A peer-to-peer collaborative 3D virtual environment for visualization

Yi Pan and Francis T. Marchese*

Computer Science Department, Pace University, 163 William Street, 2nd Floor, NY, NY 10038

ABSTRACT

A peer-to-peer collaborative visualization system has been built that can support both traditional displays and 3D virtual reality hardware. The software is built around Sun's Java3D graphics and JXTA peer-to-peer networking APIs, allowing two users to load VRML geometry files and manipulate their contents. Although this software takes advantage of VR hardware, it may be used between any two Java supporting peers. Finally, because no dedicated server is required, collaborative visualizations across the web become easier to initiate and more spontaneous.

Keywords: collaborative visualization, peer-to-peer networking, virtual reality, Java3D, JXTA.

1. INTRODUCTION

The Internet has made it possible for individuals or groups to collaborate at a distance. Collaborative virtual environments (CVEs) have been developed to support a wide range of collaborative activities including gaming, military training, scientific visualization, and engineering design.[1] Collaborative visualization software systems include VisAD,[2] COVISA,[3] Sieve,[4] and others.[5-7] In general, these programs have been developed to provide a generic set of collaborative tools for data sharing, representation, and visualization. However, the usability of such systems may suffer for two reasons. First, many CVEs require special purpose hardware and software. For example, visualizations in CAVEs,[8] on Immersadesks[9] or Responsive Workbenches[10] exploit expensive dedicated hardware that must be shared and scheduled for use. As a result, casual, impromptu collaborations, of short duration, that convene at odd hours, or employ an amalgam of mainstream conferencing and problem-specific software products may be either barred from these systems or made inconvenient to use. Second, most CVEs are client-server applications[2-4] that subject collaboration to constraints. Since a server is responsible for data transmission, it must have a resolvable IP address for clients to make a connection. Yet, the Internet's growth has left fewer machines with static IP addresses, leaving many powerful computing systems with dynamic IP addresses that may inhibit their abilities to be used as servers. And client-server applications cannot traverse firewalls.

We have recently investigated the design and use of dedicated hardware for visualization[11] and the implementation of software for anytime-anywhere collaboration.[12] The former research focused on building an inexpensive stereographic table that could run mainstream visualization software rendering side-by-side stereo pairs. It proved successful at delivering stereoscopic displays of visualizations to groups of two or three users. The latter research transformed an open-source 3D graphic visualization program written in Java into a collaborative system utilizing Sun's JXTA peer-to-peer API.[13,14] Sun's peer-to-peer protocol allows any network-connected device to communicate and collaborate. Peer-to-peer (P2P) networking solves client-server problems by interconnecting machines so that each node acts as both server and client. And P2P application users can communicate through firewalls. Our research demonstrated that a single-user program could be quickly converted into a functional and useful P2P collaborative application within a very short time, resulting in web-based collaboration that is easier to initiate, more spontaneous, and supported by a wide range of visualization software.

Encouraged by the success of these explorations we decided to build a web-based peer-to-peer collaborative visualization system that would support a recently constructed, generic, desk-side 3D virtual reality system. The visualization software would be built around Sun's Java3D and JXTA APIs, allowing two users to load VRML geometry files and manipulate their contents. Java3D[15] was selected because it can perform stereographic 3D rendering, provides

* Send correspondence to F.T.M.: E-mail: fmarchese@pace.edu.

for interaction, and contains interfaces to mainstream VR devices. Although our application takes advantage of VR hardware, it may be used between any two Java supporting peers. An account of the system is found herein.

The following section presents the requisite background to JXTA and Java3D. Section 3 contains the system design. Section 4 presents a sample session for running the peer-to-peer collaboration. Discussion and conclusions are found in Section 5.

2. BACKGROUND

2.1. JXTA –Peer-to-peer Networking

JXTA technology is designed to enable interconnected peers to easily locate each other, communicate, participate in community-based activities, and offer services across different P2P systems and communities.[13,14] Platform, services, and applications are the three layers of the JXTA architecture. Platform is the core JXTA layer containing the elements of every P2P solution. Services provide access to JXTA protocols. And applications use services to access the JXTA network and utilities. Unlike other peer-to-peer systems, such as Napster, Gnutella, and AIM, which are built for delivering a single type of service, JXTA is transport independent. It may be implemented atop TCP/IP, HTTP, Bluetooth, HomePNA, and other protocols. This interoperability ensures that systems built on top of JXTA can communicate with each other, as long as there is a correct transport protocol handler between them. For instance, in JXTA, a peer chooses from different protocols to fit its needs. Peers communicate across a firewall using the HTTP protocol, while TCP is chosen for communication across a LAN.

JXTA is founded on the following basic concepts: peer, peer group, endpoint, pipe, network transport, advertisements, protocols, and discoveries. A peer is a virtual communication point. There are three basic types of peers: simple peer, rendezvous peer, and router peer. A simple peer has the least functionality, and is used behind a firewall. A rendezvous peer processes queries from other peers, and is used when content transmission is required. A router peer enables peers to communicate with other peers separated by a firewall. A peer group organizes peers, and provides specific services to group members. Data may be shared within the group's scope, and peers may check another peer's status before it is allowed to join the group, subject to security requirements. Network transport manages data transmission over the network. JXTA allows peers to choose different protocols to fit specific needs. HTTP protocols are used for peers to communicate through firewalls, while TCP is chosen for intranets. A network transport system is composed of endpoints, pipes, and messages. An endpoint refers to an address of a peer. A pipe is a unidirectional, asynchronous, and virtual connection of two or more endpoints. Messages contain the data being transmitted. Transmitted data is packed into a message, which is then sent over the output pipe. At the other end of the pipe, a peer receives the message from its input pipe and extracts the transmitted data. An advertisement is a structured representation of an entity, service, or resource made available by a peer or peer group as a part of a P2P network. It is an XML document in JXTA, containing the description of a message, peer, peer group, or service. Peers discover advertisements on the network to find other peers. They can use a cached advertisement, rendezvous peer, or router peer to discover each other within a LAN or through a firewall. JXTA has a number of protocols for advertising, sending, routing, propagating, and securing messages. These protocols are used to join a peer group, find another peer, create pipes between peers, and propagate messages among peers. In sum, they are used to effect peer-to-peer collaboration.

JXTA is cross-platform. Its stable versions run on Windows95, 98, 2000, ME, and XP, as well as on Solaris and Linux, with appropriate Java Runtime Environment support.

2.2. Java3D

The Java3D API was established by Sun for interactive 3D applications development. It provides high-level structure for creating and manipulating 3D geometric objects. Java3D has been used for applications in many fields, including e-commerce, data visualization, collaborative work, and entertainment.

Based on Java, Java3D is an interface of Java classes supporting a platform-independent 3D graphics development mechanism. Built on low-level graphics APIs, such as OpenGL and DirectX, Java3D takes advantage of hardware acceleration to perform native rendering, while supporting high-level scene representations utilizing a scenegraph model. In addition, Java3D includes a view model that eases the transition from a screen-centric rendering model to a projected model. Finally, Java3D's support for stereoscopic viewing provides for greater visual realism.

A Java3D program creates instances of Java3D objects and places them into a scenegraph data structure.[15] The scenegraph is a tree structured arrangement of 3D objects that completely specifies the content of a virtual universe and how it is to be rendered. A scenegraph is a directed acyclic graph, containing no looping paths with nodes connected in parent-child relationships. SceneGraphObject is the base class for objects in a scenegraph. Its child, Node, is the basic component of the scenegraph. There are four kinds of nodes. A shape node represents 3D object geometry. An environment node provides light sources and background. A group node organizes the scenegraph elements. And a ViewPlatform node places 3D objects so a viewer may see them.

Java3D's tree structure contains a root node called VirtualUniverse that may possess one or more locales composed of scene or view branches. A scene branch represents the scene content, including 3D geometric objects and all environmental nodes such as light, background, and texture. A view branch has a view platform whose location decides how far in front of a user 3D objects are to be positioned. A ViewPlatform creates views by setting a position, orientation, and angle from which an object may be viewed. Multiple views allow users to look at the same object from different viewpoints.

Java3D can either create 3D objects utilizing its own built-in library or load other 3D modeling files, translate them into its own scenegraph, and manipulate them. In contrast, 3D modeling files, such as VRML, are read-only files, have no knowledge of other types of 3D files, and can only be read by a VRML browser.

3. COLLABORATIVE SYSTEM DESIGN

3.1. Hardware

The immersive desk is a generic virtual reality system intended to run any stereographic or VR application or library. It was designed and built so that it should be compact enough to fit within a small laboratory or large office; support multiple viewers; accommodate all computer and display components; and be easily moved, disassembled, reassembled, and stored. Figure 1 displays the system. It is 72" (H) x 72" (W) x 36" (D), and shallow enough to be placed adjacent to a work surface such as an office desk. The framework is on wheels, allowing it to be rolled from office to laboratory.

The generic immersive desk's design has been influenced by three notable stereographic display systems: EVL's Immersadesk,[9] Kreuger and Froelich's Responsive Workbench,[10] and UNC's Nanomanipulator.[16] They are characterized as being self-contained units (excluding computer), having either horizontal or angled rear projection screens employing a stereographic projector with a high enough refresh rate (typically 120Hz) to support active stereo shutter glasses. However, the prohibitive expense of these systems has engendered design of systems based on passive stereo projection, running on PCs, typically using Linux as the operating system. For example, Jones, Parker, and Kim have built a Linux PC with dual-head display for passive stereo projection onto a silver lenticular screen.[17] AGAVE (Access Grid Augmented Virtual Environment) is a rear projection passive stereo system for large groups. It runs on Linux-PCs employing the CAVERNsoft library.[18] In contrast, our previous stereographic table[11] was notably Microsoft Windows-based and ran any application with superimposed side-by-side stereo pairs.

The lessons learned from this combined research found its way into the desk's design as the following requirements: an immersive desk should be reasonably compact and moveable, support single or multiple viewers, and run nearly all stereographic display and virtual reality applications without extensive reconfiguration. Our desk design meets all these requirements. As a result, any stereo-enabled software that runs under any operating system will run on the desk.

There are seven components to the immersive desk: framework with mirrors, rear projection screen, projectors with polarization filters, video card, pointing devices, speakers, and computer. The framework is constructed from 1.5" square rigid extruded aluminum framing manufactured by 8020 Inc. Two mirrors are attached to the frame to guide the projected images to the rear projection screen. The rear projection screen is a "Disney" Black polarize-preserving screen with a "Snapper" mount from Stewart Filmscreen Corporation that snaps onto a 39" x 51"x 1.5" sq., black aluminum frame to create a 36"x 48" display area. Two NEC VT540 1000 lumen LCD projectors were selected for the system that have native XGA resolution (1024 x 768) and offer extensive control over the projected image including: positive and negative keystone correction; front, rear, ceiling, and table mounting; and independent adjustment of rgb color channels. The two projectors are stacked to superimpose left and right images using a Chief adjustable 35mm slide projector stacker. Standard, three-inch square linear polarizing filters were installed in front of each projector lens, set orthogonally at $\pm$ 45°. The video card is a 3DLabs Wildcat III 6110 with 208 MB of memory and stereographic sync. It is attached to a Cyviz Xpo.2 demultiplexer that converts an active 120Hz stereo signal from the video card into left and

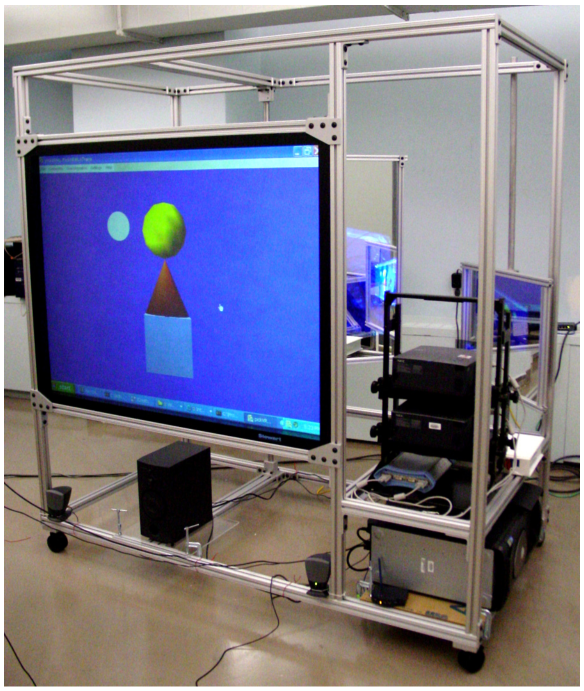


Figure 1. Generic virtual reality desk – a self-contained movable unit with computer, stereographic projectors, rear projection screen, and sound.

right eye 60Hz signals, one for each projector. Two pointing devices used: a GyroRemote by Gyration and a 6DOF Spaceball. The GyroRemote is an in-air mouse that detects hand motion and coverts it to cursor movement, communicating over radio frequencies with a base station attached to a PC's USB port. The Spaceball can specify positions and orientations in three-space. Harmon Kardon speakers are attached to the computer's sound card. The immersive desk is connected to a Dell Precision 450 Workstation with dual 1.8GHz Intel Xeon processors, 1GB of memory, 80GB hard disk, and 1Ghz network card, running Microsoft Windows XP Professional.

3.2. Software

A lightweight collaborative visualization system (LCV) was designed as a visualization tool for rendering, review, and manipulation of data stored as VRML2 models. It was built to work on standard displays as well as with the immersive desk. LCV allows any scene object to be moved in any of the three Cartesian directions or rotated about its local reference frame under mouse control by clicking with the requisite mouse button. Dropdown menus provide additional control for each collaborating peer including: turning stereographic display on and off and adjusting stereo separation; selecting type, color, and extent of the light source; adjusting camera controls; and changing transformation parameters. Finally, as observed by Singhal and Zyda,[1] since online chat is a powerful tool for exchanging ideas among peers in virtual worlds, a chat window has been added to LCV.

LCV was constructed to meet the design objectives for collaborative visualization systems put forward by Wood and coworkers.[3] Such systems should allow users to share program control, collaborate dynamically, use the software in an instruction scenario, learn it easily, and exchange data readily. In this collaborative system, either participant may control any of the rendering parameters or move any part of the visualization model at any time, thus supporting variability in visualization scenarios. Either participant may work as an instructor, because each peer may take control of the analysis at any time, and every action in a sequence is replicated on the collaborating peer. The system is easy to learn because the visualization and interface components are consistent with standard GUI widgets. And the networking components are accessed through GUI menus as well. Finally, all data is exchanged between peers, meaning both peers are synchronized in identical collaboration states.

A number of design decisions were made to minimize implementation details and complexity. First, it was decided to create a two person collaboratory founded on a simple telephone conversation model of communication. It was assumed

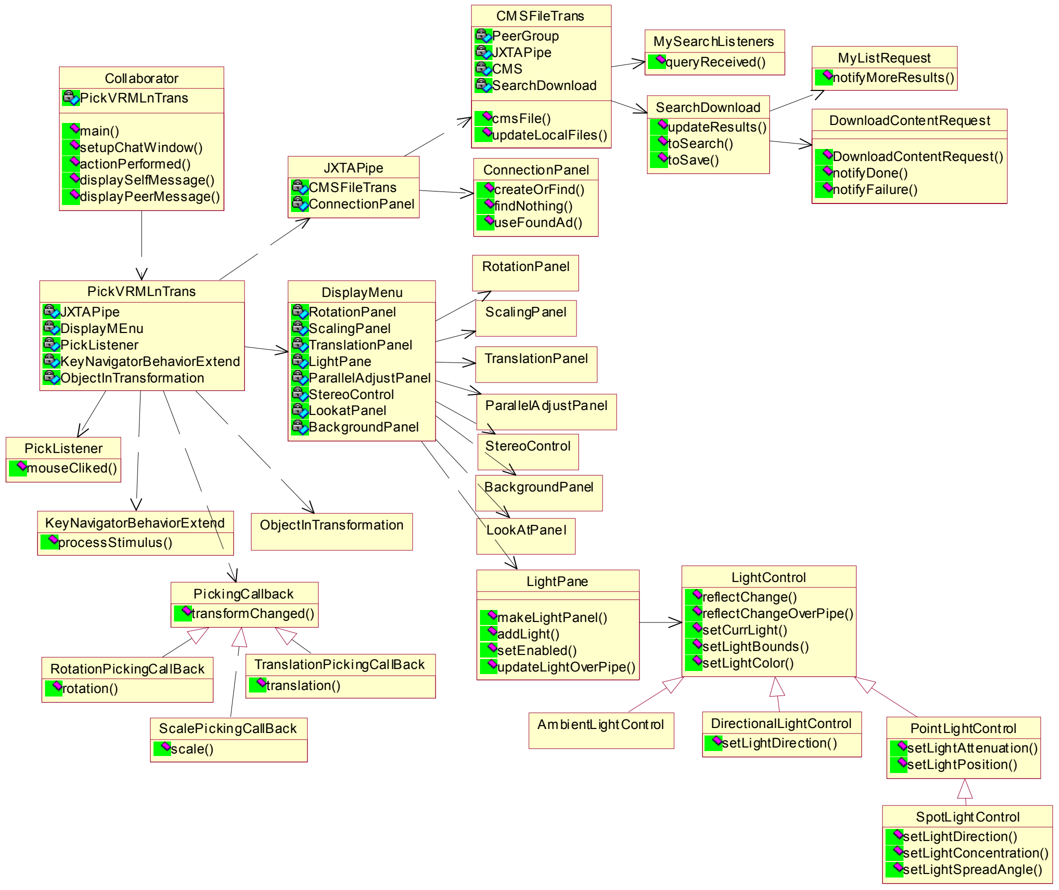


Figure 2 Lightweight collaborative visualizer class structure.

that the software would be used in a casual way by two collaborators who wished to visualize and manipulate a dataset as part of a web-based conversation. This would give either user unconstrained access to the shared state of the system. As a consequence, it would be possible to simultaneously access the shared state, possibly causing data corruption. However, such scenarios may be mitigated if participants' actions are coordinated as part of conversation and exploration. Precedence for such a design decision is found in the free access model of cAVS, in which any participant may affect the common collaborative state at any time.[19] Second, each collaborator receives a synchronized view of the system's state. As a result, any event that occurs on one peer is immediately communicated to, and executed on the second. Finally, LCV was designed as a single-user program with open provisions for a collaborating peer. The rationale comes from our experience using JXTA to adapt a single-user scientific visualization program for P2P collaboration.[12] Because visualization programs are GUI-intensive, it was found that embedding JXTA controls within the GUI allowed messages to be sent to synchronize a peer. Therefore, LCV was designed initially for an individual user, fitting it for collaboration during the course of development.

LCV has three major parts: GUI, network communication, and 3D graphics processing. The complete class structure is shown in Figure 2.

LCV's GUI relies on Java Swing and AWT to provide event-driven mechanisms for triggering interaction and collaboration. Each interface component is linked to a listener interface that captures a unique event. Thus, mouse movement and menu selection initiate events that are captured and acted upon by event handlers. This is where the

JXTA code is embedded. Thus, when a GUI event occurs, the controlling peer transmits a keyword designating the GUI operation along with its numerical operands through the input end of the JXTA pipe, and the receiving peer extracts the message at the pipe's output end. The receiving peer then immediately updates its system state to match that of the sending peer.

LCV's network system employs the JXTA Content Manager Service (CMS)[13] for file transfer and JXTA to transmit all other information about the 3D scene and its objects, such as transformation, lights, and background changes. CMS acts as a framework for sharing/exchanging content among JXTA peers. The service provides the ability for a peer to host content that it shares with other peers and locate and retrieve content from others. When another peer requests a search of local content or for a copy of shared content, the Content Manager will automatically handle the request.

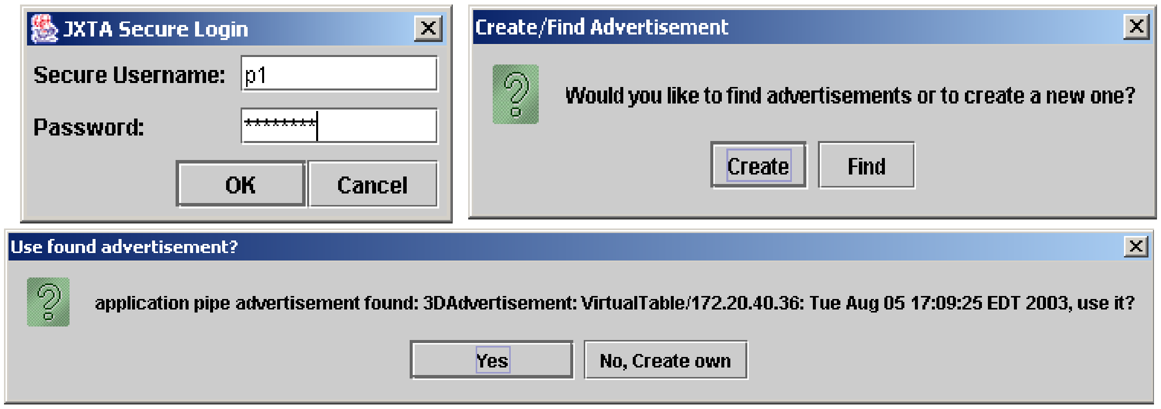
The 3D graphics component of LCV loads a VRML2 file and converts it into a Java3D scenegraph using the XJ3D VRML file loader.[20] It should be noted that although VRML's scenegraph structure is similar to Java3D, it is read-only. In addition, the degree of interaction a VRML file provides is limited to interaction with the whole scene. As a result, it is necessary to sift through all the objects in the VRML scene to construct a Java3D scenegraph in which each shape node has the ability to be manipulated. During the reconstruction, some details must be resolved, such as unusable parent-child relationships in automatically reconstructed scenes, potential multiple parents, shared groups, and so on. Besides scene content, background, lighting, and other environmental nodes must be defined. A background node is a leaf node in a Java3D scenegraph that contains a bounding sphere specifying the working boundaries of the scene. LCV sets an initial value for color and bounding sphere, and provides users with a pull-down menu to change these parameters during a session.

Java3D supports ambient, directional, point, and spotlights. Light source color, its influencing bounding sphere, and on/off status may be set within LCV. Point lights and spotlights may be moved throughout the scene. Other than changing a light's position by mouse, the software provides a light control panel with all lights as tabbed panes. A class LightPane acts as the messenger between the main program and the panel. It processes data from the main program and from the collaborating peer, and passes them to light control. Based on the hierarchy of lights in Java3D, the software adds interactive functions to each light forming four light controls. A user can change lights by clicking the corresponding light control pane, typing in modifications to color, influencing bounds, direction, or other factors.

A user may transform objects individually or collectively within LCV. The keyboard controls review of the entire scene by simulating a VRML file browser's "pan" and "study" behavior. Individual objects are manipulated in two ways. First, an object is picked by clicking and dragging the mouse to execute geometric transformations of translation, scaling, or rotation. Second, pull-down menus are used after picking, allowing the specification of detailed transformation information by typing in the desired input.

4. SAMPLE SESSION

LCV is implemented as a Java application and is loaded either from the command line or batch file. A typical scenario for P2P collaboration using LCV begins as follows: start JXTA, create and publish pipe advertisements, locate and use the created pipe advertisement, and send outgoing and listen for incoming messages. LCV initiates collaboration by joining the Net peer group. The default Net group gives access to the basic JXTA services. When the JXTA platform is started, it searches for certain configuration files that record network settings specific to the peer. If this file is not present, a JXTA Configuration screen is shown. It is here that a username, password, and advanced network specifications can be configured. Once logged in, a message from JXTA requests the user to either create an advertisement for a new peer session, or find an advertisement for an existing peer session (Figure 3). The former is selected to originate peer collaboration; otherwise the peer will attempt to find the originating peer on the network. If the user has selected advertisement, the GUI will load and advertise to the netgroup; otherwise it will search the network looking for a peer connection. When an advertising peer has been located, the searching peer has the option to make a connection. After the connection is made, the GUI will be displayed. At this point one peer will load a VRML2 model through the "File" drop-down menu to begin collaboration. The file name is propagated through the controlling peer's out pipe and added to its shared content list in the form of XML. The other peer actively searches for the propagated file's content. Once the file is located, the peer downloads and converts it to form the same 3D scenegraph on its own site, and then draws the model (Figure 4). Either collaborator can transform the scene, move individual objects, or adjust rendering parameters. For example, when one collaborator selects the traditional VRML-viewer behaviors like "pan" and "study," both users see exactly the same view, smoothly translating or rotating in three dimensions.


JXTA Secure Login

Secure Username: p1

Password: *********

Create/Find Advertisement

Would you like to find advertisements or to create a new one?

Use found advertisement?

application pipe advertisement found: 3DAdvertisement: VirtualTable/172.20.40.36: Tue Aug 05 17:09:25 EDT 2003, use it?

Figure 3. Creating a network connection.

Any scene object may be moved in any of the three Cartesian directions or rotated about its local reference frame with the mouse. Dropdown menus provide additional control for each collaborating peer including: turning stereo display on and off and adjusting stereo separation; selecting type, color, and extent of the light source; adjusting camera controls; and changing transformation parameters. Each of these selections produces a pop-up dialog box.

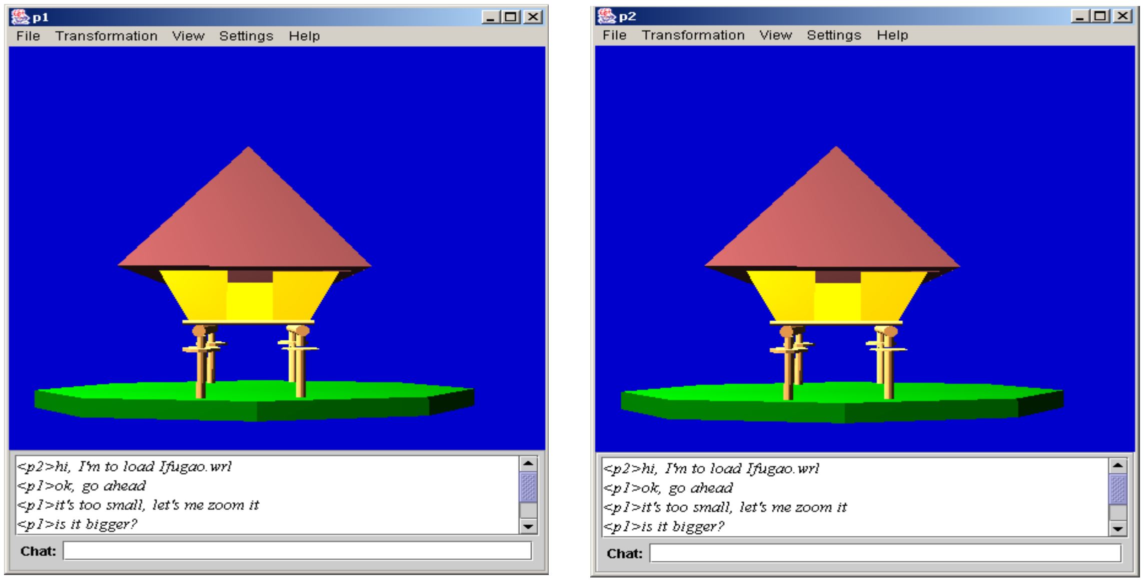
During the course of their engagement, collaborators may carry on a text chat with the built-in chat facility or use any other communication tool available for audio or video conferencing. When it is time to end the session, all that is required is to shut down the application.

5. DISCUSSION AND CONCLUSIONS

The immersive desk was tested with a variety of visualization applications and games. It had no difficulty running stereographic Quake under Microsoft Windows and automatically ran a wide range of freeware and commercial stereo-enabled visualization applications written in C++ with OpenGL and Java3D. It was found that screen placement about 27" off the floor worked well for users who sat or stood near the desk. Indeed, the system excelled for demonstrations to groups of about a dozen. Whether sitting or standing, within a range from three to fifteen feet, the stereographic effect was strong. In addition, the GyroRemote in-air mouse worked well for lectures, allowing the presenter to move about the room, while still interacting with the data. One significant advantage the immersive desk has over our original stereographic table[11] is its vertical screen. The horizontal screen on the stereographic table could only accommodate two workers comfortably and its twenty-nine inch height was limiting for shorter viewers who had difficulty with the lower angle of view across the horizontal surface. The vertical screen on the immersive desk remedied these problems.

LCV was tested with the immersive desk at one node and a standard PC at another, over closed and open local area networks using TCP. Generally, LCV worked as intended. Collaborative users could load models, manipulate objects, and modify scene and rendering parameters. LCV worked best within a closed network, a VLAN (virtual LAN), and on an open network with low traffic. However, LCV did not always collaborate as designed. In particular, a receiving peer would wait for messages that did not arrive. Indeed, system latency increased significantly as traffic increased. In an attempt to characterize these problems, tests were performed that isolated a set of network switches, suggesting that these switches service far more traffic than warranted. Investigations are currently underway to track down the source of these problems.

Another source of latency may be the JXTA pipe itself. Basic pipes in JXTA are designed as asynchronous, unidirectional, and unreliable. Because of the pipe's unreliability, there is no guarantee that messages will be delivered, hence leaving the receiving peer to wait indefinitely for a lost message. Indeed, members of the JXTA community have observed that length-increased messages never get delivered. Furthermore, Seigneur, Biegel, and Jensen suggest that the roundtrip time for JXTA message passing for "TCP-only" communication can be so long that it can interrupt a user's flow of thought. As a result, they conclude that the Java implementation of JXTA may not be ready for building responsive applications.[21] Our tests have yet to demonstrate this to be a problem for LCV either over a closed network or VLAN.


p1
File Transformation View Settings Help
Chat:
p2
File Transformation View Settings Help
Chat:

<p2>hi, I'm to load Ifugao.wrl
<p1>ok, go ahead
<p1>it's too small, let's me zoom it
<p1>is it bigger?

<p2>hi, I'm to load Ifugao.wrl
<p1>ok, go ahead
<p1>it's too small, let's me zoom it
<p1>is it bigger?

Figure 4. Synchronized peers viewing scene.

One final comment should be made about latency. Since broadcasting is used in a LAN connection for message discovery, the source peer receives its own message. As a result, a source self-checking mechanism is employed to prevent duplicate rendering at the source peer, which may cause delays in synchronization of scene updates between the source and receiving peers. However, for most non-continuous scene modifications, such as changing the background color, there appeared to be no perceptible latency. The reason is that LCV was designed so that each peer was responsible for its own graphic rendering, and messages sent between peers are compact (containing only a few bytes of data). But latency can occur when there are continuous scene updates within a short time period. For example, mouse-controlled picking-rotation behavior sends a series of messages as a user drags the mouse across the viewing window that reflect each updated mouse position and rotation matrix. Because, the receiving peer must continually interpret the messages and update its own scenegraph, there may be a slight delay in rendering.

Loading and saving VRML files remains an open issue. Since VRML files are the most commonly used 3D files over the web, we tested third-party loaders for VRML files including: NCSA's Portfolio, Web3D's XJ3D, and Cyber97. All load VRML2 files correctly, but with some differences. Other than VRML files, Portfolio supports the most varied types of other 3D files. But it is no longer under active development. Cyber97 was not selected because of its limited support for converting VRML nodes into corresponding Java3D nodes.[22] XJ3D was chosen as the VRML file loader. XJ3D is an open source project to develop a spec-complaint implementation of the X3D and VRML97 specifications.[20] It was started with a donation of source from Sun Microsystems in 1998 and today is a project of the Web3D consortium's source task group. It is still under very active development, meeting our need to build a system that can evolve with new technologies. As such, XJ3D is not a perfect VRML file loader, with some node types not yet supported. In addition, Java3D's ability to save modified scenegraphs is limited to simple scenes originally created with Java3D, not the complex nodes converted from VRML by LCV. However, there may be a way to save a LCV scene by creating a set of serialized classes from the unserialized Java3D scenegraph structure. Work continues on resolving this issue.

Java3D supports stereoscopic viewing through the underlying OpenGL API. In theory, any video card that supports OpenGL stereo should be able to display the stereo effect created in Java3D. However, there are video cards that do not natively support stereographic display. For example, machines tested with the Nvidia Geforce2 MX/MX400 video cards that use the OpenGL API, do not support programs in Java3D stereo. Even when augmented with a stereo patch from Nvidia, OpenGL stereographic programs functioned, but Java3D stereo did not. High-end video cards, such as the WildCat III 6110 card from 3D Labs, worked because they have native stereo support. In addition, Java3D stereo display settings must be tuned to optimize the final stereographic effect. Default stereo settings in Java3D are for normal CRT monitors, not the projection screen of the immersive desk, and default stereoscopic settings are for head mounted displays. As a result, eye position must be set manually to achieve a better sense of the stereoscopic effect. And since eye separation and its policy-related data must be adjusted for different users and different display devices, LCV provides a GUI widget for the user to adjust these values.

In conclusion, this research suggests that JXTA should be a powerful tool in building, adapting, and deploying P2P collaborative systems. By focusing on embedding JXTA collaborative code within a program's GUI, it is possible to quickly transform the design for a single user program into a two-person, peer-to-peer system. As a result, collaboration across the web becomes easier to initiate, more spontaneous, and supported by a wide range of software. It also suggests that further work is needed to characterize JXTA messaging and communication within real collaborative applications. We are currently looking into these issues.

ACKNOWLEDGEMENTS

This project was underwritten in part by Pace University's Center for Advanced Media and by National Science Foundation Grant ANI-0125043. The author would also like to thank Susan Merritt, Dean of the School of Computer Science and Information Systems, for her continued support.

REFERENCES

1. S. Singhal and M. Zyda, *Networked Virtual Environments*, Addison-Wesley, 1999.
2. W. Hibbard, "VisAD: Connecting people to computations and people to people," *Computer Graphics* **32(**3), pp. 10-12, 1998.
3. J. Wood, H. Wright, K. Brodlie, "Collaborative visualization," in *IEEE Visualization '97*, pp. 253-260, 1997.
4. P. Isenhour, J. Begole, W.S. Heagy, and C.A. Shaffer, "Sieve: a java-based collaborative visualization environment," in *IEEE Visualization '97 Late Breaking Hot Topics Proceedings,* October 22-24, pp. 13-16, 1997.
5. A. Pang and C.Wittenbrink, "Collaborative visualization with CSPRAY," *IEEE CG&A* , (March/April) pp. 32-41, 1997.
6. C. Bajaj and S. Cutchin, "Web based collaborative visualization of distributed and parallel simulations," in *Proceedings of IEEE Parallel Visualization and Graphics Symposium*, October, pp. 47-54, 1999.
7. M.Abbott and L.K. Jain, "DOVE: distributed objects based scientific visualization environment," in *ACM 1998 Workshop on Java for High-Performance Network Computing*. ACM Press, 1998.
8. C. Cruz-Neira, D. Sandin, D., and T. DeFanti, "Virtual reality: the design and implementation of the CAVE," in *Computer Graphics* Proceedings, Annual Conference Series*,* ACM SIGGRAPH, pp. 135-142, 1993.
9. M. Czernuszenko, D. Pape, D. Sandin, T. DeFanti, G. Dawe, and M. Brown, "The immersadesk and infinity wall projection-based virtual reality displays. *Computer Graphics,* ACM SIGGRAPH, **31(2)**, pp. 46-49, 1997.
10. W. Krueger and B. Froehlich, "The responsive workbench," *IEEE CG&A* **14**(3), pp. 12-15, 1994.
11. F.T. Marchese, "A stereographic table for biomolecular visualization," in *Proceedings of Information Visualization: IV'02,* July 10-12, IEEE Press, pp. 603-607, 2002.
12. F.T. Marchese, J. Mercado, and Y. Pan, "Adapting single-user software for collaborative use," in *Proceedings of Information Visualization: IV'03,* IEEE Press, pp. 252-257, 2003.
13. D. Brookshier, D. Govoni, and N. Krishnan, *JXTA: Java P2P Programming*. Sams, 2002.
14. B. Wilson, *JXTA*, New Rider's Publishing, 2002.
15. A.E.Walsh and D. Gehringer, *Java3D: API Jump-Start*, Prentice-Hall, 2002.
16. R.M. Taylor, W. Robinett, V.L. Chi, F.P. Brooks, Jr., W.V. Wright, R.S. Williams, and E.J. Snyder, "The nanomanipulator: a virtual-reality interface for a scanning tunneling microscope," in *Computer Graphics* Proceedings, Annual Conference Series*,* ACM SIGGRAPH, pp. 127-134, 1993.
17. S.T. Jones, S.E. Parker, and C.C. Kim, "Low-cost high-performance scientific visualization," *Computing in Science and Engineering*, July/August, pp. 22-27, 2001.
18. J. Leigh, G. Dawe, J. Talandis, E. He, S. Venkataraman, J. Ge, D. Sandin, and T.A. DeFanti, "AGAVE : access grid augmented virtual environment," in *Proceedings of the AccessGrid Retreat*, Argonne, IL, 16 January 2001.
19. "Collaborative AVS," http://www.tacc.utexas.edu/cavs/overview.html, current December 04, 2003.
20. "Xj3D Open Source VRML/X3D Toolkit", http://www.web3d.org/TaskGroups/source/xj3d.html, current December 04, 2003.
21. J-M. Seigneur, G. Biegel, C. D. Jensen, "P2p with JXTA-Java Pipes," in *Proceedings of the 2nd International Conference on the Principles and Practice of Programming in Java*, ACM, 2003.
22. S. Konno, http://www.cybergarage.org/vrml/cv97/cv97java/overview/, current December 04, 2003.