On Extending Collaboration in Virtual Reality Environments

VÍCTOR THEOKTISTO[1], MARTA FAIRÉN[1]

[1]Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, Barcelona, Spain
{vtheok,mfairen}@lsi.upc.es

Abstract. We characterize the feature superset of Collaborative Virtual Reality Environments (CVREs) out of existing implementations, and derive a novel component framework for transforming standalone VR tools into full-fledged multithreaded collaborative environments. The contributions of our approach rely on cost-effective techniques for loading graphics rendering, user interaction and network communications software components into separate threads, with a top thread for session collaboration. The framework recasts VR tools under a scalable peer-to-peer topology for scene sharing, callback hooks for event broadcasting and multicamera perspectives of avatar interaction. We validate the framework by applying it to our own ALICE VR Navigator. Experimental results show good performance of our approach in the collaborative inspection of complex models.

1 Introduction

Virtual Reality (VR) and Augmented Reality (AR) tools have been applied in all engineering fields in order to avoid the use of physical prototypes, to train in high risk situations, and to interpret real or simulated results. In medical applications they help patient monitoring, interpretation of scanned data and surgery planning. In architectural settings enable designing, building, visiting and stress-testing upcoming facilities. In these virtual reality environments or VREs, individual users inspect 3D scenes, navigate inside models and manipulate objects and properties.

Most implementations of VREs begin as standalone applications, with collaboration requests arising from the natural desire of exchanging experiences. Allowing several clients to collaborate on the inspection of a model usually requires the development of a whole new application with distributed capabilities, adding network communications, and in general confronting code portability problems due to the absence of a known migration strategy.

We propose a "snap-on" superset framework for evolving complete Collaborative Virtual Reality Environments (CVREs) out of existing VR applications. The main contribution of our approach is a novel multithreaded architecture with a scalable peer-to-peer network topology that incorporates session layer management, a crossplatform message-passing communications library, and a hybrid collaborative interaction model with multiple avatar roles. The framework adjusts easily to working VR tools without affecting graphics performance.

In section 2 we evaluate existing CVREs under a feature superset characterization, and explain relevant collaborative interface paradigms.

We develop in section 3 the generic blueprint for the transformation of VR navigators into small or medium scale CVREs, in the form of a framework providing network and session management. In section 4 we validate the framework in the conversion of the ALICE VR Real Time Inspector (developed at the Universitat Politècnica de Catalunya) into a complete collaborative VR environment.

Section 5 shows performance test results of enhanced ALICE clients using high-level display systems in a busy network set-up. Finally in section 6 we plan for extending new capabilities into the framework.

2 Collaborative Virtual Reality Environments

Applications in which remote users collaborate on tasks to accomplish a common goal fall under the term *Computer Support for Cooperative Work* (CSCW) [1]. When combined with network model sharing, 3D data visualization, and real world user-interaction metaphors they become Collaborative Virtual Reality Environments. Remote participants using visual identities (called *avatars*) may navigate inside the virtual space, interact with other remote avatars, and propagate changes to neighbouring objects.

A representative sample of existing CVREs were profiled as part of this research in [2]. The categories, summarized next, allow the designer to specify the most suitable feature set for creating a visual sense of presence within a collaboration framework

i) **Session awareness:** the enduring effect of user actions [3]; Persistance may be just during the *session*, journaled for later *state recovery* (Massive-3 [4]) or *continuous* (SIMNET [5]).

ii) **Scalable topology:** the scene sharing scheme among participants [6], such as

   – *Homogeneous replication*, independent replicas broadcasting changes (SIMNET, DIVE [7]);

- *Shared-centralized on a server*, one scene shared by all, managed at a central server (CAVERN [8], NPSNET-V [9]);
- *Shared-distributed with client/server groups*, in which clients are connected to the nearest server (DIVE, Massive-3, Octopus [10], VELVET [11]);
- *Shared-distributed using P2P actualization*, peer-to-peer connections among clients, comes in two flavors:

  P2Pr, replicating the same scene graph at each node (DOI [12]).

  P2Ps, shared distributed scene graph [13] with remote objects (Diverse [14], GNU/Maverik [15])

iii) **Network transmission:** appropriate protocol, UDP or TCP/IP; broadcast (SIMNET), unicast (most systems) or multicast addresses (CAVERN, DIVE, Diverse, DOI, GNU/Maverik, Massive-3, NPSNET-V, Octopus, VELVET); network *latency* issues [16] [17].

iv) **Collaborative user interaction features:** the collaborative set of manipulation and visualization interfaces, teleconference capabilities (chat, video and audio), flexible support for model construction, synchronous and asynchronous collaboration modes, adaptive multiresolution strategies, interoperability standards, and virtual space shared utilization.

Crucial features for CRVEs are: action indicators for remote event notification, alternate views, selectable avatars, and expected low latency response times.

v) **Object granularity:** determines the network broadcasting cost of scene changes [18], as

- *Light objects*, short state messages for event and control information (all systems).
- *Remote references*, network references to remote objects (All but SIMNET).
- *Heavy objects*, medium-atomic objects, able to fit in the client's memory, e.g. object 3D geometry (all systems).
- *Real-time streams*, large-segmented data to be transmitted in pieces or continuously, e.g. volumes, textures, video (CAVERN, Massive-3).

From the above, it is evident that most CVREs use unicast or multicast protocols for UDP or TCP/IP communications. The most recent environments tend to P2P or small client/server groupings topologies, with replicated or shared scene graphs. Only CAVERN and Massive-3 fully integrate large segmented data such as video feeds, while some of the others resort to variable multiresolution schemes or out-of-core segmentation.

Massive-3 is the lone provider of a journaling mechanism for interaction recovery. Avatars are a common feature, but none allow multiple perspectives. Only DIVE, Massive-3, VELVET, GNU/Maverik seem capable of handling large user loads or huge data models.

As far as the former reviews show, there is no clear strategy allowing an orderly and easy migration path from standalone VR applications to collaborative ones.

2.1 Collaborative Interaction Models

A special attention must be provided to interaction issues in distributed setups. There are two widely used conceptual paradigms in the design of user interfaces: the Model-View-Controller paradigm, known as MVC [19], and the Abstraction-Link-View paradigm or ALV [20], shown in Figure 1. MVC is the classical model for user interface design, factoring all application objects in three categories according to their functional roles:

Model	objects residing in an algorithmic layer.
View	objects located in a visualization layer.
Controller	objects, user interface widgets' layer that translates interaction into actions.

Communication among layers is achieved by an internal messaging system that feeds user actions into an switchboard event loop with dispatcher. Callbacks connect each switch hook with the corresponding widget object(s), which in turn effect changes in the domain.

In the Abstraction-Link-View (ALV) paradigm, objects are factored in abstraction, view and link layers.

Abstraction	objects are models shared by all users.
View	objects handle user interaction and visual rendering
Link	objects are constraint sets synchronizing abstraction and view objects.

The ALV's Abstraction layer is equivalent to the MVC's Model layer, while the ALV's View component layer merges

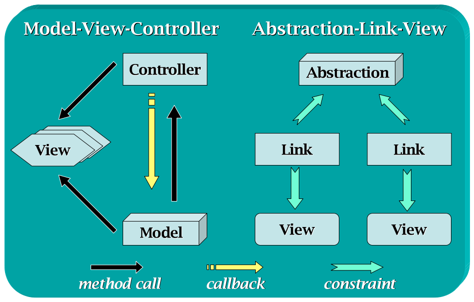

Figure 1: Side-by-side correspondence between the MVC and ALV collaborative user interaction paradigms

both View and Controller layers of MVC. The ALV's links connect abstractions and their views, using references to remote objects. Consistency in ALV is kept tracking local state changes at a central repository, while the MVC's Controllers handle communications among all its objects.

For CRVEs, the decoupled MVC approach proves insufficient because it does not provide for a common persistence layer to hold the shared state properties of remote interactions. The ALV model does provide a method for keeping track of object and session changes, but it is heavily slanted toward a client-server distributed model. In subsection 3.3 we propose a more suitable session management model to render reliable object flows at high speed rates.

3 Collaboration Framework Architecture

Many virtual reality applications begin as scene and object visualization environments, having special user interface metaphors for navigation and manipulation, and shown on display devices ranging from CRTs to immersive stereo projection systems. Most science disciplines (and the entertainment industry) use VR techniques to enhance user experiences. As research shows, users *always* desire to share these virtual experiences, either by showing models to prospective audiences, or by having an active remote participation in the environment.

Evolving collaboration at this stage usually entails the redesign and development of a (new) application, inserting a networking infrastructure under the environment, and other software-porting problems. Issues such as synchrony overheads, concurrent user load and system lags may degrade interaction and adversely affect graphics performance. There are generic API libraries for implementing shared scene graphs [21] that could be used for building multithreaded CVREs. The rationale behind our approach is that the object-oriented nature of current standalone VR applications, usually having rendering and user-interface components, would facilitate their transformation into complete CVRE's, by allowing the seamless attachment of a network-based component to enable collaboration.

In the following subsections we describe the collaborative features for the proposed superset framework. Given that the different VR tools may spread across platforms and support varied output display systems, the ideal solution should not compromise current designs or imply extensive recoding of components when fitting the collaborative framework. Massive or large-scale implementations were discarded due to user administration performance considerations, although the proposed framework has scaled well for a reasonable number of (less than twenty) participants.

Based on the features described in Section 2, our solution involves the implementation of a *multithreaded software components* architecture, a scalable *P2P sharing topology*, a layer implementing *session awareness capabilities*, and a flexible *crossplatform library for network transmission*. We have left for a future implementation the treatment of real-time streaming, since the framework does not modify the current *object granularity* of the target application. On the practical side, it is a portable generic framework, requiring only the instantiation of a custom message interpreting class for the shared session.

3.1 Multithreaded Software Components

We asume that a good VR tool is the final product of a sound systems design, developed under a classical MVC paradigm. A standard software engineering practice in Computer Graphics is the factoring of application objects into at least two weakly cohesive software functional components, graphics rendering and user interface. We decouple the Graphics Rendering (GR) and User Interface (UI) parts and instantiate them in separate threads. The same approach is taken with the new network communications component (NC), launched in its own separate concurrent thread. In this way, advantage is taken of the underlying operating system's context switching, loading the new software components without altering functioning code. This extensible approach allows the addition of more component threads, such as one dedicated to tracker data acquisition or interaction with haptic devices.

A snapshot of a working framework model is shown in Figure 2, detailing each software component. The NC component thread handles communications and message parsing; the top Shared Session (SS) management layer (see the MVCS model in subsection 3.3) launches all concurrent threads, tracks users' avatars, propagate state changes to the UI and GR components using callbacks, and is in general responsible for the emerging collaborative behavior; the GR and UI components are mostly untouched except for the connecting ”glue” to the Shared Session layer.

This setup is implemented by means of an abstract

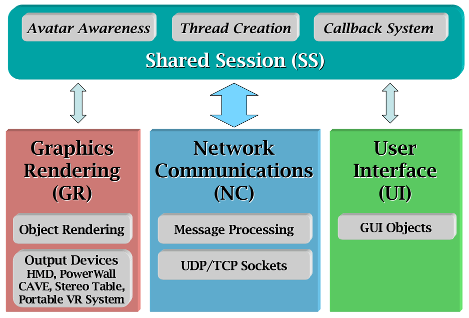

Figure 2: Collaboration-enabling Threaded Processes. The framework includes original components (GR and UI), and adds a session layer (SS) with the network component (NC)

class wrapper incorporating network awareness and a corresponding message protocol. An appropriate set of mutexes avoid shared state inconsistencies and race conditions when updating information.

3.2 P2P Sharing Topology

Fitting any of the client/server topologies would have implied the creation of at least one central server from scratch and compromised the applications' standalone behavior. We chose instead a peer-to-peer scalable topology, the most adequate for equal participants with separate access to their models. There are two possible topologies available in the framework: P2Pr [*Peer-to-Peer with scene replication*] and P2Ps [*Peer-to-Peer with scene sharing*].

In a P2Pr topology, each client has its own local replica of the scene. Since only a few scene objects are modified in the session, collaboration starts as soon as all clients have loaded their common model, and situated themselves within it. If there are no other participants in the environment, it defaults naturally to the standalone behavior.

A P2Ps topology must first build a shared scene graph, with each individual client adding whole chunks. For a particular client, scene graph objects are labeled *local* or *remote* depending on whether they are cached internally or need to be fetched elsewhere. If a client fails, its part of the shared scene must be reconstructed by the others.

Having no central server, both approaches require a third party application providing locating services for clients willing to enter in a session.

Thin broker for session administration
In our proposal, this third party is called a *message broker*, tracking session participation and interaction, as seen in Figure 3. It is based on some CORBA [22] facilities, but without the associated overhead. Shared state information is kept through the following services:

- A name service for location and client registration.
- A session management service.
- A session and client state report and mirroring service.

Given that the broker is not a bridge, client messages go directly to their destiny. Each client keeps track of other participants, and periodically may send its current state to the broker for shared session recording purposes.

3.3 Session Awareness Management

After analyzing the desirable characteristics exhibited by existing CRVEs, we concluded that a minimal collaboration feature set should include the following:

- Collaborative user interface model.
- Session administration with differentiated user roles.
- Client awareness using *avatars*.

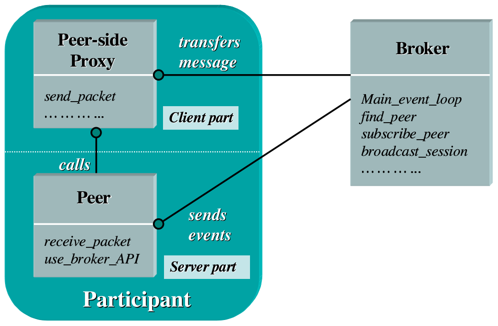

Figure 3: Peer-to-peer Broker Class Model. Both peers and broker have local thin instances of each other (proxies).

- Shared annotation and 3D marker highlighting.
- External real-time verbal communication channel.

For a client in this scenario, there must be perceptual evidence that other entities (human or otherwise) are participating, so 3D client embodiments within the environment (avatars) are used to dynamically reflect their position and state in the scene. Clients may want to call others to attention by placing special 3D signals, leaving trails in the scene or modifying the environment. Some users could just browse through the model, while others could have object editing privileges. A collaborative interface metaphor allows the remote manipulation of objects, and a session task can keep a journaled record of the interaction's history.

Collaborative user interface model
The problem to solve when recasting existing VR navigators as CVREs is how to implement the maximal collaborative feature set with the least possible implementation cost, and without affecting the original standalone behavior. We pick from each category of Section 2 the items that better support awareness under a hybrid Model-View-Controller-Session (MVCS) approach, tying the ALV's *links* as network pipelines to MVC objects, in which:

- MVC objects may not all reside together at the same network node, having their Model (structure and behavior) defined at one client, many different Views elsewhere (renderings, at least one for each client), and flow control effected by all. Nodes may have several visualization layers (cameras), allowing for multiple perspectives and resolutions of the same scene.

- Controllers operate on both using a callback mechanism, routing to the corresponding network nodes for non-local objects, as shown on Figure 4. Session layer coherence is maintained by existing network-aware controllers at each node, who also notify the broker. It does not matter whether objects are shared or replicated, so it allows either P2Pr or P2Ps approaches.

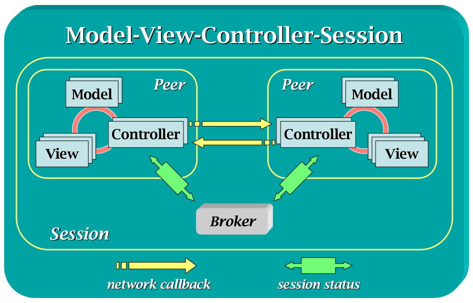

Figure 4: Model-View-Controller-Session (MVCS) Objects showing an external broker maintaining session states

Session administration with differentiated session roles
We have identified five different collaborative user behaviors: *standalone, peer, incognito, slave*, and *master*. A *standalone* client is not aware of other clients. It defaults to the original isolated behavior of the application. *Peers* are clients that communicate among themselves using the common message protocol. Users traveling *incognito* may observe scene interaction in "voyeur" mode without other clients knowing it. A *slave* is a peer that is bound to another, correspondly called a *master*, in the sense that the *master*'s current state is continuously replicated by the remote *slave(s)*. A self-explanatory three bit code catalogues their functional role (from left-to-right): *bit* 2 means whether the client broadcasts its messages to others, *bit* 1 whether the client listens to remote messages, and *bit* 0 whether there is a special binding between clients. Thus, [*standalone* (000), *peer* (110), *incognito* (010), *slave* (011), *master* (111)], leaving open the possibility of adding more. These client roles are voluntary and changeable during a session.

Client awareness using *avatars*
Each client has its own 3D representation traversing the environment, having several active camera perspectives at any time. Avatars broadcast a number of state attributes, such as *position*, *orientation* and *velocity* camera vectors for dead reckoning calculations.

Shared annotation and 3D marker highlighting
Users must not only be aware of each other, they must be able to call the attention of remote participants to some feature or object in the environment. This is accomplished by temporal 3D markers such as arrows, billboards or banners, objects that a "guide" pins at some interesting locations.

External real-time verbal communication channel
Collaborative environments use at least one real-time communication channel to allow the human users behind the workstations to exchange impressions about the virtual experience. The framework does not provide this service, but external suitable cross-platform alternatives such as Gaim, Gnomeeting and others have been used with equivalent ease.

3.4 Crossplatform Network Transmission

Since communication is what enables collaboration, the new software component handles network communication capabilities. This is done by a cross-platform networking class that allows either datagram-oriented (UDP) or connection-oriented (TCP) communications under IPv4 and IPv6 networks. The NC thread, under a common message protocol implements the following basic kind of services, each one running on its own separate listening socket:

- Shared event pipeline for sending environment state changes and callback messages
- Continuous streaming of some client properties, such as camera position and orientation
- A notifying service for the *Broker*.

When a client reports to the broker, it posts its network address and listening ports. A configurable setup accounts for external firewalling rules, allowing several clients to run concurrently on the same machine by choosing unique port numbers. This enhances performance tests, because it permits the simulation of heavier client loads independently of available workstations. Network traffic is generated only for broker requests, for position or orientation changes, and for shared callbacks (such as object manipulation).

System synchronization
The framework avoids hosting a central time server by keeping *relative* time differences for every peer-to-peer connection at the client's side. The local event time or *timestamp* is included in each network message. Clients at the other end may process incoming messages as either

Immediate:	messages are processed at once, or
Buffered:	messages are queued by timestamp.

When using the first approach, network latencies may produce jumpy updates and short temporal inconsistencies. The second is more suitable for replaying events in exact time sequence, at the expense of bigger time delays.

4 The ALICE Virtual Reality Navigator

The ALICE VR Real Time Inspector and Navigator [23] is a standalone VR software platform for the real time inspection and navigation of very complex virtual models, developed at the Universitat Politècnica de Catalunya. It has been used in a number of applications such as navigation in urban environments or interior ship design among others.

In order to allow the users of these applications to be able to navigate and inspect complex 3D models in several VR systems, ALICE offers the following features:

- Several modes for stereoscopic visualization: active stereo for our local CAVE; passive stereo for a less expensive system such as the MiniVR system [24]

- User position and orientation tracking: allowing implicit interaction by following input device movements, making the user feel that he is inspecting a real object instead of a virtual one.
- Use of varied interaction devices: mouse, joystick, VR gloves, and haptic devices.

In addition, ALICE stores objects and information such as textures in an hierarchical scene graph. It implements an extensible callback system for interactively working in highly complex scenes, using many advanced data structures and computer graphics algorithms for levels of detail handling, visibility culling and collision detection.

4.1 Framework validation in ALICE

The ALICE application is already factored into two software components, Graphics Rendering and User Interface. The User Interface component is provided by v3.x of Qt, an object-oriented user interface toolkit (using the MVC paradigm), with cross-platform deployability in MS Windows, Linux, several flavors of UNIX and MacOS X.

The decoupled callback hook system in ALICE connects user events to the graphics pipeline by means of a indexed command list. Each element of the lists stores a settable reference (the "hook") to some object's method (the "callback"). When an UI event triggers a particular command, its corresponding callback hook is executed with the provided event information and current environment state.

Given all the above, it was considered a suitable candidate for enhancing its collaboration features. Just changing some flags at the compiling phase allows the UI component to run in its own thread as needed. Next, the following steps were taken to fit ALICE into the framework:

1. Instantiate the shared session (SS) layer class, holding all common state awareness attributes, such as the scene graph, Avatars, remote references for the broker and the list of participants.
2. Choose the scalable topology (P2Pr, for this version).
3. Devise the *peer-to-peer* and *peer-to-broker* message protocols.
4. Instantiate the message parser class to process event messages, and place it in the networking communications (NC) component.
5. Wrap the GR, UI and NC software components as SS layer class attributes, and launch each of them in a separate thread.
6. Add one method call to provide a callback hook linking the message parser class in the NC component to the session-update method of the SS component.
7. Add one method call in the UI's main method to provide a callback hook to the SS layer.
8. Add one method call in the GC to provide the callback hook syncing the cameras and states of network peers just before rendering.
9. Instantiate the broker class, adding the necessary services.

A scalable P2Pr topology was chosen initially, given that all clients already function with local scene replicas, and it would not change much ALICE's behavior. In shared mode, the broker indicates the remote reference of the current scene, so hopefully everyone would be placed in the same model.

The message protocol is short and simple. There are three kinds of messages: *session*, *location* and *manipulation*. Session messages are the ones exchanged between the broker and the clients: connecting and disconnecting, reporting internet addresses and ports, number of active cameras, avatar appearance, global scene file, and other relevant data. Location messages are mostly for *avatar* properties being broadcasted among all participants. Manipulation messages (such as a local client touching, grabbing, adding or modifying an object) are sent to remote users by the callback system to maintain scene coherence among all participants. Out of the growing callback set of ALICE (around 100), only a subset of 14 affect issues as model integrity, shared scene state and object appearance, although more may be defined in the protocol.

Since all of this happens in the NC thread, mutexes are activated when this thread is modifying data such as clients' cameras. There is a corresponding set of mutexes placed just before rendering to avoid race conditions so common to concurrent programming.

The broker must be active for a session to be initiated by at least two subscribing participants. Each client may choose a session role (usually as a *peer*) and an avatar representation (from a menu), as shown in Figure 5, while keeping a list of the current active interactions with other users. As they navigate, clients can chat to each other, or place 3D markers on the scene to call attention to some feature.

Clients can also take the role of "voluntary slaves" for some other user, which now becomes a camera server. The *slave* shuts down its own cameras and reflects the *master*'s camera viewpoint and actions, the latter effectively taking possession of the slave's remote display devices. This feature may also be used to "teleport" a participant to the position of another, which is very useful to avoid losing virtual eye-contact among peers. Each node does independent renderings, which allows a client to show a wireframe representation while another fully renders the same scene.

A practical side arising of an implementation based on abstract wrapper classes, is that it is platform-agnostic and extensible, which makes it quite portable. Each application only needs to inherit from the message parsing class, add

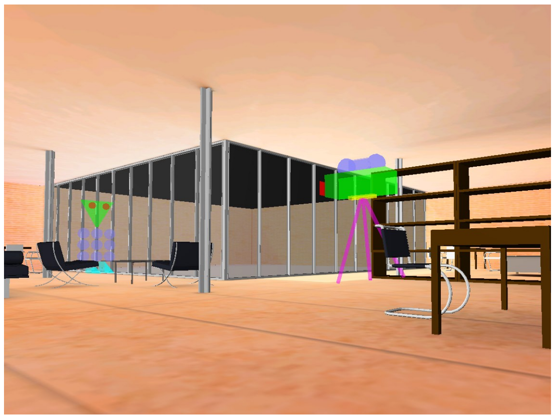

Figure 5: *Peers* (the “camera” and “upecito” avatars) collaborate on an inspection, as seen from an *incognito* client’s viewscene. Some videos of *peer* interaction may be seen at *http://crv1.lsi.upc.es/ vtheok/siacg04/*

its own protocol processing code and provide the hooks for the UI and GR components.

5 Performance Evaluation of the Architecture

We have tested ALICE’s remote collaboration and navigation services in the several VR systems in our lab, and also in sessions with the Girona University (located 100 Km. from the Barcelona campus) through a 10Mb wide area network connection. In our lab we have available HMDs, a stereoscopic table, a CAVE, a MiniVR system and flat displays; and a similar setup at the Girona campus. The results obtained from our tests can be seen in the following table. The scene used on these tests (the interior of a ship) contains 50.000 polygons, but on purpose does not have complex textures that could skew graphics performance. The table shows the results obtained in the communication of 2, 4 and 8 workstations using unicast addresses from both sites.

Participants	1	2	4	8
Av. Number of messages	–	2539	8067	14331
Av. Total Net. Time (msec)	–	35	31	46
Av. Roundtrip Time (msec)	–	13	46	57
Framerate	47.3	45.2	44.2	42.7

In the table we observe the average total number of messages sent through the network in a series of repeated navigation trials, each test lasting 4 minutes. The total network time (in milliseconds) gives information about how much time ALICE spent in the transmission of messages during these 4 minutes tests (this means that only around 0.1-0.2% of total time was spent in network communications). The roundtrip time is also indicated in milliseconds. Since for this test we use unicast addresses, roundtrip time increases as more peers participate in the session. Finally, the table shows the average rendering framerate achieved for each case, which indicates that increasing the number of nodes affects graphics performance very slightly compared to the standalone performance, and is comparable to similar setups in the studied environments.

As already stated in subsection 4.1, the migration of ALICE to a CVRE was fast and uneventful. Based on the fact that the application was already designed considering graphics rendering and user interface as separate components, its porting to our framework only required to define an adequate message protocol, connect the appropriate callback hooks, and add two method calls and corresponding code hooks in order to attach the application to the new network and session parts. Following the same migration scheme, it would be easy as well to transform any other VR application into a collaborative VR application. In fact we are presently porting another application built in our lab which addresses inspection and management of medical models.

Some fine-tuning must be performed to adjust threaded execution. Some highly textured models make take a while to render, making timely interaction slow and difficult. Although this can not be avoided, it may be reduced by changing thread priorities to model complexity and network traffic. Although we have not done experiments in slow networks, we simulated a fictitious one and we found out that sequential processing of arriving avatar information may cause clients to fall out-of-sync. In order to minimize these latency problems, there is an option to process only the most recently received information packet from each camera in the environment, at the expense of a somewhat jumpier navigation.

The proposed mechanism for camera management and sharing is reasonably easy to learn for users and seem to be adequate for collaboration tasks. We want to make some experiments with untrained users soon in order to have a more accurate perception of ease-of-use.

6 Conclusions and Future Work

Based on a characterization of generic collaborative features for VR systems, we have proposed a versatile framework for evolving collaborative capabilities in standalone VR navigators. Our approach incorporates a hybrid distributed user interaction model, multithreaded software components, network communications under a peer-to-peer scalable topology, message passing channels with a custom protocol, and changeable user roles in a multicamera subscription model.

The framework’s development has been validated by a fast porting of the ALICE VR Navigator. The generic cross-platform design allows an easy migration of similar VR applications into complete collaborative virtual reality environments. As for future work, we are working on extending the collaborative breadth of the framework by including in the Session layer a fourth thread for handling haptic devices, adding high frequency force-feedback events to the interactive session repertoire. Given the huge scene size of current VR scenes and objects, we plan to migrate applications towards a peer-to-peer with sharing scheme (P2Ps), and also to allow the incremental streaming of multiresolution objects to improve rendering performance and scalability.

References

[1] Schuckmann C., J. Schummer, and Seitz P. Modeling collaboration using shared objects. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work*, Phoenix, Arizona, United States, 1999.

[2] V. Theoktisto, M. Fairén, and I. Navazo. Enabling collaboration in virtual reality navigators. Technical Report LSI-04-13-R, Universitat Politècnica de Catalunya, Barcelona, Spain, 2004.

[3] J. Leigh, A. E. Johnson, and T. A. DeFanti. Issues in the design of a flexible distributed architecture for supporting persistence and interoperability in collaborative virtual environments. In *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing (CDROM)*, pages 1–14. ACM Press, 1997.

[4] C. Greenhalg, M. Flintham, J. Purbricc, and S. Benford. Application of temporal links: Recording and replaying virtual environments. In *Proceedings of the IEEE VR, 2002*, 2002.

[5] M. R. Stytz. Distributed virtual environments. *Computer Graphics and Applications*, 16(3):19–31, 1996.

[6] D. Duce, D. Giorgetti, C. Cooper, J. Gallop, K. Johnson, and C. Seelig. Reference models for distributed cooperative visualization. *Computer Graphics Forum*, 17(4):219–233, 1998.

[7] C. Carlsson and O. Hagsand. Dive: A platform for multi-user virtual environments. *Computers and Graphics*, 17(6):663–669, 1993. http://www.sics.se/dce/dive.html.

[8] J. Leigh, A. E. Johnson, and T.A. DeFanti. Cavern: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Journal of Virtual Reality Research, Development and Applications*, 2(2):217–237, 1996.

[9] M. Capps, D. McGregor, D. Brutzman, and M. Zyda. Npsnet-v: A new beginning for dynamically extensible virtual environments. *IEEE Computer Graphics and Applications*, 20(5):12–15, 2000.

[10] P. Harting, C. Just, and C. Cruz-Neira. Distributed virtual reality using octopus. In *Proceedings of IEEE Virtual Reality 2001*, pages 53–62, March 2001.

[11] Jauvane C. de Oliveira and Nicolas D. Georganas. Velvet: An adaptive hybrid architecture for very large virtual environments. *Presence*, 16(6):555–580, December 2003.

[12] G. Hesina, D. Schmalstieg, A. Fuhrmann, and W. Purgathofer. Distributed open inventor: A practical approach to distributed 3d graphics. In *Proceedings of ACM VRST'99*, pages 74–80, 1999.

[13] B. Zeleznik, L. Holden, M. Capps, H. Abrams, and T. Miller. Scene-graph-as-bus: Collaboration between heterogeneous stand-alone 3-d graphical applications. In M. Gross and F.R.A. Hopgood (Guest Editors), editors, *EUROGRAPHICS 2000*, volume 19, 2000.

[14] J. Kelso, S. G. Satterfield, L. E. Arsenault, P. M. Ketchan, and R. D. Kriz. Diverse: a framework for building extensible and reconfigurable device-independent virtual environments and distributed asynchronous simulations. *Presence: Teleoper. Virtual Environ.*, 12(1):19–36, 2003.

[15] R. Hubbold, J. Cook, M. Keates, S. Gibson, T. Howard, A. Murta, A. West, and S. Pettifer. Gnu/maverik: A microkernel for large-scale virtual environments. In *Proceedings of the ACM Symposium on VR Software and Technology*, pages 66–73. ACM Press, 1999.

[16] Martin Mauve. How to keep a dead man from shooting. In *Lecture Notes in Computer Science*, volume 1905. Springer-Verlag Heidelberg, 2000.

[17] M. Meister and C. A. Wüthrich. On synchronized simulation in a distributed virtual environment. In V. Skala, editor, *WSCG 2001 Conference Proceedings*, 2001.

[18] D. Brutzman, M. Zyda, K. Watsen, and Mike Macedonia. Virtual reality transfer protocol (vrtp) design rationale. In *Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE): Sharing a Distributed Virtual Reality*, Massachusetts Institute of Technology, Cambridge Massachusetts, June 1997.

[19] G. E. Krasner and T. Stephen. Pope, a cookbook for using the model-view controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[20] R. D. Hill. The abstraction-link-view paradigm: Using constraints to connect user interfaces to applications. In *Proceedings of the SIGCHI Conference on Human Factors and Computing Systems*, pages 335–342, Monterey, California, United States, 1992.

[21] G. Voss, J. Behr, D. Reiners, and M. Roth. A multi-thread safe foundation for scenegraphs and its extension to clusters. In *4th Eurographics Workshop on Parallel Graphics and Visualization, EGPGV02*, Blaubeuren, Germany, 2002.

[22] F. Deriggi, M. Kubo, A. Sementille, J. Ferreira, S. dos Santos, and C. Kirner. Corba platform as support for distributed virtual environments. In *Proceedings of IEEE, Virtual Reality'99*, Houston, Texas, March 1999.

[23] C. Andújar, M. Fairén, and P. Brunet. Affordable projection system for 3d interaction. In *1st Ibero-American Symposium in Computer Graphics*. University of Minho, Portugal, July 2002.

[24] M. Fairén, P. Brunet, and T. Techmann. Minivr: A portable virtual reality system. *Computers & Graphics*, 28(2), April 2004.