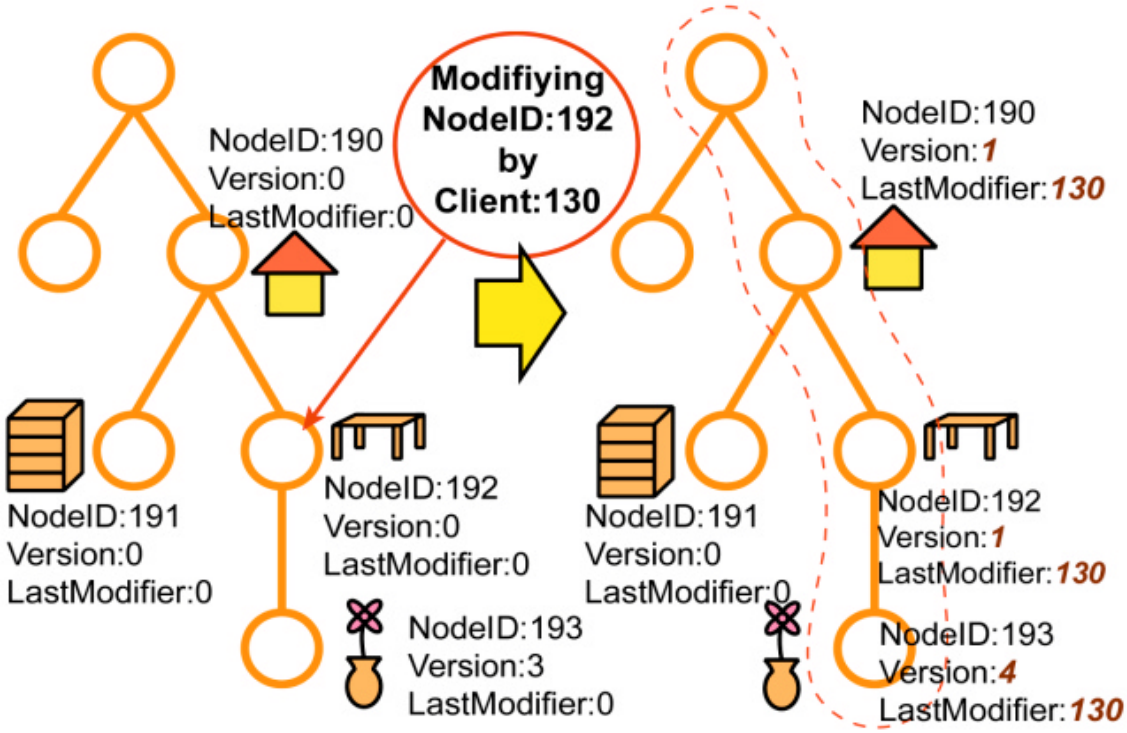

Fig 10. Updating version number and last modifier

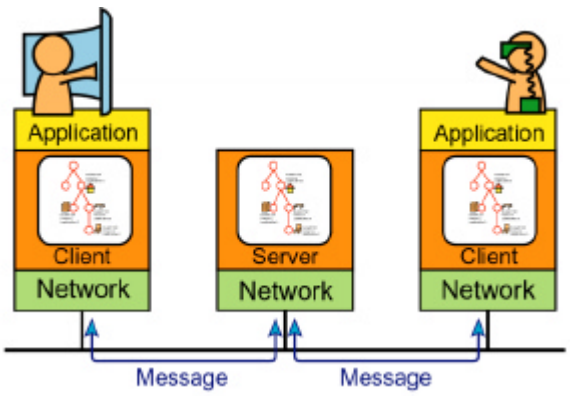

Fig 11. Network model

5.3 Network model

The protocol is based on client-server model, using one or more clients to provide interface for users and one server to manage the virtual environment, including coherency control. All clients and the server have information of the current virtual environment as a tree structure. TCP/IP is used for data transfer between clients and the server. See figure 11.

Clients provide interface through the 3D Engine for creating, moving, rotating, and deleting objects. When user modifies an object, the client updates current tree strcuture and sends the information concerning the modification as a message to the server. The message includes the type of operation, identification number of the modified node, modifying parameters, current version number and current last modifier.

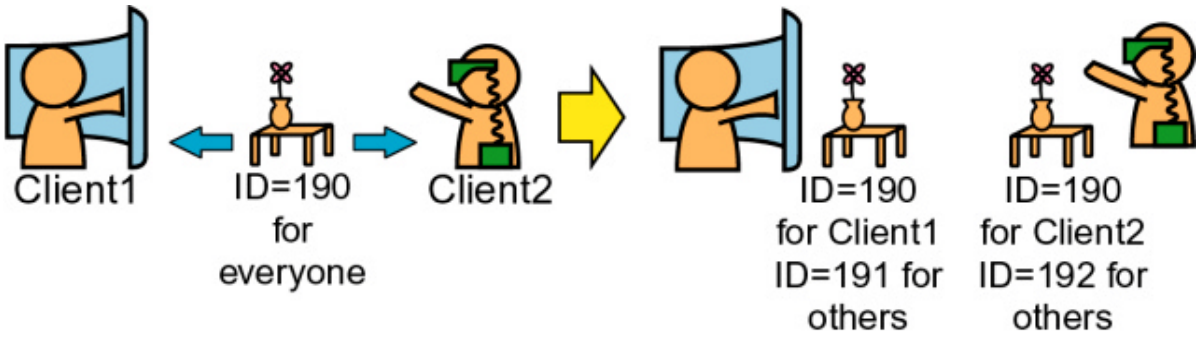

Fig 12. Node identification number translation

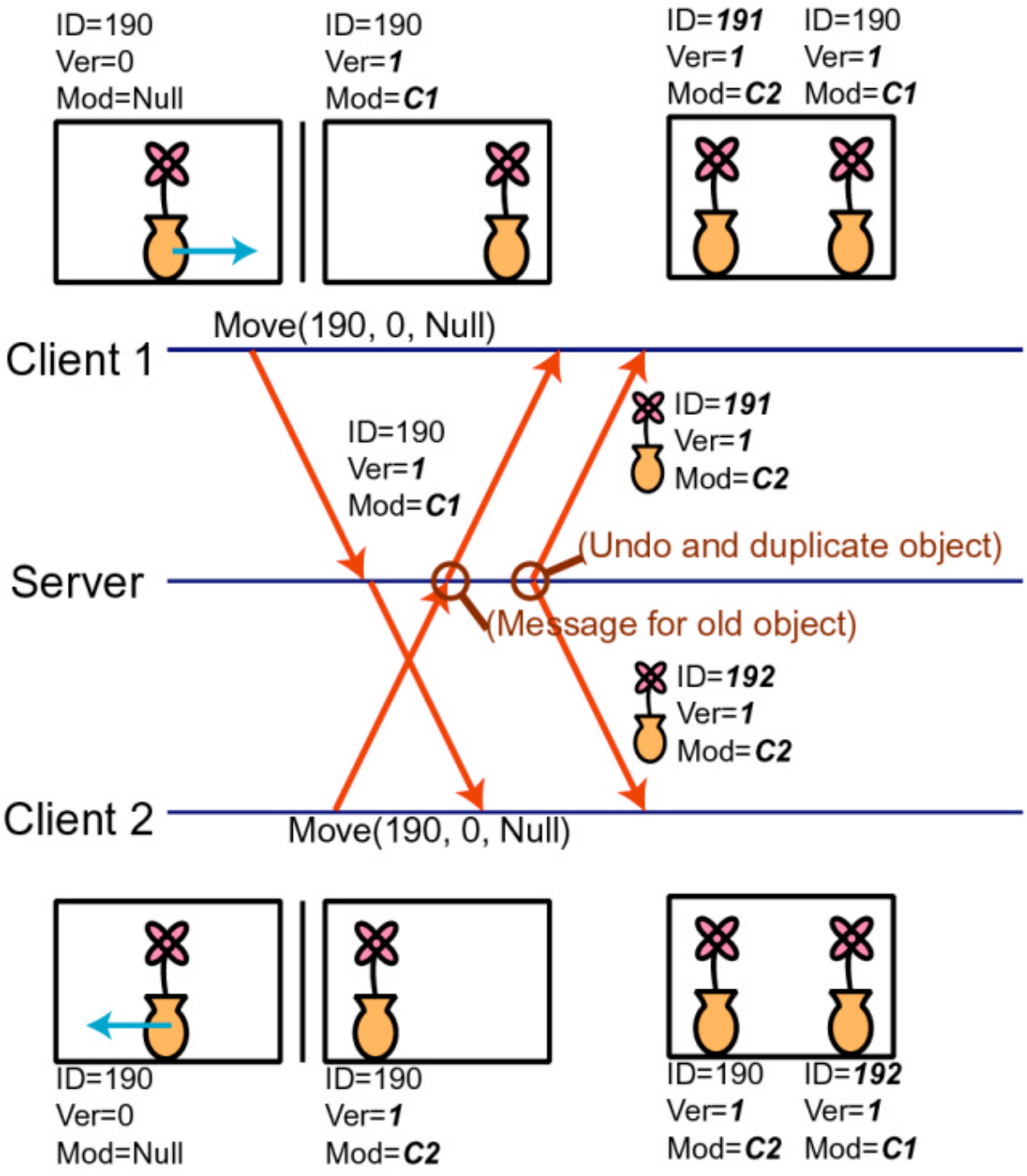

Fig 13. Conflict detection

Server manages connections from clients, data distribution to all clients, and coherency control. When the server receives a message from a client, it does the modifications specified in the message on its own tree structure before sending the message to other clients. In case of node duplicatioon, the node identification number is sometimes different between server and client, as the duplicated node gets a new identification number. See figure 12. For keeping track of different numbers, server has a node identification number translation table for each client, and it is used to translate the identification number whenever a message is sent or received. The translation table is updated when an object is duplicated. When the client receives a message from other clients via the server, it updates its own node.

5.4 Coherency Control

For detecting a conflict, all messages exchanged between a client and the server have a version number and information of last modifier. Information of last modifier is the name of the client who last modified the node. Upon receiving a message the version number and the last modifier in the message are compared with the version number and the last modifier of the corresponding node in the local tree structure. If the values in the message are different from corresponding node's values in the local tree, the node is known to have been modified between the time of creation and the time of having received the message, hence implicating a conflict. See figure 13.

When a client detects a conflict, it ignores the message and destroys it. When the server detects a conflict, it performs node duplication and starts undoing the entire tree until the required node with correct version number and last modifier is found. By this method the existance of the required node is proved. After undoing and locating the required node, the node is duplicated. Nodes to be duplicated are decided in the same way as updating version number and last modifier, as can be seen in figure 10. Duplicating multiple nodes is required for handling nodes having parent-children relation.When a node is duplicated, the server updates the node identification number translation table. Translating node identification number is done so that each client considers the locally modified object primary. After duplicating nodes, the server executes the modification required in a message. Then it redoes the whole tree excluding the duplicated nodes. At this point, the tree has two sets of nodes, one for each user's requirements. Last, the server sends duplicated nodes to the clients as messages. The clients receiving this message add the duplicated nodes into their own tree structures.

5.5 Advantage of the protocol

The proposed protocol provides a new kind of design procedure. The designer and the client can move the furniture around with no limitations from the protocol. After a conflict happens, the designer and the client can discuss which solution is better and modify the scene accordingly. The new procedure can reduce the number of required discussions compared to a case where traditional coherency control method is used. Discussion, in this context, is thought as a sequence of voice communications during design process.

Using a traditional coherency control method only one solution in a conflict situation can be displayed at a time. If either party wants to see the result after both possible modifications, two separate discussions are needed, one after each modification. Using the proposed protocol, both parties can make their own modification at the same time and only one discussion is needed in the end of the modifications.

6 Conclusion

To realize useful interior design system between heterogeneous virtual reality platforms, a portable 3D Engine and a network protocol are presented. The 3D Engine realizes the platform-independency among different platforms, such as a CYLINDRA and a PC-based immersive multi-display system. The network protocol realizes sharing a virtual space without locking while maintaining coherency, making the communication uninterrupted and smooth.

Possible future step in developing the system is changing one end of the system to a portable see-through head-mounted display system. This portable system allows client to stay at his home and see the design using augmented reality, adding virtual furniture into the real space making it even easier to imagine the final result.

References

1. http://www.solidray.co.jp
2. http://www.sgi.com/software/performer/
3. http://www.sense8.com/products/wtk.html
4. http://www.sgi.com/software/opengl/
5. Woo M., Neider J., Davis T., Shreiner D., "OpenGL Programming Guide, 3rd Edition" (1999)
6. http://reality.sgi.com/mjk_asd/glut3/glut3.html
7. Kilgard M., "The OpenGL Utility Toolkit (GLUT) Programming Interface, API Version 3", (1996)
8. http://www.sgi.com/software/opensource/glx/

9. Woo M., Neider J., Davis T., Shreiner D., “OpenGL Programming Guide, 3rd Edition”, pp.632-633 (1999)
10. Leigh, J. et al., “CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environment”, Journal of Virtual Reality Research, Development and Applications, pp.217-237 (1997)
11. Singhal, S., Michael, Z., “Networked Virtual Environments”, Addison Wesley (1999)
12. Lea, R. et al., “Scaling a shared virtual environment”, http://www.csl.sony.co.jp/person/rodger/ICDCS/icdcs2.html (1996)
13. Katayama, A. et al., “Collaborative CyberMirage: A Shared Virtual Environment with High Photoreality and Mutual Awareness”, Transaction of IPSJ, pp.1484-1492 (1998)
14. Broll, W., “Distributed Virtual Reality for Everyone”, A Framework for Networked VR on the Internet”, http://fit.gmd.de/pages/VRAIS.pdf (1997)