Sharing Visualization Experiences among Remote Virtual Environments

Terrence L. Disz, Michael E. Papka, Michael Pellegrino, and Rick Stevens
Mathematics and Computer Science Division, Argonne National Laboratory
Argonne, IL 60439 USA
{disz,papka}@mcs.anl.gov

Abstract

Virtual reality has become an increasingly familiar part of the science of visualization and communication of information. This, combined with the increase in connectivity of remote sites via high-speed networks, allows for the development of a collaborative distributed virtual environment. Such an environment enables the development of supercomputer simulations with virtual reality visualizations that can be displayed at multiple sites, with each site interacting, viewing, and communicating about the results being discovered.

The early results of an experimental collaborative virtual reality environment are discussed in this paper. The issues that need to be addressed in the implementation, as well as preliminary results are covered. Also provided are a discussion of plans and a generalized application programmers interface for CAVE to CAVE will be provided.

1 Introduction

Sharing a visualization experience among remote virtual environments is a new area of research within the field of virtual reality (VR). The major work that has been done in this area has been done mainly in the area of networked nonimmersive workstation-based VR [7, 10]. In this paper we discuss the issues encountered when developing a software library used to connect several CAVE Automatic Virtual Environments (CAVEs) together. When we refer to "CAVE" during the course of this paper we mean the CAVE simulator, the ImmersaDesk, and the actual CAVE. We discuss simple test cases and measurements and present an application programmers interface (API) for developers of CAVE applications to use in joining multi-CAVE sessions. This work is part of a larger project (LabSpace) to implement distributed collaborative workspaces, with multiple CAVE interaction as just one of the many communication modalities [14].

2 CAVE

The CAVE is a virtual reality environment originally developed at the Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago and now an active research project at EVL, Argonne National Laboratory, and the National Center for Supercomputing Applications [5]. In its current implementation, the CAVE uses three projectors, displaying computer images on two walls and the floor of a ten-foot cube (Figure 1). Images are projected in stereo, so that a user wearing stereo glasses can see the images in true three-dimensional space. The user's position and orientation are tracked by an electromagnetic tracking system, thereby allowing the environment to be rendered in correct viewer-centered perspective. The user is able to manipulate objects and navigate within the CAVE by using a wand, a three-dimensional analog of the mouse of current computer workstations. The size of the CAVE, approximately 10' x 10' x 10', allows several people to be in the CAVE and share the experience. While only one user is tracked and has the correct perspective, experience shows that other users in the CAVE wearing stereo glasses see a satisfactory image.

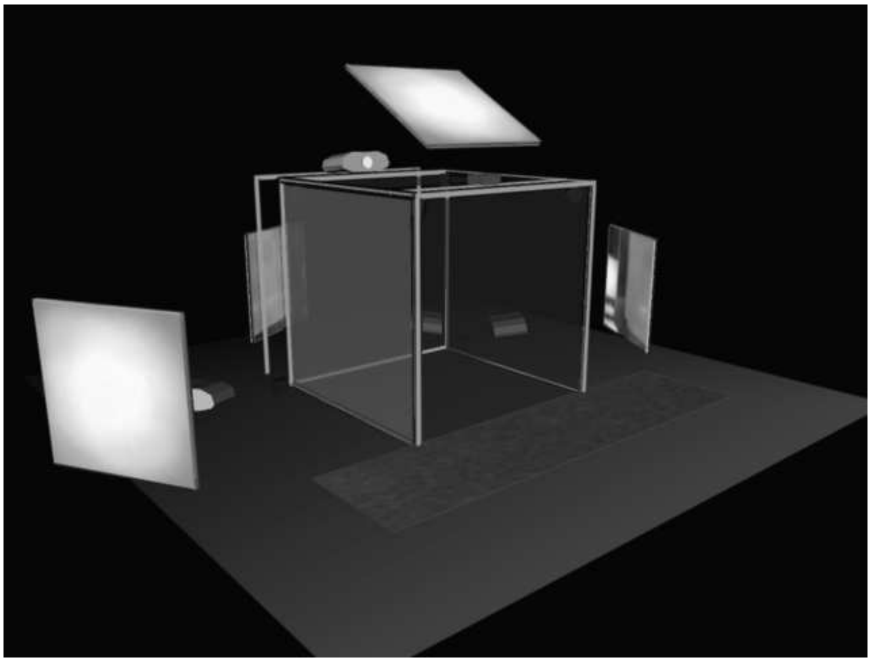

Figure 1: CAVE Virtual Environment (Milana Huang, EVL, 1994)

CAVE simulators are available to anyone having access to a Silicon Graphics workstation. A recent low-cost addition to the CAVE family of VR devices is the ImmersaDesk, which is a one-wall CAVE the size of a standard drafting table.

Several CAVEs are in operation at this time around the country, including one at each of the three major CAVE development research sites, with several more being planned for construction. Each of these CAVEs is being used for interactive visualizations of applications being run on supercomputers [11]. At Argonne, for instance, researchers have developed a drug design application, a mesh refinement demonstration, and a finite element analysis application [4].

Many of the applications, both those under way and those being planned, involve collaboration with universities and other national laboratories. Since these applications are interactive, and since the CAVE currently requires that all participants be in the same physical space, collaborators are burdened by the need to travel to one of the CAVE sites for demonstrations or testing.

To address this problem, we are investigating ways to remove the barrier of distance while sharing a virtual experience. The goal is to enable the use of the CAVE as a distributed and collaborative environment. Achieving this will remove the limits of using CAVE technology only at sites that have expensive supercomputers and full CAVE setups, thereby allowing users anywhere to join in the exploration with as little as a CAVE simulator. We call this project the CAVE to CAVE project.

3 Preliminary CAVE to CAVE Experiments

In the Mathematics and Computer Science Division at Argonne National Laboratory, a Futures-Lab group meets weekly to discuss computing futures issues, present current work, hear guest speakers, etc. After we installed our CAVE in July of 1994, the discussion often turned to new and innovative ways to utilize the CAVE. We spent countless hours discussing new paradigms for using it, arguing about the "right" way to design new libraries of functionalities and even what to call some of these imagined functions. One point we agreed on, however, was that there was an obvious opportunity to use two or more CAVEs to share a VR experience. What we could not agree on was just how to accomplish that. The personnel involved with the other two CAVEs in Illinois, at the EVL and NCSA, have also had these discussions and have run various unpublished experiments. Somewhat belatedly, we finally realized that there was too much that we did not know to effectively discuss alternative implementations, and we decided to run a series of experiments.

3.1 Experiment 1: Simple Transmissions and Representation from one CAVE to another

The first experiment we ran was to have two CAVEs continually transmit the head and wand locations of their occupants to one another. We represented the position of the occupant of the remote CAVE with a large red sphere and the wand position with a small yellow sphere. We wanted to see whether the two occupants could interact in any meaningful way. We wrote the application using an existing socket library, and we recruited participants from the EVL CAVE to help us. We knew from previous experiments that roundtrip time for a TCP/IP message from the EVL CAVE to ours is typically 30 to 50 ms. This limited us to about 20 updates per second, good enough for smooth animation of the representation.

> When running a CAVE program, the library forks separate drawing processes that communicate with the update process through shared memory. The update process computes new locations for objects and updates the shared memory region. There is not necessarily any synchronization between the update process and the

drawing processes. This configuration allows the CAVE to maintain a relatively constant screen update rate, independent of the ability of the update process to produce timely scene changes. It is this decoupling of the update and drawing processes that allows us to have the update processes stream position data and still maintain a good refresh rate on the screens.

We learned important lessons right away regarding the representation of the other person, and about the importance of orientation. The problem was that since the sphere looked the same from all sides, there was no way for a CAVE occupant to know where the front of the representation was, and so there was no easy way to cooperatively move together or to move toward or away from one another.

3.2 Experiment 2: Person Orientation Clues

The next experiment we ran was to add "eyes" to the sphere, small spheres that always pointed in the same direction in which the CAVE occupant was looking. This helped quite a bit, but pointed out the problem of CAVE orientation. Since we operated both CAVEs within the exact same reference frame, they were both oriented in the same direction. Since we can project only on two walls, one CAVE occupant could see the other only if he was between the viewer and a projected wall.

This situation, of course, caused difficulty because each occupant continually maneuvered (in circles) to place the representation between himself and a projected wall so as to see the representation.

This problem was the subject of much discussion, with simple fixes proposed such as "rotate one CAVE's reference frame by 180 degrees." While this debate continued, we decided to add more functionality to the existing representation.

3.3 Experiment 3: Additional Information

We added information about the wand position and drew a "stick man" with a head at the same height as the remote CAVE occupant, with two legs and arms and a wand attached to one of the arms (Figure 2). As the remote occupant moved his wand around, the representation in the other CAVE did the same. As the occupant of the remote CAVE turned around, stooped, jumped, or walked around, so did the representation. By running the remote CAVE from a simulator in the same room as the CAVE, we were able to provide out of band audio communications to complete the "point and say" tutorial-type model of CAVE to CAVE interaction. This approached looked like it would be quite effective, so we started another experiment.

3.4 Experiment 4. Generalization of the Model

To begin generalizing the model, we next developed a server to facilitate message exchange between CAVEs. We added the server and the representation to our molecular dynamics visualization application, an existing CAVE application for which we own the source code [6]. We were able to see immediately that our intuition was right: one could conduct a meaningful "point and say"

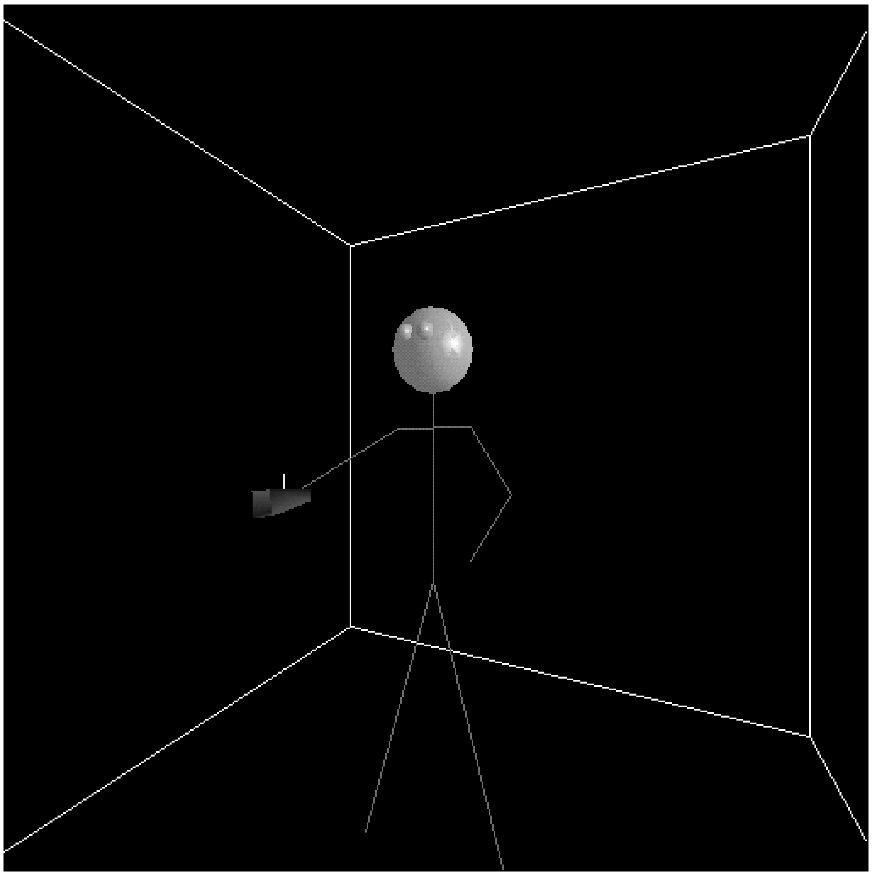

Figure 2: Stick man

tutorial-type interaction in a real CAVE application using an audio channel and only a "stick man" representation of the position of the occupant of the remote CAVE. We still had the orientation problem, and sometimes a CAVE occupant would get lost inside the representation of the occupant of the remote CAVE, but we were satisfied that this had the potential to be an important new use of the CAVE.

3.5 Experiment 5. Orientation and Navigation

To solve the orientation problem, we decided to operate within a larger world-coordinate system understood by both CAVEs. We developed world- to-local space transforms and added that functionality to the simulations. We placed the simulation in the space, and located the remote CAVE in some different part of the space from the local CAVE. Now, it became easy to see where the other person was in his CAVE, and in relation to ourselves. We added navigation functions that allowed one to steer and rotate his CAVE to the front of the remote CAVE, enabling easy and natural ways to see one another. To interact, one needed simply to rotate and steer his CAVE to intersect with the representation of the remote CAVE (Figure 3). The two occupants could then interact with the same part of the data. During these experiments, we continued to have the CAVEs stream data to each other, through the server, using the MPI message-passing software. We started the processes using the p4 process startup as part of the p4 MPI layer. Next, we wanted to see what it would be like to have more than one remote CAVE.

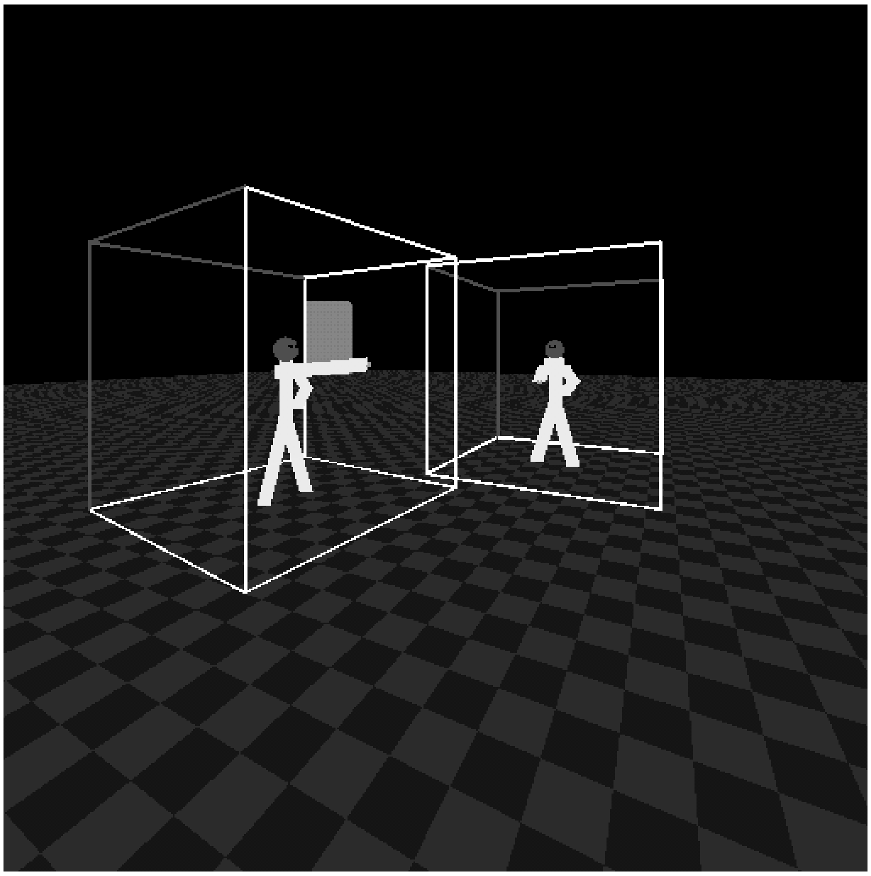

Figure 3: Snapshot of virtual users taken from CAVE simulator

3.6 Experiment 6. Scaling the number of CAVES

We ran tests adding remote CAVEs to the experiments and learned that there was an inverse linear relationship between the number of CAVEs added and the update rate we could expect in our CAVE. At ten remote CAVEs, the system was unusable because of lag and slow response time. Our solution was to vary the rate at which the CAVEs transmitted their location. By having every CAVE transmit only on every Nth time step, where N is the number of CAVEs participating in the session, we found we could sustain smooth animations of the representations and not noticeably degrade performance of the simulation. With more than ten CAVEs participating, we started to notice a jerky movement of the representation of the CAVEs, due to the fact that we were not sampling the position often enough. We believe that these effects can be mitigated through the use of lag compensation algorithms [15]. At this point, we had learned enough to be able to discuss various types of CAVE to CAVE scenarios. In doing so, we were able to articulate the issues involved in developing a model and to propose a general programming model to add to the CAVE library for CAVE to CAVE interaction.

4 CAVE to CAVE Scenarios

When developing the support needed to address the issues of a Collaborative Distributed Virtual Environment (CDVE), we need to study the basic components of a CAVE application. Considering past experiences, we find that most of our applications fall into one of the following categories:

- Real-time connection to interactive simulation, either running locally on the CAVE graphics computer or running on a remote supercomputer.
- Playback of precomputed data with or without interaction, with the data either residing locally or on a remote system.

What follows is a general discussion of these basic CAVE application issues, their relationship to CDVE, and examples of applications. The examples will be discussed in two ways: how the application currently is implemented, and how it could be added to a CDVE.

4.1 Real-Time Connection Issues

One of our primary virtual environment interests is the connection of the CAVE to computer simulations, running in real time on either our local IBM SP2 or via a high-performance network to remote supercomputers. Real-time connection of the CAVE to a computer simulation permit interactive steering. Thus the user of the CAVE can make judgments and push the simulation toward user-defined goals. This configuration, combined with a CDVE, would allow multiple sites to view the simulation at the same time and allow remote collaborators to work together in new ways.

4.2 Real-Time Examples

An example of a real-time connection is the interactive molecular modeling application developed by Carolina Cruz-Neira, Paul Bash, and others [8]. In this application, the user guides the docking of a drug molecule to its molecular receptor. As the user guides the drug molecule into the active site of a protein, he/she receives real-time feedback from the simulation running on the IBM SP2. Another real-time connection is the simulation of a grinding process by Tom Canfield et al. [3]. In this application, the user controls the placement and force of material against a grinding wheel (Figure 4). This placement controls a finite element analysis running on the IBM SP2. Real-time feedback is provided: the coloring and shading indicate thermal stress on the material and the wheel.

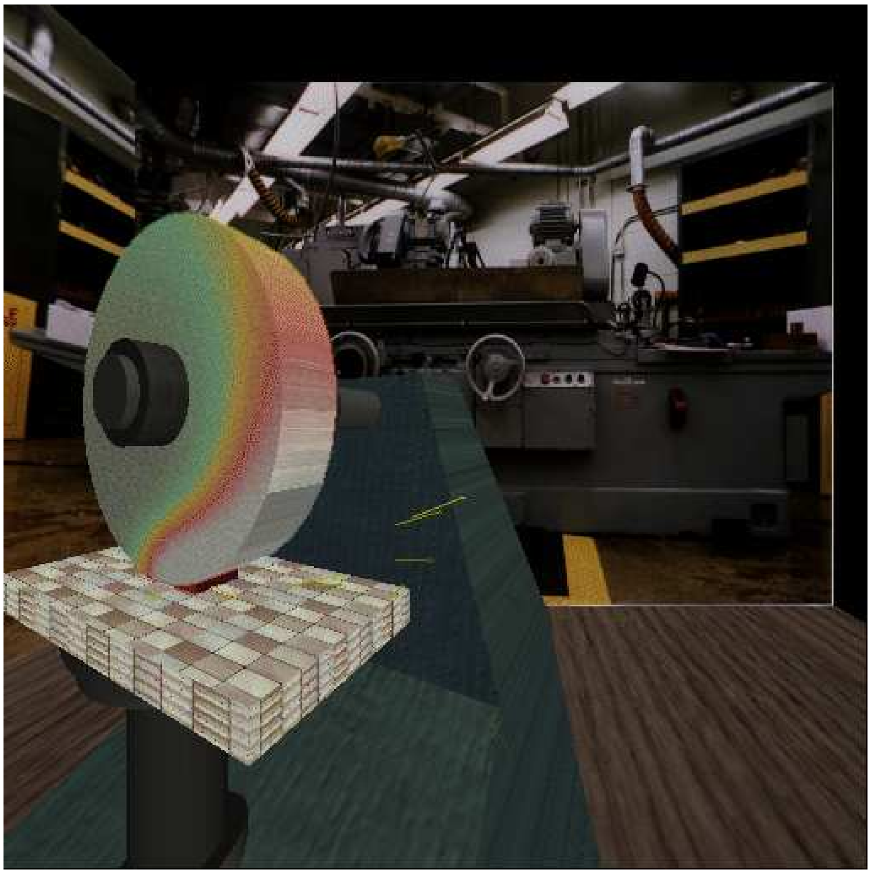

Figure 4: CAVE Grinder Application (Shannon Bradshaw, ANL, 1995)

4.3 Real-Time CAVE to CAVE Issues

Traditionally a real-time CAVE application communicates with a remote supercomputer over an arbitrary communications package. The CAVE sends button and joystick readings and user and wand positions to the simulation. The simulation in turn responds with data needed for the visualization (Figure 5).

In a CDVE environment there is still a controlling CAVE (Master) and a remote supercomputer, but additionally there is the possibility of multiple viewers at distant CAVEs (dCAVEs). The master CAVE directs the information to the simulation, and the simulation in turn broadcasts the results

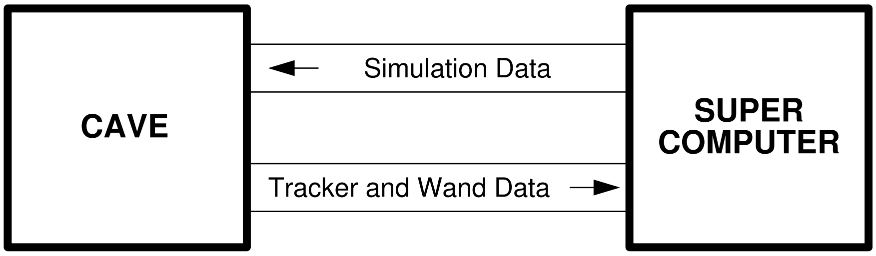

Figure 5: Single CAVE Supercomputer Simulation

to all the participating dCAVEs (Figure 6). If the developer chooses, remote representations of other participants can be displayed in each dCAVE.

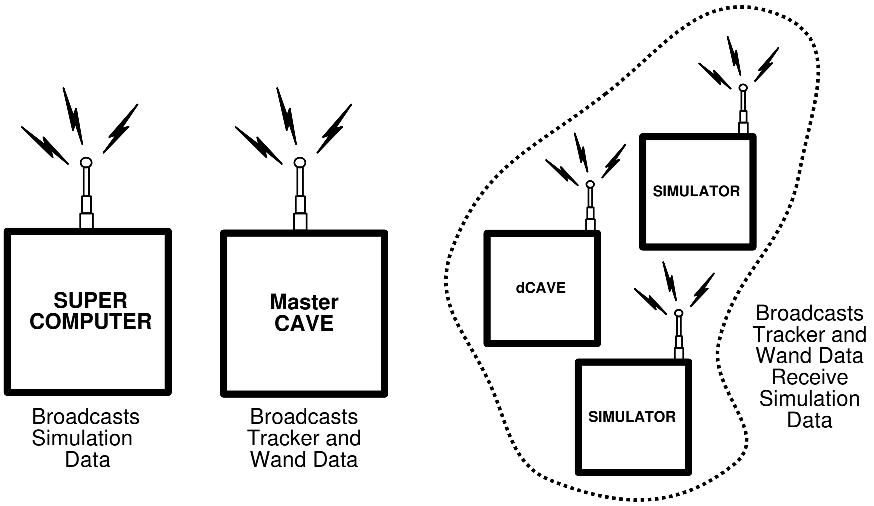

Figure 6: CAVE to CAVE Supercomputer Simulation

The following issues are raised when we consider sharing these types of simulation among two or more CAVEs assuming all participants share one simulation:

- How is control of the simulation arbitrated?
- What protocol is in place for subscribing/leaving sessions?
- How much bandwidth is required to transmit
    - Control data?
    - Application data?
- How are virtual users of the dCAVE represented?

4.4 Precomputed Data Issues

We have developed several examples of animated playback applications. Typically these applications are ones in which the simulation cannot be run in real time, or where the simulation has not yet been parallelized or ported to an appropriate platform.

Applications of this type are characterized by frequent independent calls to stop, go back, go forward, etc. as individuals express their preferences in exploring the data space. Users are often required to navigate around the virtual world and to manipulate objects within the world. In a shared experience with multiple viewers in the same CAVE, the additional viewers can be thought of as riders on a tour bus, with the user controlling the navigation acting as the tour guide. These applications are largely tutorial in nature.

4.5 Precomputed Data Examples

An example of animated playback is the simulation of a casting process [13]. The simulation of the process requires too much computation time to run it in real time. Therefore a number of timesteps are written to files and then animated in the CAVE. Based on how the data is stored, some interaction is allowable. In this application, for example, the user can look at different temperature surfaces; to achieve this capability, isosurfaces are computed in real time. Additional control of the playback is done with a VCR-like control panel.

Navigation of architectural space is another example of using precomputed data. The reactor walkthrough application developed an ANL by Randy Hudson et al. was designed to provide an inside view of a Fast Breeder Reactor at the Argonne West reactor facility in Idaho. Operators of the reactor had never seen the inside and could only imagine what internal conditions were occurring in response to their manipulation of the reactor controls. The application allows CAVE passengers to navigate around and within the reactor, remove parts to improve visibility, and run fuel-handling sequences.

4.6 Precomputed Data CAVE to CAVE Issues

When multiple CAVEs are connected, a whole new range of possibilities must be addressed.

Assuming each CAVE user has his own copy of the data and can play it back independently, the following issues are relevant:

- How are session participants represented?
- How is the state of other sessions represented?
- How does one "join" another session?
  - Go to the same place in the playback?
  - Go to the same viewpoint?

5 CAVE to CAVE Issues

The scenarios discussed in the preceding section provide us with a way to think about the nature of CAVE to CAVE interactions. We see collaborative-type interactions, where users can independently explore the data set, pointing and saying things; shared exploration of the data space, with accompanying pointing and saying actions; and tutorial type interactions, which require shared navigation and have a predominately one-way pointing and saying interaction.

By considering the above scenarios, we are able to focus on five issues that are of immediate importance to our users in our exploratory attempts to achieve useful shared CAVE experiences:

- Session Management (Connection/Authentication, Brokering)
- CAVE to CAVE Reference Frame
- Representation of Collaborators and Their CAVEs
- Synchronization (Events and Tracking)
- Navigation

5.1 Session Management

A session is a multiple CAVE and/or supercomputer interaction. A method for process startup must be defined, and if necessary, copies of static data sets must have been previously made available to each copy of the application. Once the session is begun, the server provides session management through control data streams. An API is provided for the following functions:

- Registration: notifies others of available service
- Session status: allows participants to learn about the presence of other participants and of existing sessions
- Session attachment and detachment
- Data subscription and cancellation
- Subscription to predefined data from other CAVEs, such as trackers and buttons, and to user-defined data from other CAVEs or from participating supercomputers

Connection to data sources depends on the context of the data. If the data is to be shared in such a way that each user can modify the data, then a way of communicating that change must be determined to keep scenes synchronized among the various CAVEs.

Precomputed datasets can be copied to each machine at startup to minimize network traffic of data being transferred on demand. If datasets become too large, this may not be possible. The use of compression schemes should also be explored to determine whether certain types of data can be compressed without loss of meaning or content.

Datasets that are generated on demand by a CAVE that is controlling a simulation, but being viewed by multiple CAVEs will need to broadcast the content of the calculated data to all CAVE's involved. Datasets that are generated on each individual CAVE will need to synchronize simulations, once one viewer wants to see what another CAVE is doing. Ideally one would not want to synchronize simulations but instead become a viewer of the CAVE of interest. Once the interested party is done viewing the other CAVE, he can go back to working where he left off. The other needed feature is to be to be able to synchronize simulations so that one could start exploring from the other CAVE's location without being tied to watching only that CAVE.

Interactive precomputed datasets would need to use a combination of the two situations described above. As a means to lower network traffic one could allow the precomputed data to reside on each participating CAVE. CAVEs would need to synchronize to keep the movies in step.

5.2 CAVE to CAVE Reference Frame

A world reference frame becomes a requirement when building a CAVE to CAVE library. The CAVEs will not only have their own local coordinate system but will need to be able to broadcast their positions to all other participating CAVEs. This can be done by developing a global coordinate system where each CAVE broadcasts its location within the world and then each participating CAVE is responsible for reconstruction of that CAVE at that location. The simplest case involves the displaying of the dCAVE's user in one's CAVE. From that lowest level one can expand the amount of detail of the dCAVEs such as position, orientation, and what is being viewed. A protocol will eventually be proposed for establishing reference synchronization.

5.3 Representation of Collaborators and Their CAVEs

The quality of rendering of the dCAVE's representation required to give the feeling of presence is not a quantifiable measurement. The higher the quality, the more realistic the feeling can be, but this also brings up the question of computational cost of the representation. A very realistic representation of the dCAVE user will slow the rendering of CAVE visualization by requiring more polygons to be drawn. In terms of the tutor scenario, only the representation of the teacher needs to be drawn in each of the dCAVEs; the location of the students is of no concern. On the other hand, the navigation of a space may require the representation of all dCAVE's in each individual's CAVE. In this case a lower-quality representation will work [12].

Through our experiments we have found that the use of an audio channel greatly enhances the CAVE to CAVE interaction.

5.4 Synchronization

Synchronization signals must be defined for the following:

- Control arbitration
- Position/viewpoint

- Animated playback
- Data sets
- Animation state
- Direction, speed, viewing options, timestep
- Static object or space examination
- Viewing state
- Object features (i.e. transparency)

We need more experience in order to determine how frequently synchronization signals must be passed, how much network latency can be tolerated, and how much data is required.

5.5 Navigation

A continuing thread in the development of the CAVE to CAVE library is the representation of information from one CAVE to the next. Navigation of one's own CAVE about its space is now a standard part of the CAVE library. How to represent the movement of the dCAVEs in one's own CAVE is an issue. How to tell which CAVE is navigating, how to pass the control of navigating from one CAVE to the next, and how network latency affects the experience are all under study.

6 The Model

6.1 LabSpace architecture

The LabSpace architecture proposes a mediated client/server model. (Figure 7) The broker provides session management functions, while data is communicated directly between clients and servers. We have developed our own server to mediate sessions between CAVEs, supercomputers, or other network citizens adhering to our protocol.

6.2 Our Programming Model

In developing the programming model, we used several guidelines. First, the general model had to fit into the overall LabSpace model of mediated client/server. We anticipate that our programming model will eventually be subsumed by the LabSpace architecture. Second, we wanted the library to be robust and extensible. We wanted to provide all the functionality required to satisfy the scenarios we mentioned above and still provide enough functionality and extensibility to support other scenarios that we had not thought of. Third, we wanted to present a simple layer to the applications programmer to hide most of the complexity of performing CAVE to CAVE communications. Fourth, we wanted

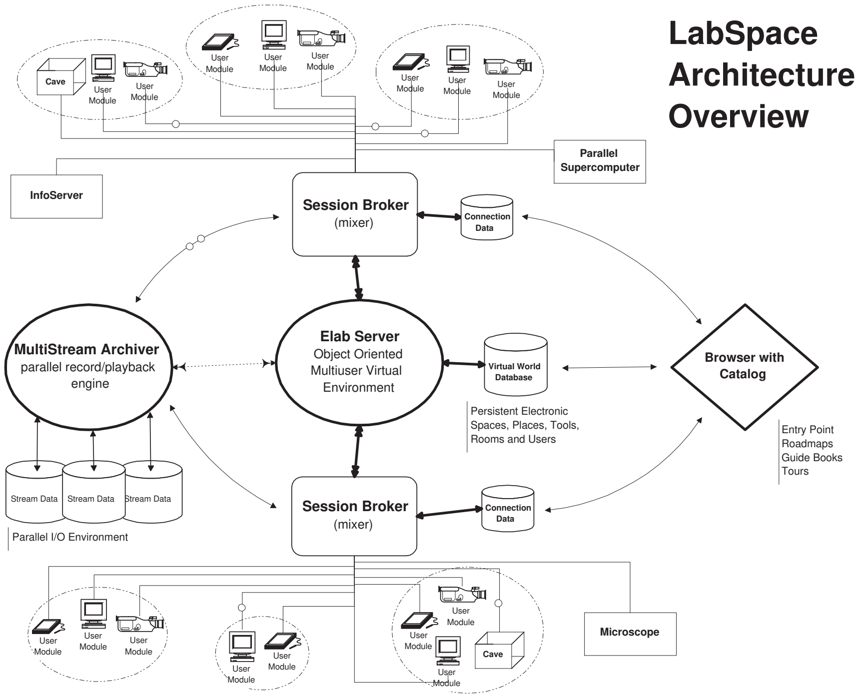

Figure 7: LabSpace Architecture Overview

to make use of existing portable standards-based software wherever possible (Figure 8).

The main library functions in the CAVE to CAVE protocol layer are described in the API found in Appendix A. The functions are designed to facilitate session management, data management, and communications. We have not yet designed a library to facilitate inter-CAVE object management, but we intend to in our next iteration of the library development.

6.3 MPI

Notice that we have used the MPI message-passing system as an intermediate portable communications layer. While satisfying many of our requirements (portable, standards based, efficient, etc.), MPI falls short in several important areas.

First, MPI communicator groups are static, while we require that CAVEs be able dynamically to join and leave sessions. We have worked with the MPI developers at Argonne to specify MPI extensions that provide for dynamic communicator groups. We have developed the underlying layer to implement these extensions, soon to be available in the next release of MPICH [9]. Another

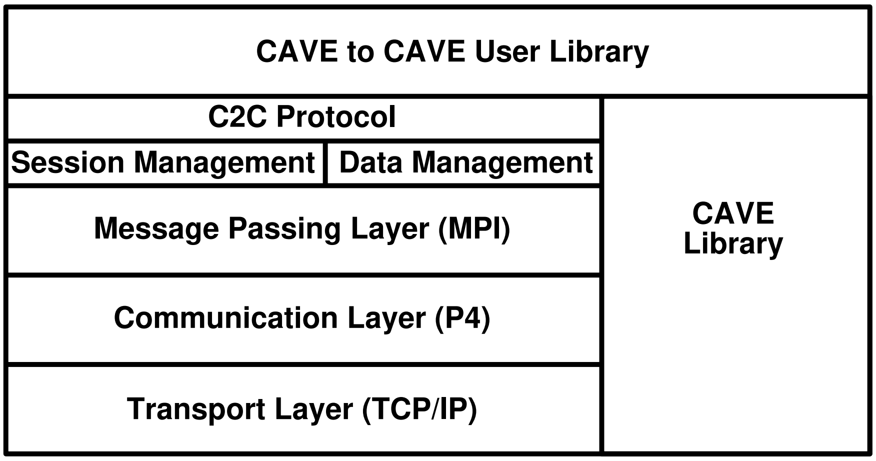

Figure 8: CAVE to CAVE Library

feature we require, also available in the next release of MPICH, is the ability to support multiple-protocol communications (e.g. SP2 switch and TCP from SP2 to another machine in the same MPI program). Second, MPI deliberately does not specify any means of process startup. We have decided not to propose any development in this area while we wait for the development of the labspace system which will address the issues of authentication and security in remote process start up. In the meantime, we use the p4 secure server to start processes on remote systems [1, 2]. Lastly, we anticipate that for the sake of efficiency, we will require a multicast-like capability to adapt to dynamically changing low-level link configuration. This feature is not available in any MPI implementation today.

6.4 User Level API

Using these library functions, we have written a simple layer for application programmers to use in making their applications available to CAVE to CAVE sessions. We provide the following functions for consideration:

- void C2CInit(int argc, char *argv[])

  Initializes specific variables needed for CAVE to CAVE functions. Connects to broker as specified in CAVE config file.

- void C2CUpdate(C2C_ID_LIST request)

  Requests updates to local variables of remote CAVEs states.

- void C2CExit()

  Disconnects CAVE from broker.

- void C2CDrawRemoteCaves(C2C_ID_LIST request, void (*function)(), int number_args, ...)

If this function is not called remote CAVEs are represented by a simple stick figure. If this function is called the drawing function passed will be used to represent the remote CAVEs.

- void C2CPostData(C2C_ID_LIST request, int nbytes, char *data)

  Post a generic chunk of data to the broker bulletin board.

- void C2CGetData(C2C_ID_LIST request, int nbytes, char *data)

  Get a generic chunk of data from the broker bulletin board.

- void C2CViewServiceList()

  Graphically displays to the user what is available from the CAVE to CAVE broker.

- void C2CChangeServiceRequest(C2C_ID newService)

  Choose new service to subscribe to.

- void C2CTeleport(C2C_ID CAVE, C2C_VIEW view)

  Teleport local CAVE user to the location of chosen CAVE, with the viewpoint as specified by the variable view.

7 Communication Requirements

Using the model developed above, we have built a test bed to use in examining the boundaries of the communications requirements imposed by CAVE to CAVE interactive sessions. We have run experiments testing CAVE to CAVE latency effects, throughput requirements, and usability as the number of sessions scale. The volume and nature of data transmitted is characterized as follows:

- Tracker data: Per message - 48 bytes (12 floats) plus 24 for MPI header
- Button data: Per message - 48 bytes (12 floats) plus 24 for MPI header
- Reference Frame Information: Per message - 48 bytes (12 floats) plus 24 for MPI header
- Synchronization Data Per message - 0 bytes plus 24 for MPI header (MPI tag denotes data type).

Messages per second vary with the number of CAVEs interacting (Figure 9).

8 Results

During development of the CAVE to CAVE library a comparison of different message sending techniques was done to find the fastest MPI mechanism for sending information from each CAVE to all the other CAVES (Figure 9).

We then looked at the amount of data that needed to be sent from CAVE to CAVE, using both unicast and multicast methods (Figure 10). These results

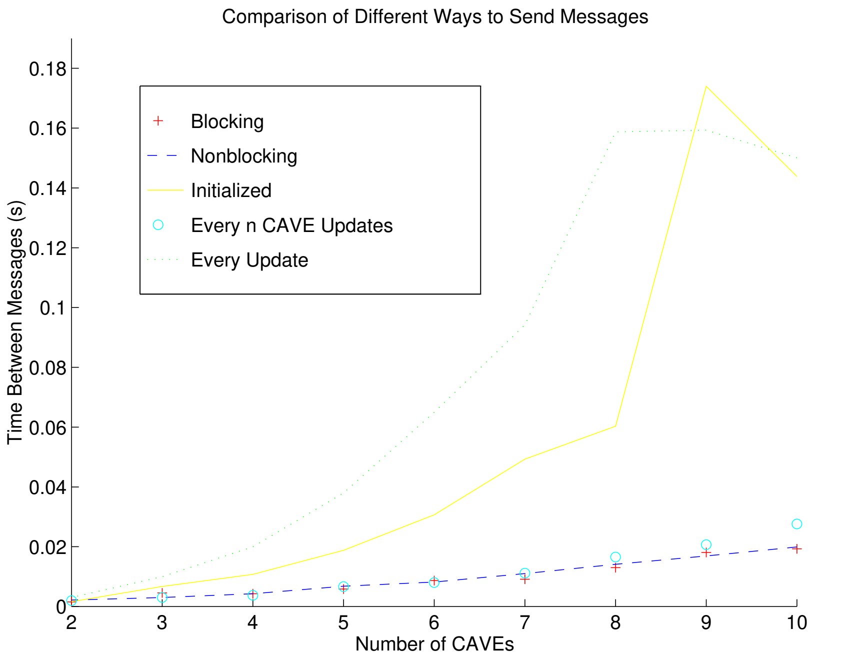

Figure 9: Measurements of time needed to send tracker data using various MPI message sending techniques.

demonstrated that a multicast method was the only reasonable method to use to avoid saturation of the available network bandwidth. In addition, even using the multicast communication method requires a network of at least OC-3 bandwidth to handle more than ten CAVEs. It should be noted that the CAVEs are sending tracker data as often as they can. It as been determined that this is not absolutely necessary, and we have worked on various techniques to reduce the number of sends.

9 Conclusion and Future Plans

Through experimentation, user survey, and group discussion, we have developed a set of requirements for CAVE to CAVE interaction. Based on these requirements, we have developed a mediated client/server model. We have implemented the model through a library of functions designed to be robust and extensible. We use the MPI message-passing system as an intermediate-layer communications library. Using our CAVE to CAVE library, we have run a series of timing experiments designed to test the boundaries of communication requirements in CAVE to CAVE interactions.

We have not addressed the issue of object sharing and manipulation between CAVEs in the current library. We plan to run experiments to discover issues involved in sharing objects and will develop a set of requirements and design a model to implement CAVE to CAVE object sharing.

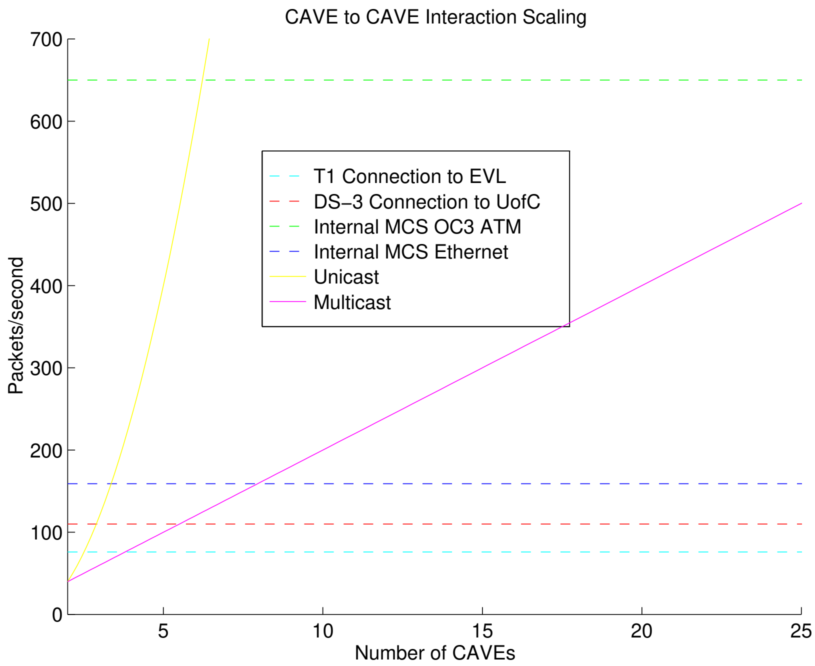

Figure 10: Comparision of Unicast and Multicast communication methods across relative networks.

Acknowledgments

The authors wish to thank Remy Evard, William Nickless, Robert Olson, and Valerie Taylor, along with rest of the Futures Lab Group for insightful discussions on this subject. This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

A CAVE to CAVE API

A.1 Audience

- CAVE
- ImmersaDesk
- CAVE Simulator
- Supercomputer Simulations
- Other Applications

A.2 Model

One or more session brokers exist. A session broker manages multiple sessions for an arbitrary group of participants. Participants are bimodal - client and server. Each participant serves the requested data to the requesting parties at the request of the session broker. Participants subscribe to one or more data streams via the session broker. Data streams are communicated directly from participant to participant.

A.3 Session Management Functions

- C2C_PARTICIPANT_ID C2CRegisterParticipant(char *broker_address, char *environment)

  Informs broker of a participant's availability and capabilities. The broker registers this information in an internal data base. Returns unique id.

- int C2CUnRegisterParticipant(char *broker_address, C2C_PARTICIPANT_ID id)

  Instructs broker to remove all knowledge of a participant and all connections to that participant.

- C2C_SESSION_ID C2CRegisterSession(char *broker_address, C2C_SESSION_INFO info)

  Informs broker of a new session that is available and the requirements and capabilities of that session. The broker adds the new session to global list of available sessions and returns the unique identifier of the session.

- int C2CUnRegisterSession(char *broker_address, C2C_SESSION_ID id)

  Removes session from global list and notifies all participants that the session is going away.

- int C2CGetSessionList(char *broker_address, int *number_sessions, C2C_SESSION_INFO *list)

  Returns the number of sessions available and each sessions capabilities.

- C2C_ID C2CAttachToSession(char *broker_address, C2C_SESSION_ID session_id, C2C_PARTICIPANT_ID my_id)

  Attaches participant to requested session.

- C2C_ID C2CDetachFromSession(char *broker_address, C2C_Session_ID my_id)

  Detaches the participant from the session.

A.4 Data Management Functions

- C2C_Subscribe(C2C_Participant_Id data_source, C2C_Stream_Type data_type, C2C_Data_Parameters parameters)

  Instructs the broker to have the participant identified by data_source start sending the requested data stream to the requester. The broker verifies that the participant has the capability to send the requested data type.

- C2C_UnSubscribe(C2C_Participant_Id data_source, C2C_Stream_Type data_type)

  Instructs the broker to have the participant identified by data_source stop sending the requested data stream to the requester.

References

[1] R. Butler and E. Lusk. User's guide to the `p4` parallel programming system. Technical Report ANL-92/17, Argonne National Laboratory, 1992.

[2] R. Butler and E. Lusk. Monitors, messages, and clusters: The `p4` parallel programming system. Technical Report P362-0493, Argonne National Laboratory, 1993.

[3] T. Canfield, W. Jester, J. Rowlan, E. Plaskacz, M. Papka, and S. Cohen. Simulation of a grinding process in virutal reality. In L. Petrovich, K. Tanaka, D. Morse, N. Ingle, J. Ford Morie, C. Stapleton, and M. Brown, editors, *Visual Proceedings*, COMPUTER GRAPHICS Annual Conference Series, page 224. SIGGRAPH, ACM SIGGRAPH, 1994.

[4] C. Cruz-Neira, T. A. DeFanti, R. Langley, R. Stevens, and P. A. Bash. Vive: A virtual biomolecular environment for interactive molecular modeling. *Science*, 1993. Submitted for Review October 1994.

[5] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *ACM SIGGRAPH '93 Proceedings*, pages 135–142. SIGGRAPH, ACM SIGGRAPH, 1993.

[6] T. Disz, M. Papka, M. Pellegrino, R. Stevens, and V. Taylor. Virtual reality visualization of parallel molecular dynamics simulation. In *1995 Simulation Multiconference Symposium*, pages 483–487, Phoenix, Arizona, April 1995. Society for Computer Simulation.

[7] R. Gossweiler, R. J. Laferriere, M. L. Keller, and R. Pausch. An introductory tutorial for developing multiuser virtual environments. *PRESENCE: Teleoperators and Virtual Environments*, 3(4):255–264, 1994.

[8] G. E. Lent, J. Rowlan, P. Bash, and C. Cruz-Neira. Interactive molecular modeling using real-time molecular dynamics simulations and virtual reality computer graphics. In L. Petrovich, K. Tanaka, D. Morse, N. Ingle, J. Ford Morie, C. Stapleton, and M. Brown, editors, *Visual Proceedings*, COMPUTER GRAPHICS Annual Conference Series, page 223. SIGGRAPH, ACM SIGGRAPH, 1994.

[9] E. Lusk. Mpich release document. World Wide Web, 1995. http://www.mcs.anl.gov/home/lusk/mpich.

[10] M. R. Macedonia, D. R. Pratt M. J. Zyda, P. T. Barham, and S. Zeswitz. Npsnet: A network software architecture for large-scale virtual environments. *PRESENCE: Teleopertors and Virtual Environments*, 3(4):265–287, 1994.

[11] T. M. Roy, C. Cruz-Neira, and T. A. DeFanti. Steering a high performance computing application from a virtual environment. *PRESENCE: Teleoperators and Virtual Environments*, 1994. To Be Published.

[12] D. W. Schloerb. A quantitative measure of telepresence. *Presence: Teleoperators and Virtual Environments*, 4(1):64–80, 1995.

[13] R. Schmitt, H. Domanus, J. Rowlan, M. Papka, and S. Cohen. Visualization of casting process in foundries. In L. Petrovich, K. Tanaka, D. Morse, N. Ingle, J. Ford Morie, C. Stapleton, and M. Brown, editors, *Visual Proceedings*, COMPUTER GRAPHICS Annual Conference Series, page 224. SIGGRAPH, ACM SIGGRAPH, 1994.

[14] R. Stevens and R. Evard. Distributed collaboratory experimental environments initiative labspace: A national electronic laboratory infrastructure, 1994. Grant Proposal.

[15] M. M. Wloka. Lag in multiprocessor virtual reality. *Presence: Teleoperators and Virtual Environments*, 4(1):50–63, 1995.