DESIGN AND IMPLEMENTATION OF A FRAMEWORK FOR IMMERSIVE ENVIRONMENTS IN A SHARED CONTEXT

Yves Boussemart

Department of Electrical and Computer Engineering

McGill University, Montréal

October 2004

A thesis submitted to McGill University in partial

fulfilment of the requirements of the degree of

Master of Engineering

Your file Votre référence
ISBN: 0-494-12588-8
Our file Notre référence
ISBN: 0-494-12588-8

À mes parents.

Abstract

This thesis presents the framework used in our research group – the Shared Reality Environment Lab – for use in our spatially immersive setups. Our main interest is the application of virtual reality techniques in the context of computer-mediated human to human communication and collaboration. The purpose of this architecture is to allow users to quickly develop remote collaborative applications without the need to manage low level operations. The framework is comprised of multiple independent components: the Qave graphics engine, user trackers, sound spatializer, coherence server, communication library and overlapping projectors interface. Some components were already implemented when this work started and the contributions of this thesis are the graphics engine, the coherence server as well as the integration of the other components in that framework.

Résumé

Cette thèse présente l'infrastructure mise en place spécifiquement pour le "Shared Reality Environment Lab". Cette dernière a pour vocation de gérer les environnements immersifs de notre laboratoire. L'intérêt principal de notre recherche est d'exploiter les possibilités offertes par les univers de synthèse dans un contexte de communication et de collaboration entre individus. Le but de l'architecture présentée ici est de permettre à des développeurs de créer des applications collaboratives pour les espaces virtuels, sans avoir à se préoccuper des tâches sous-jacentes. Cette infrastructure en tant que telle est composée de différents modules indépendants: le moteur graphique Qave, les processus de suivi des utilisateurs, le module de spatialisation sonore, le serveur de cohérence, la librairie de communication, et finalement le système de projections redondantes. Certains de ces modules étaient déjà en place au début de ce travail, nous y avons contribué par l'ajout du moteur graphique, du serveur de cohérence et par l'intégration des différents modules dans l'infrastructure décrite.

Acknowledgments

First and foremost, I want to thank my parents for supporting me throughout my studies, and for giving me the opportunity to see all that I've seen and learnt by traveling in so many different countries. Secondly, I want to thank my supervisor, Prof. Jeremy R. Cooperstock for his time and effort involved in helping me with this Master's thesis. I also gratefully acknowledge his financial support. I also want to sincerely extend my appreciation to Prof. Marcelo Wanderley for his counseling and for all the stimulating discussions we had. Of course, I have to thank all the member of the SRE Lab, past and present, for their invaluable camaraderie, insights and friendship. I want to express special gratitude to Frank, both Mikes (Woz and Perez) and Nadia for the (very) ungrateful work of proof-reading my thesis and helping me fight against the English grammar. Finally, I want to thank my closest friends on this continent, Seb, Darius, Le, Kristin and so many others for always reminding me that there is so much to learn outside of school. Special thoughts go to my friends from France, Belgium and Denmark, we'll meet up sooner or later.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

"I hear, I forget; I see, I remember; I do, I understand."
—P.R. Halmos

1.1. Research Problem

The concept of virtual reality (VR) is compelling because it marks a radical shift from the viewing paradigm used in conventional computer graphics. Instead of viewing an application through a 2D porthole, such as a screen, the user of a VR system becomes immersed in a virtual space through a combination of software and hardware, effectively transcending the boundary between the real and virtual world. In effect, this allows the user to "go through the looking glass" and discover another reality.

VR is a field that has been maturing rapidly over the past few years, as the increase in computing power and the advance in display technology have made it much more accessible not only to the academic world but also to industry and the arts. The primary objective of the Shared Reality Environment Lab (SRE Lab) at McGill University is to facilitate human-human communication and collaboration through the use of VR techniques. A prototype three-walled projection-based spatially immersive environment was constructed for that purpose.

A new project has recently begun in our research group, in which the goal is to offer a higher resolution display in a multiply overlapping front-projection environment. Being able to use two immersive spaces simultaneously is valuable because it allows the investigation of issues related both to network (e.g. low latency algorithms, dynamic video bandwidth adaptation) and collaboration (e.g. importance of gaze awareness or avatar realism). Furthermore, additional ongoing research in the SRE Lab pertains to user interfaces for such immersive setups. Our main goal is to provide an untethered, purely *walk in and use* environment. This is another important consideration for this work. For all these reasons, it is desirable to have a consistent framework that can be adapted to both our spaces.

The purpose of the framework is to support applications of remote collaboration between our two distinct research setups (see Section 1.4). Besides specific component-level design requirements, there were two main overarching guidelines we tried to follow. Firstly, the framework should allow the creation of applications that rely on an user interface that is as natural as possible. In that sense, the user should not have to don any special apparatus before being able to experience and interact with the environment. Secondly, the architecture should be flexible in the sense that specific mechanisms or algorithms are likely to change as the research effort progresses. The framework thus divides the different tasks in distinct modules. These modules communicate through a network layer thus providing a layer of abstraction between the internals of each component. For example, a tracking component may be upgraded with a newer detection algorithm, but as long as the communication interface (that is both message syntax and rate) is respected, the other components need not know about the change.

1.2. Contribution of the Thesis

The work presented here is based on the framework that was previously used in conjunction with CaveLib as the graphics renderer. Our main contributions were to provide a lightweight alternative to CaveLib, and extend the original framework capabilities by adding audio spatialization to the virtual scene along with the ability to share a common virtual world among multiple nodes. Providing a coherent represenation of the world between locations allows users to see the changes made to the world by another remote participant. Therefore, multiple distant users are able to work concurrently as if they were present in the same physical space.

Figure 1.1 shows an overview of the complete framework, which consists of multiple independent parts:

- Qave: The rendering engine, which runs the User Application.
- Trackers: Vision-based user tracking system.
- Sound Spatializer: Creates a 3D point source for the remote sound and simulates the acoustics of the scene.
- WorldServer: Maintains the world coherence between multiple Qaves, and allows for collaborative work.
- ServerLib: Network Library used for all message passing and video streaming between different Qaves.
- Multiply Overlapping Projection System: Takes the framebuffer of a Qave and splits it over multiple computers while providing for dynamic shadow detection and removal.

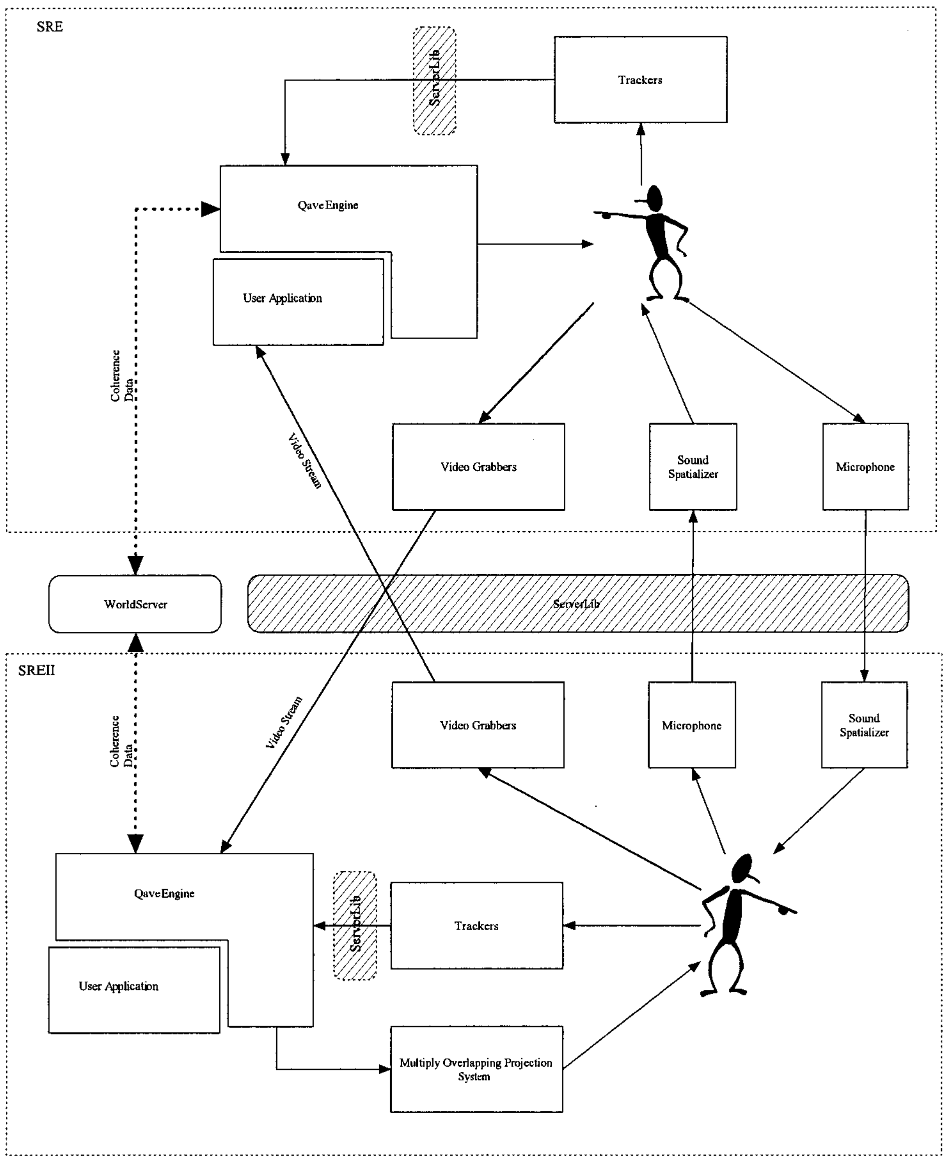

FIGURE 1.1. Framework Overview

As mentioned, parts of the infrastructure were already in place before the design started, namely the Trackers,[1] ServerLib[2] and sound spatializer.[3] The Multiply Overlapping Projection System is currently still in development.[4]

The contributions of this thesis are:

[1]The trackers were developed mainly by Stéphane Pelletier and currently being updated by members of our research group

[2]ServerLib was developed by J. R. Cooperstock

[3]The Sound Spatializer was developed by Zack Settel, http://www.zeep.com

[4]The Multiply Overlapping Projection System is supported by Daniel Sud and Maria Nadia Hilario

- The Qave rendering engine, which manages the displays
- The WorldServer, which allows for collaboration to take place
- The modification and adaptation of the Trackers for use in this framework, which provide the engine with user data
- The adaptation of the Sound Spatializer, which provides an immersive auditory space
- The communication protocols between these modules.

1.3. Design Goals

The central component of this thesis is the Qave, i.e. the graphics engine. Based on the specific setups available and the global research interests of the SRE Lab, a list of design goals was established. These goals will be referred to as (DG) in the rest of this thesis.

(DG1) **Unix Compatibility:** Both our environments have different display setups and computing facilities, which will likely be upgraded in the future. In terms of operating systems, the engine must be able to run on at least Irix and Linux.

(DG2) **Immersive:** The engine must be able to make use of the three-screened spatially immersive display of our original environment, as well as the large screen display in our new setup.

(DG3) **Dynamic Perspective Correction:** Compensating for the users' position in the environment so as to constantly give them a correct perspective of the synthetic world. The latency between the user movement and the perspective update should be minimal.

(DG4) **Simulator Mode:** The user application can be tested without needing to run on the actual immersive hardware. In that mode, the Qave should not rely on tracker data, and thus should have an alternate mode of input that can emulate external sensors.

(DG5) **Tracker Information:** The ability to receive and process data from the Trackers. The data stream usually comes through the network as ServerLib messages. There should be as small a latency as possible between the time a message arrives and the time the displayed world updates accordingly.

(DG6) **Coherence Data:** Receiving and sending data to other Qaves through the WorldServer in order to maintain a coherent view of the world. More importantly, this allows for collaborative work to take place between Qave users.

(DG7) **Remote Video:** Embedding a remote video stream in the virtual scene. Video input usually comes from a framegrabber as a ServerLib binary stream.

(DG8) **Extensible:** Easily modifiable by other researchers. Easy access to the native OpenGL layer, and thus providing flexibility for future development.

As for the WorldServer, the main design goals were:

(DG9) **Coherence:** Hold a unique coherent representation of the world shared by clients.

(DG10) **Data Propagation Protocol:** Messages exchanged between the clients should be in human readable format for easy auditing purposes. While this somewhat limits the potential for optimizations, the bandwidth used should be kept as low as possible by avoiding to transmit unnecessary information.

In summary, this thesis presents a general and modular VR framework consisting of multiple components: an immersive graphical engine, coherence server, and trackers, allowing for interaction and audio spatialization.

1.4. Research Setups

Our original setup, the SRE, is a spatially immersive environment (IE) which uses the three sides of a 1.82m[5] cube (see Figure 1.2) as display surfaces.

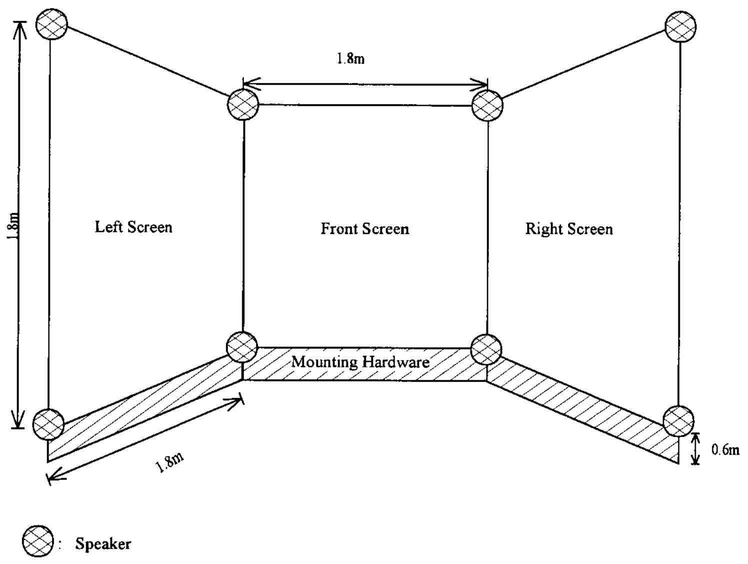

FIGURE 1.2. SRE Setup Physical Dimensions

In that prototype space, three XGA projectors are used for monoscopic rear-projection, providing a total resolution of 3072x768 pixels. Our choice of monoscopy over stereoscopy was motivated by our design goal of a *walk in and use* system, which precludes extraneous gear such as stereoscopic goggles or data gloves. This choice is also motivated by the fact that head-coupling is more effective than stereopis for the 3D perception of the scene, as shown by Ware et al. [**1**]. The hardware renderer is a Silicon Graphics Onyx2.

The prototype SRE is equipped with eight speakers located at each corner of the cube. Multiple sound sources can be spatialized to create the illusion that they originate from arbitrary locations in 3D space. This allows sound streams to be localized

[5]The exact length is 6 feet.

consistently with the screen position of the distant participants. For example, remote users' voices can be heard moving from left to right and up and down as they move accordingly. Additionally, the spatializer can emulate the acoustic characteristics of different rooms such as a highly reverberant church or an open field, thereby providing a more effective immersion in the synthetic environment.

In our second instantiation of the SRE, our setup involves creating a large display surface composed of the entire 7x2m back wall of that room, as well half of the adjacent side walls, where six SXGA projectors render a coherent seamless image. This requires the geometric calibration of each projector and intensity blending in overlapping regions. However, a user standing between a projector and the wall may create a shadow on the display. These shadows can be automatically detected [**2**]. Occlusions can then be removed by selectively increasing the intensity of other projectors covering that region. The adopted approach,[6] is similar to that of the Metaverse [**3**]. This environment also employs eight speakers (four along the top of the front wall and another four along the top of the back wall), allowing for bi-directional spatialized communication between our two setups.

1.5. Sample Applications

An initial sample application was implemented and is used throughout this thesis to illustrate the rendering engine and the overall framework capabilities. This application is a computer-aided, immersive collaborative tool for object manipulation. As such, it allows the users to jointly move and rotate objects in a virtual scene.

The user's hand is represented by a synthetic hand-model (see Figure 1.3). This improves the on-screen feedback and the degree of engagement of the user. Although an arrow or a crosshair with greater precision could have been used instead, allowing

[6]This work is conducted by Daniel Sud and Maria Nadia Hilario.

users to see on the screen a synthetic hand directly controlled by their real hands increases the degree of immersion. Poupyrev et al. have evaluated those two interaction paradigms, and have noted that displaying a virtual hand more closely simulates real world interactions, thereby increasing the familiarity of the interface for the user [4]. There is a direct sense of embodiment (or self-representation) as users realize that the movements of the virtual hand on the screen are analogous to those of *their* hands.

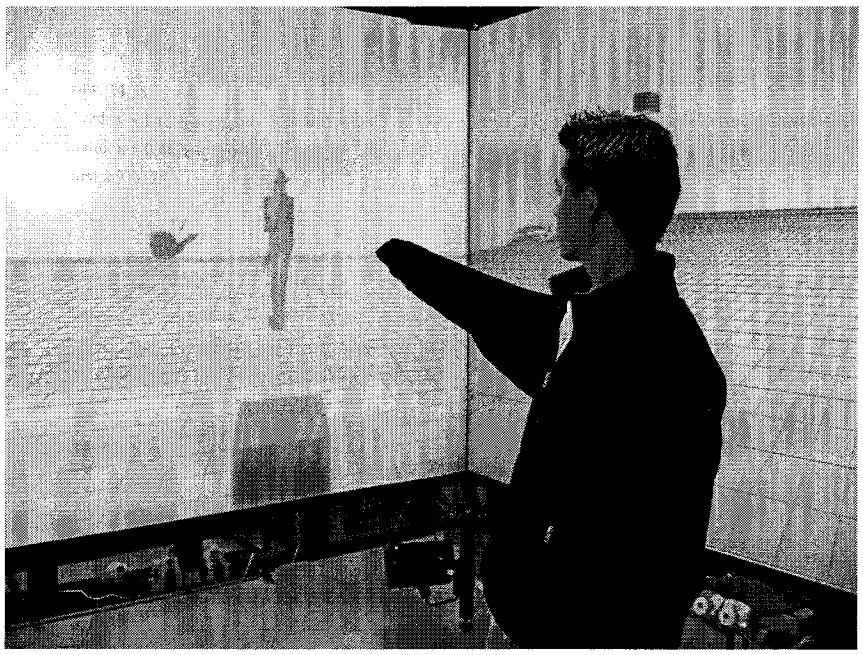

FIGURE 1.3. Qave Rendering with User's Hand

This application was chosen as it clearly illustrates the different functions of all the components of the Qave framework. Moreover, the interaction paradigm of object selection and manipulation is sufficiently generic that it can be adapted to numerous applications in other fields. Further, the possibility of bringing a remote participant into the synthetic world allows for collaborative work to take place.

Rioux et al. have implemented a more sophisticated application [**5, 6**] using the proposed framework. A bimanual interface for object manipulation in immersive spaces using *pieglass* widgets was developed.

1.6. Thesis Overview

The next chapter presents background information on virtual reality as well as an overview of significant work in that field. Chapter 3 describes in detail the design and implementation of the Qave rendering engine. Chapter 4 discusses the WorldServer application that allows for different Qaves to render the same consistent world. Chapter 5 presents the modifications made to the trackers that were needed for integration in the framework, along with an overview of the sound spatializer structure and the communication protocol adopted between the trackers and the spatializer. Finally, Chapter 6 concludes this thesis and highlights some possible research directions for the future.

CHAPTER 2

Literature Review

"The Guide is definitive. Reality is frequently inaccurate."
—Douglas Adams

Much progress has been achieved recently in the field of virtual reality. In this chapter, we first provide background information on that field (Section 2.1), and then present some of the most relevant previous works. Because the framework described in this thesis is aimed to be used with spatially immersive environments, we partition previous VR projects by display paradigms into three categories: desktop-based VR (Section 2.2), HMD-based VR (Section 2.3) and projection-based VR (Section 2.4), with a special emphasis on the latter.

2.1. Virtual Reality Background

2.1.1. Definition. The term "virtual reality" (VR) was introduced by Ivan Sutherland in the 1960s [**7**, **8**]. Sutherland's idea was for users to wear a device on their head that would place a small screen in front of each eye. The first prototype, built in 1968 and dubbed "Sword of Damocles," was the first head-mounted display (HMD) ever built (see Figure 2.1). It consisted of two CRTs mounted alongside each ear to generate monoscopic wire-frame images displayed on a pair of half-silvered mirrors directly in front of the user's eyes. The system was suspended from the ceiling (hence its name) by a mechanical arm that had two functions: to support the weight of the displays and compute the user's gaze direction. The image displayed on the screens was updated according to the user's gaze and the application running on the system.

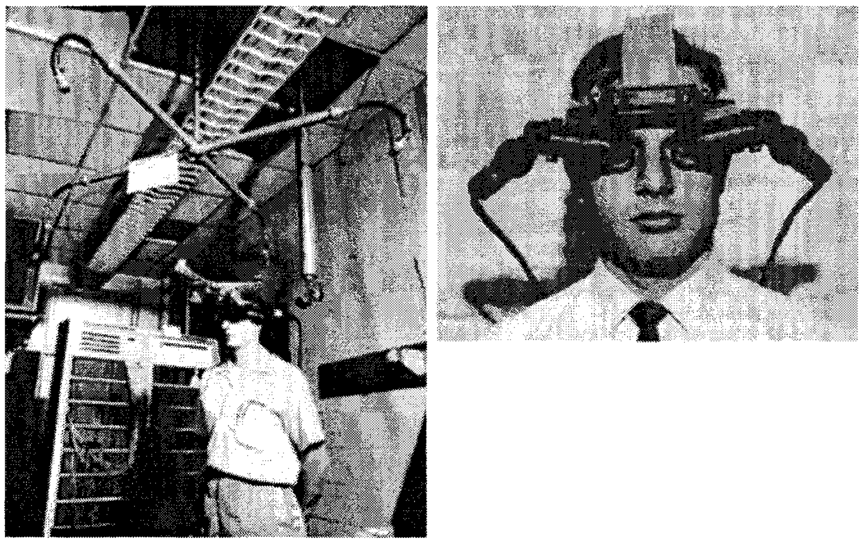

FIGURE 2.1. Sword of Damocles

Thirty-five years later, the term *VR* is broadly used to encompass a range of technologies, from HMDs to single screen 3D stereo imaging or to spatially immersive environments. The definitions of the term "virtual reality" vary greatly from author to author, but all usually share at least five common features:

(1) **Interactivity.** An interactive system reacts to input from the user. In other words, the VR system must be user-centered. This level of interaction allows the user to feel transparently connected to the environment in the sense that the environment responds directly to its stimuli. In our case, the study of interaction involves Human Computer Interactions (HCI) in an immersive context.

(2) **Immersion.** VR applications must be perceptually immersive, in the sense that the system must convey sensory cues to users that they are *surrounded* by the application.

(3) **Engagement.** The degree to which the user feels involved in the environment is its engagement. For most applications, being able to engage the user is the most difficult aspect of the problem, since the system is only as engaging as it is convincing. To be convincing, the system must lead the user to the point of *suspension of disbelief*, where the user will accept the synthetic environment as real. Because of technological limitations,[1] most VR spaces do not currently reach that point.

(4) **Multi-Sensory.** To be engaging, a VR system needs to involve the users' senses with congruent stimuli. Ideally all five senses should be engaged, but due to technological limitations the ones usually involved are the visual, auditory and sometimes haptic.

(5) **Synthetic.** The computer system must synthesize a dynamic environment in real time.

More formally, a generally accepted definition of virtual reality is provided by Cruz-Neira [**9**]: "Virtual Reality refers to immersive, interactive, multi-sensory, viewer-centered, three-dimensional, computer-generated environment and the combination of technologies required to build these environments." As a side note, the term "virtual environment" is often used as a synonym for "virtual reality system," but presents the advantage of avoiding philosophical conundrums regarding the use of the concept of "reality."

[1]The main limitations are display technoogy and computer processing power.

2.1.2. Virtual Reality System Components. It is worthwhile to describe what composes a generic VR framework. Figure 2.2 presents an overview of the different components commonly used in a VR system.

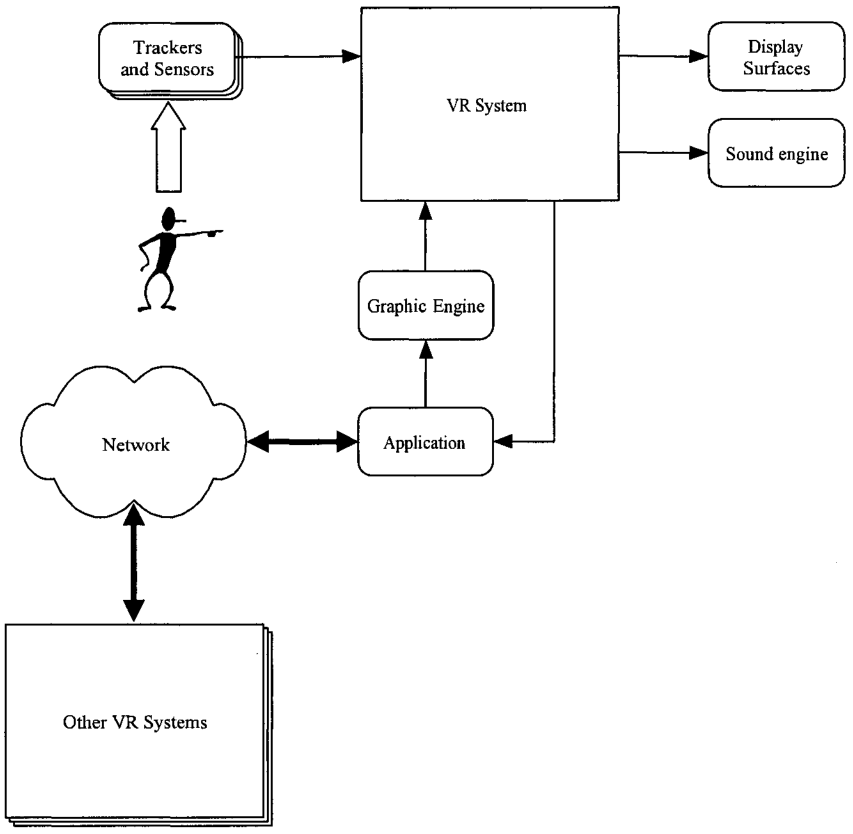

FIGURE 2.2. General Overview of a VR System

A VR environment relies both on software and hardware to immerse the user in a synthetic world. The hardware receives the data (typically user position and user input) from trackers and other sensors. It then transmits the data to the application which in turns feeds multi-modal stimuli back to the user, effectively conveying the feeling of being in a virtual world. Multiple VR systems may be linked through a network to render a coherent world for all the participants, thus allowing them to share their work and collaborate.

2.1.3. Applications in Research, Industry and Arts. VR systems are useful for the research community. In particular, researchers in the field of supercomputing are usually faced with tremendous amounts of data. The analysis of such solution sets on regular CRTs or LCDs is typically difficult because the researcher can only display and visualize a small portion of the information at a time. Spatially immersive environments are then useful because they provide larger display real-estate. Off-the-shelf LCD or CRT displays typically have a diagonal size of 60cm, i.e. a surface of 0.2m^2. Spatially immersive environments, on the other hand, normally use multiple projectors to covers areas as large as 12m^2. Available resolution for projectors is limited, so there is a trade-off between large real estate and pixel density. However, it has been suggested that the use of such large display surfaces allow for better data visualization and thus interpretation [**10**].

Today, academic research in VR is still dynamic and significant projects related to this thesis will be described in the next sections. VR systems have started to be used outside of academia and have pervaded into industry and art exhibits [**11, 12**].

For example, a field in industry where VR is starting to show great promise is automobile design. In the early days, designers and engineers would sketch proofs by hand before creating a full-scale model of a car. These full-scale models traditionally were made out of clay or plaster. Needless to say, making such large models was time consuming and expensive. However, it clearly was a necessary step as design flaws may remain undiscovered in drawings or small-scale models As designers and engineers started to use increasingly powerful computer assisted design (CAD) packages to speed up the design process, the hand-drawn sketches disappeared. Building a new full scale model for every design iteration, on the other hand, was still a necessity.

A major industry player using such VR systems is General Motors [**11**]. GM replaced the clay with spatially immersive projection systems. In these environments, the full-scale models can be rendered immediately from the CAD representation.

System users can then view the model in 3D, rotate it about all axes and modify the design on the fly. Ultimately, a full scale model must still be constructed, but all the major design decisions will have been based on a synthetic model of the car, hence saving time and money during the process. Furthermore, by networking similar systems together, designers from all over the world can collaboratively work on the car, while being physically separated by thousands of kilometers.

This feature is usually referred to as *telepresence*, defined as creating the illusion that remote participants are present in the physical space of the local user [**13**]. Goebbels and Lalioti [**14**] argue that immersive environments are successful at overcoming some of the problems of sharing and collaborating over distance. Projection-based systems in particular allow for more natural collaboration metaphors to be used in a recreated face-to-face communication. This is mainly due to three factors: shared physical context, screen size (hence displayed size of the remote user's body) and gaze awareness, the latter being critical for effective human-human communication [**15**].

The entertainment industry, always concerned with creating multisensory experience, was one of the first to use VR systems to provide a more stimulating experience to its clients. Museums, as cultural and educational institutions, have also started to benefit from the technological innovations available to the entertainment industries and academia. These venues, traditionally hesitant to use cutting edge computing tools, are starting to use VR to attract visitors. Recent such projects include the Hayden Planetarium's 400-seat system at the American Museum of Natural History[2] in New York and the spatially immersive environments installed at Ars Electronica in Austria[3] and the Foundation of the Hellenic World in Greece [**12**]. Although these are still costly, fixed, semi- or fully-immersive installations; scaled down versions of such

[2]http://www.amnh.org/rose/haydenplanetarium.html
[3]http://www.aec.at/

systems should be able to enter smaller museums or even schools in the foreseeable future.

2.2. Desktop-based Virtual Reality

Desktop-based environments are the most basic of VR systems as their display setup usually consists only of the computer monitor. Because such a display offers a restricted field of view, the visual interface is often enhanced by stereoscopic goggles, which increase the feeling of immersion but require some form of head tracking to provide the correct parallax information. Workstations for which the display is coupled with head-tracking (and sometimes hand-tracking) along with stereographic goggles are usually known as *fish tank* VR environments [**1**]. Another good example of such a desktop-based system is described by Deering [**16**]. The main advantage of such systems is availability, as they can usually be built from off-the-shelf components at a reasonable cost. Conversely, desktop-based VR suffers from a low degree of immersion due to its limited field of view.

2.3. Head-Mounted Display for Virtual Reality

HMD devices place a pair of screens directly in front of each of the user's eyes. Usually, a helmet worn by the user supports the display and contains all the hardware required to run the screens. Because the displays cover all of their field of view, users can seemingly be removed from their surroundings, which can potentially be replaced by a completely synthetic world. If the user can see through the goggles, computer-generated information may be overlayed onto the physical world, in which case the HMD is used in an *augmented reality* (AR) context (see [**17**] for an example of AR and its applications).

Historically, HMDs were the precursor of all VR environments. Since the 1960s, considerable efforts have been devoted to improve HMD systems. Most of these efforts were undertaken by the military for flight simulation purposes [**18**] at a time when display technology was not capable of providing sufficient resolution and refresh rate. More recently, Butterworth et al. created a 3D surface modeling system that uses a HMD as display [**19**].

The general agreement is that the main advantage of HMDs lies in their large field of view and the resulting sense of immersion for the user. The trade-offs are that HMDs are highly invasive because the displays must be located as close to the eyes as possible, and that their use can cause motion sickness. There are two main causes for such simulator sickness: update latency and non-coherence of somatosensory senses of body movement. An example of the latter is when users move in the virtual world while their bodies stay stationary. A solution for this problem would be to have the users walk on a treadmill-like device, but technical issues[4] make it less than ideal. Paush et al. have shown that the update latency between head movements and synthetic world changes (also known as end-to-end latency) causes a sensory conflict because the visual information is not consistent with vestibular information [**20**]. The issue of lag can be alleviated by using predictive Kalman filters to anticipate the head movements, thereby reducing user errors by an order of magnitude [**21, 22**]. Similarly, Richard et al. have studied the effect of display update rate on the manipulation of virtual objects, and it has been found that for reasonable interaction the minimum rate is 14Hz [**23**]. Even though the advances in the technology tend to alleviate these problems, HMDs are usually limited by lower resolution and the weight of the helmet that must be worn by the user. For instance, a usual SVGA (800x600) HMD usually weighs about 200g,[5] while an SXGA (1280x1024) one usually weighs about 1kg.[6]
As of late, fewer projects use HMDs as it has been shown that large displays are usually less invasive and more cost-effective immersive systems [**24, 25**].

[4]The main issues are noise from the mechanical parts and limitation in displacement speed.
[5]http://www.vrealities.com/5dt.html
[6]http://vresources.jump-gate.com/articles/vre_articles/analyhmd/analysis.htm

2.4. Projection-Based Virtual Reality

The use of projectors to display the synthetic world has multiple advantages, the main one being the physical size of the display obtained at a reasonable cost without the need for invasive devices. The two possible configurations are single and multiple screens, the latter usually allowing for better spatial immersion.

2.4.1. Single Screen. The simplest projection-based VR system is an extension of the fish tank paradigm, where one uses a single large screen display to cover the whole field of view of the user (as opposed to using a smaller computer screen). Czernuszenko et al. have created two such setups, the Immersadesk and the Infinity Wall [**26**], which use head-tracking and stereoscopic goggles as in desktop-based VR. The Infinity Wall uses a fully vertical front-projected display; whereas the rear-projected screen of the Immersadesk is angled at 45 degrees from the horizontal so that users can look downward as well as forward. When the user stands close, this system covers a 110 degree horizontal field of view. The Immersadesk also features hand-tracking through the use of a magnetic sensor. The Responsive Workbench [**27**] is another illustration of single screen VR, but differs from the previous systems in that it uses the tabletop paradigm, where the display is back-projected onto a horizontal surface. Because objects can be physically placed *on* the display, the Responsive Workbench affords the use of widgets and other augmented physical objects as input devices. Recently, Ishii et al. have taken the idea of the Immersadesk and augmented for use with *phicons*, or physical icons. The resulting system is the metaDESK[**28**].

2.4.2. Spatially Immersive Environment. Spatially immersive environments have received considerable attention since Cruz-Neira et al. developed the CAVE Automatic Virtual Environment [**9**] (CAVE). These usually consist of several large screens onto which information is projected. In the case of the CAVE, the sides of a cube – and potentially the floor and the ceiling – serve as projection surfaces. Each one of the displays is *tiled* in that they together provide a seamless omni-directional view of the virtual scene. Further, the CAVE uses stereopsis to render 3D information within the user's space. The head of the user must be tracked in order to provide the correct perspective as the user moves about in the space. That is, perceptually, the user sees the virtual scene in a consistent manner as if it were real.

Many toolkits have been developed that aim to take advantage of CAVE-like spaces. Some are designed to run on single machines (usually supercomputers), while others use a cluster-based approach.

2.4.2.1. *Single Computer Rendering.* The original CAVE system uses the CAVELib [**9**] development environment to render the synthetic world. CAVELib provides a low-level API for creating VR applications. It handles the setup and initialization of the processes, and provides access to various input devices. Each display is managed by a different process, thus making the use of shared memory necessary. The main limitation of CAVELib is that it forces the user to deal with shared-memory issues. The library's main strength is its level of acceptance, as CAVELib has been used in multiple projects over the years [**29, 30**].

Other development environments have been created based on the same principle, such as VR-JUGGLER [**31**], ALICE [**32**] or AVOCADO [**33**]. Both AVOCADO and ALICE are interesting in that they distribute the processing of the information between multiple computers but rely on one machine to do the actual rendering.

The main problem with single computer rendering is cost. The price of high-end visualization systems tends to be several orders of magnitude higher than that of regular PCs while being much harder to upgrade when outdated. For this reason, many newer toolkits tend to favour clustered rendering techniques.

2.4.2.2. *Clustered Rendering.* Off-the-shelf PCs are usually used as the computing nodes of a cluster, which, unlike the processor grid of a supercomputer, does not need to have rigid topological constraints. Clusters were first developed for distributed number crunching applications, especially in fields such as physics, chemistry or military applications. However, using clusters for inherently interactive applications such as VR presents a new set of challenges. VR systems need to be able to receive input, process application data and produce output at least 10 times per seconds in order to keep from degrading the sense of immersion [**34**]. Original cluster hardware and software were not designed for closely synchronized multiprocessing. In fact, the purpose of clusters is to provide extremely fast communication between the computing nodes, but not with the outside world. Conversely, the main concern for interactive applications is not only frame rate, but also responsiveness of the system. Data must go through the network to reach all the nodes, hence adding some latency. Additionally, one must add a specific layer to coordinate the load balancing.[7] This further increases the latency.

One of the first projects to use clusters for rendering, not only for data processing, was Net Juggler [**36**]. It was built on top of VR-Juggler and uses MPI [**37**] (Message Passing Interface) for the communication between the cluster and the VR setup. Net Juggler relies on high-end PC graphics cards along with SoftGenlock [**38**] to synchronize the displays. The graphics primitives are intercepted and broadcast to the node controlling the appropriate display. Cluster Juggler [**39**], another layer on top of VR Juggler, has the same purpose but relaxes the requirements (for example that all nodes should have identical hardware) on the cluster back-end. Many other projects use the same principle to create an affordable yet powerful cluster-based rendering system, for example Chromium [**40**], a low level system for manipulating streams of graphics API commands on clustered workstations; blue-c [**41**] and X-Rooms [**42**], which are both complete VR setups with display surfaces and dedicated

[7]Static partioning across processors (i.e. static load balancing) is sub-optimal and outperformed by all others dynamic schemes (such as Grid-Bucket or KD-Split) for rendering systems [**35**].

clustered graphics engine; and more recently JINX [**43**], which is a cluster-based data browser for VR environments.

Clustered rendering is thus attractive for the computer power and the rendering speed it can achieve at a reasonable cost. On the other hand, it requires more effort to synchronize all the nodes and may introduce higher latency due to the network transport of information between the nodes.

CHAPTER 3

Qave Engine

"Who are you going to believe, me or your own eyes?"
—Groucho Marx

The Qave graphics engine is the centrepiece of this thesis. It is responsible for rendering the virtual world, which usually consists of pre-designed models in a textured environment. In order to achieve a good sense of engagement, the synthetic world must not only change dynamically with as little lag as possible in response to direct user input,[1] but also compensate for the user's position as to render the correct perspective. In other words, the renderer must be able to modify the synthetic world based on information received from independent tracking sources.

In our framework, the user information obtained from distinct processes is transmitted through the network using the ServerLib library, which allows message passing between applications registered with a nameserver, and provides support for binary data (such as video) streaming. On the send-side, ServerLib offers both blocking and non-blocking socket I/O. In the blocking case, the sender waits until it has received acknowledgment that the last sent packet has been received. In the non-blocking

[1]Examples of direct user input are object selection and displacement.

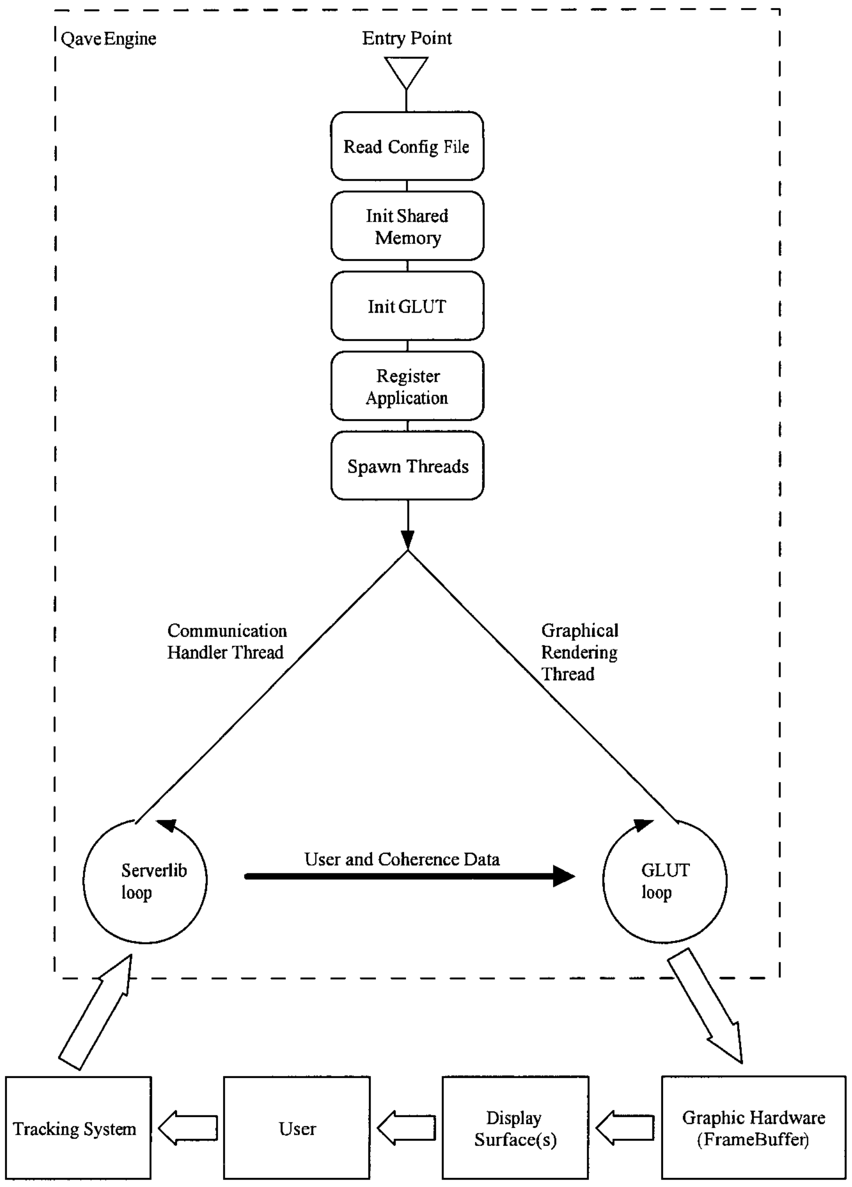

FIGURE 3.1. Qave Threads

case, the sender transmits the data packets without waiting for such an acknowledgment. Similarly, both blocking and non-blocking socket I/O is available on the receiving side. A timeout value is required in both cases but can be set to 0, thereby effecting a poll. However, polling is inefficient in its use of system resources, and thus, ill-favoured for applications with bounded constraints of responsiveness. It was therefore decided to use blocking I/O on the receiving end. Blocking I/O also implies certain constraints, as the execution of the program must stall until a packet of data is received or a timeout occurs. The Qave engine mostly receives data, and as such, its graphics branch could have been implemented as a timeout handler. However, since the OpenGL main loop is non-terminating, it will never cede control unless explicitly preempted, thereby resulting in deadlock. To avoid stalling issues, the engine spawns two distinct threads,[2] one for the graphics renderer and the other one for the communication handler.

A thread is an encapsulation of the flow of control in a program where all data except for stack and registers is shared. Threads can run concurrently by sharing the processor time. For the Qave engine, these characteristics imply that any update made by the I/O thread will be available transparently to the rendering thread without the need to stall when not receiving data. This allows Qave to achieve both high frame-rate and low latency to user input (DG2).[3] Figure 3.1 shows an overview of the work division and data flow between the two threads and the outside world. It must also be mentioned that such a multi-threaded design is quite common for high-performance graphics engines [**31, 32, 33**].

What makes this engine different from others is that it was purposefully designed to be a performant lightweight alternative to CaveLib that could be run in both our environments. The native OpenGL layer is accessible to the programmer, who can then directly use optimization techniques at the rendering level (such as hidden surface removal for example). Because of its simpler internal structure, Qave is usually able to perform better than its commercial counterpart CaveLib.

[2]To be more specific, POSIX threads (also known as *pthreads*) were used. Pthreads ensure a broad compatibility among UNIX based-platforms.

[3]The engine must be able to make use of the three-screened spatially immersive display of our original environment, as well as the large screen display in our new setup.

It must also be noted that newer frameworks are viable alternatives both to CaveLib and the one we present here. Chronium [**40**] in particular is interesting in the sense that it allows the use of a computer cluster where the rendering load is split between the nodes. Scalability –in terms of number of nodes in the cluster– is also notably good. The downside is that Chronium, in its current form, can only render pre-loaded data. This means that major features of our proposed extended framework such as live video blending and texturing are not supported. Other frameworks such as blue-c [**41**] or X-Rooms [**42**] are also practical alternatives, but they both necessitate customized hardware (such as glass panels with liquid crystal layers in the case of blue-c). This is a requirement that we wanted to avoid.

In the remainder of this chapter, we present the architectural and implementation issues from the different parts of the engine. Section 3.1 goes over the design rationale of the graphics rendering thread while Section 3.2 describes the communication thread. Section 3.3 provides details concerning our model list. Finally, Section 3.4 describes the simulator extension that was developed alongside the Qave engine in order to facilitate its use. The simulation mode allows the program to be run on a regular PC without having to use the SRE's immersive setup.

As mentioned in Chapter 1, a sample application was developed to demonstrate the potential of the framework. References to that application will be made throughout this chapter in order to better illustrate the function of each component.

3.1. Rendering Thread

The rendering thread is responsible for displaying the appropriate data to the user. The renderer relies on the OpenGL library to interact with the graphics hardware. The OpenGL Utility Toolkit,[4] (GLUT) was used to simplify the initialization process of OpenGL and hide the low-level platform-related specifics. GLUT and

[4]See http://www.opengl.org/resources/libraries/glut.html for more details

OpenGL are both available for many operating systems,[5] and because the graphics part of the Qave engine only relies on these two libraries, adapting the engine to work with other supported platform is feasible (DG1).

3.1.1. OpenGL Background.

OpenGL was introduced in 1992, and has now become the standard application programming interface (API) for 2D and 3D graphics applications. To date, OpenGL rendering performance is commonly regarded as the standard benchmark to compare the processing power of different video hardware subsystems.

3.1.1.1. *Modelview and Projection Matrix.* OpenGL draws *primitives*, which are points, line segments, or polygons. Primitives themselves are defined by a group of one or more vertices. A *vertex* defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colours, normals, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its corresponding data are processed independently, in order, and in the same way. OpenGL relies on two matrices convert vertex data to raster data: the Modelview and the Projection matrix.

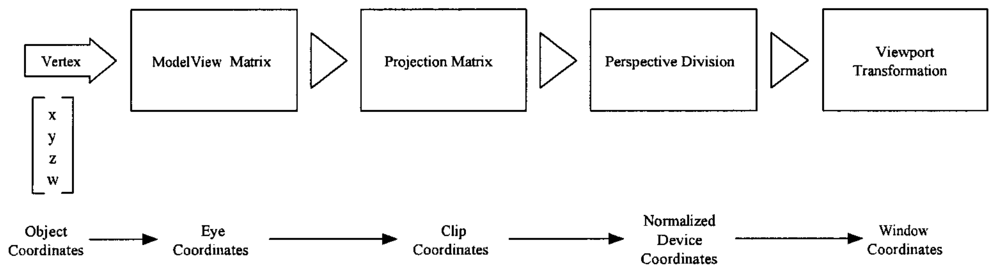

FIGURE 3.2. Vertex Coordinate Transformations

Figure 3.2 illustrates the coordinate system conversion performed at each vertex transformation. In the OpenGL pipeline,[6] a point is normally represented as a column

[5]Supported operating systems for GLUT and OpenGL so far are: Linux, Irix, Solaris, MacOS X, Windows 9X/Me/NT/2000/XP

[6]See Section 2.1 of the Blue Book [44] for more information on the rendering pipeline

matrix with four elements:

$$\begin{bmatrix} x & y & z & w \end{bmatrix}^T$$

where x, y, z are the non-homogenous coordinates and w is the homogeneity scaling factor. Having the coordinates in a homogenous reference system makes it easier to write 4x4 matrix transformations for scaling, rotating and translating.

The first matrix of interest is the Modelview matrix. Its role is to transform object coordinates to eye (or camera) coordinates. In more practical terms, it is the matrix that is used to manipulate objects in the scene. Inversely, the Modelview matrix can be used to simulate a camera moving through the scene. The Modelview matrix is initially an identity matrix, on which three operations can be applied: scaling, rotation and translation. The order of the transformation is important as matrix multiplications are not commutative. The pre-multiplication and the post-multiplication of two matrices will (generally) give different results. An easy way to see this is to realize that a rotation followed by a translation is quite different from a translation followed by a rotation. By convention, OpenGL usually uses the column major notation and post-multiplies matrix operations. Note that this convention is simply notational, as post-multiplying column-major matrices produces the same result as pre-multiplying row-major matrices. More practically, this means that the order of the transformations called will actually be performed *backwards* by the pipeline. For example, if the programmer specifies first a rotation then a scaling and finally a translation, the rendering pipeline will first process the translation, then the scaling and finally the rotation.

The other matrix of interest is the Projection matrix. The purpose of this matrix is to take a volume of 3D space and flatten the objects within it onto a plane, corresponding to the screen. In other words, it converts the vertex data from eye coordinates[7] to clip coordinates.[8] There are two kinds of projection: orthogonal and perspective. In the orthogonal case, objects are projected along the normal of the projection plane. This is the simplest form of projection, and it does not allow for depth to be conveyed. As such, its main use is to write data at a fixed location in screen coordinates. Perspective projection, on the other hand, allows for depth information to be rendered, in which case all vertex data belonging to an enclosed volume of space defined by six clipping planes (called the *frustum*) is projected onto a 2D plane, the viewport.

3.1.1.2. *Frustum.* By convention, OpenGL creates a view volume with the eye at the origin. The view rectangle is sitting at a depth of $z = -$near, with its edges defined by $x =$left, $x =$right, $y =$bottom and $y =$top planes, and the far clipping plane at $z = -$far (see Figure 3.3). In mathematical terms, the perspective projection matrix obtained is the following:

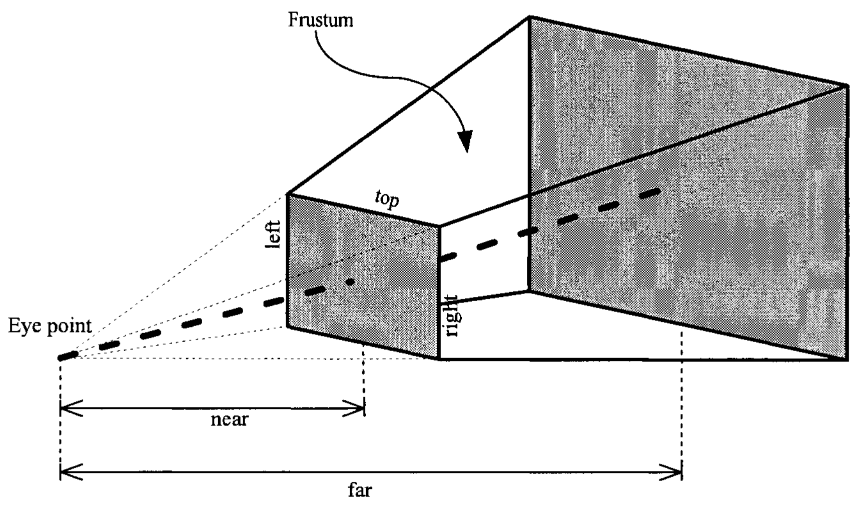

FIGURE 3.3. Perspective Projection Frustum

[7]The eye coordinate system is the reference in which the view vector coincides with the z-axis.
[8]The clip coordinate is the defines the space in which the object outside of the viewing frustum are discarded.

$$\begin{bmatrix} \text{near} & 0 & 0 & 0 \\ 0 & \text{near} & 0 & 0 \\ 0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -\frac{2\text{far}*\text{near}}{\text{far}-\text{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.1}$$

It is not, however, the exact matrix that is used in the pipeline. On top of perspective projection, OpenGL also performs a translation that shifts the scene symmetrically around the origin, and a scaling to make the transformed x and y values fall in $[-1, 1]$ for clipping efficiency. This amounts to a translation by a vector $\begin{bmatrix} -\frac{\text{left}+\text{right}}{2} & -\frac{\text{bottom}+\text{top}}{2} & 0 \end{bmatrix}^T$, and a scaling by factor $\frac{2}{\text{right}-\text{left}}$ on the x axis and $\frac{2}{\text{top}-\text{bottom}}$ on the y axis. Simple matrix algebra provides the formulation that is actually used:

$$\begin{bmatrix} \frac{2\text{near}}{\text{right}-\text{left}} & 0 & \frac{\text{right}+\text{left}}{\text{right}-\text{left}} & 0 \\ 0 & \frac{2\text{near}}{\text{top}-\text{bottom}} & \frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}} & 0 \\ 0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -\frac{2\text{far}*\text{near}}{\text{far}-\text{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.2}$$

3.1.2. Coordinate System. Because of the physical characteristics (see Figure 1.2) of the immersive space, the engine was designed with the following coordinate system: the origin is on the floor at the center of the setup, and the coordinates follow a right-handed base (i.e. with positive x pointing to the right, positive y pointing up and positive z pointing toward the back of the room). The scaling is in feet, i.e. displacing an object by 1 unit in OpenGL displaces it by 1 foot in user space (see Figure 3.4).

3.1.3. Viewports and Off-Axis Projection. In our original SRE setup, the rendering hardware takes care of splitting the single graphics channel into three distinct feeds for the projectors. This split is transparent to the developer, who can

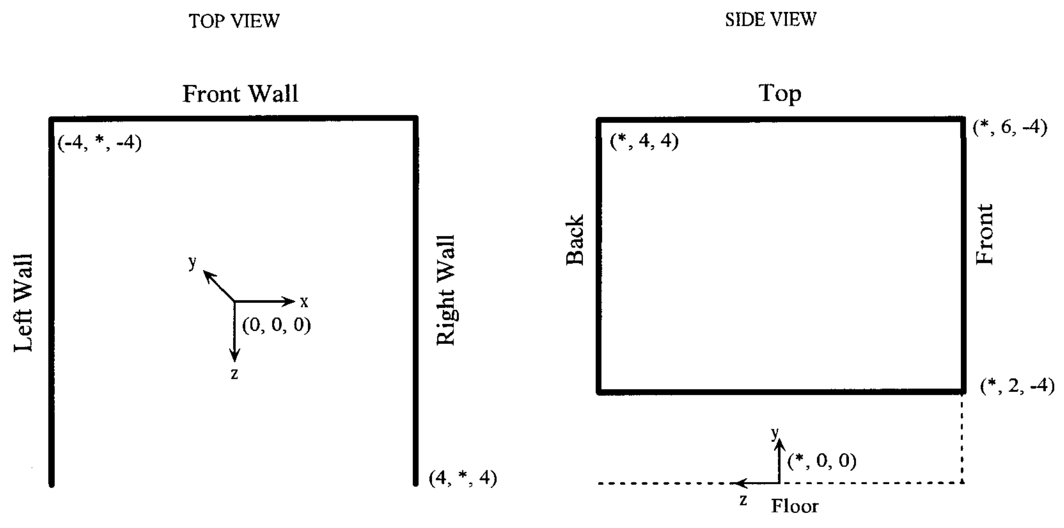

FIGURE 3.4. Coordinate System in SRE

consider what is projected on the three physical screens as being a single large window.[9] Because each screen must render the appropriate perspective for the user, that window must be split into three viewports that correspond to the physical screens. A viewport specifies an area of the frame buffer to use for rendering. The generic call to create a viewport in OpenGL is:

```
glViewport( GLint x, GLint y, GLsizei width, GLsizei height )
```

In our case, the appropriate call is the following:

```
glViewport ((N*window_width)/3, 0, window_width/3, window_height);
```

where N is an integer between 0 and 2 denoting the viewport number. Figure 3.5 shows the standard viewport configuration in the SRE, where the resolution used is 3072x768 pixels.

Once the viewports have been created at the appropriate positions in the framebuffer, the rendering loop must compute the correct projection matrix to be used in these viewports. As opposed to desktop-based systems, one significant characteristic

[9]As for our new setup, different options are still being explored for multiply overlapping projection, so in the meantime we use a single projector to display the world.

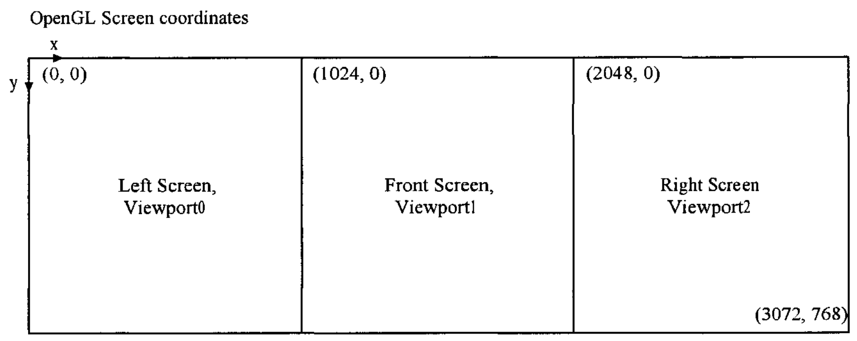

FIGURE 3.5. Viewports in Qave

of spatially immersive environments is that the user should be able to look and move around while remaining immersed perceptually in the synthetic world. Since the user is not constrained to a specific location in the physical space, the viewpoint does not necessarily lie on the normal axis of the projection plane,[10] as it normally would in a single screen environment. This means that the data displayed must be adjusted for the user's perspective, with a technique called *off-axis projection.*

As noted by Cruz-Neira et al. [**9**], the general projection matrix then becomes:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{h_x}{h_z-D} & -\frac{h_y}{h_z-D} & 1 & -\frac{1}{h_z-D} \\ \frac{h_xD}{h_z-D} & \frac{h_yD}{h_z-D} & 0 & \frac{h_z}{h_z-D} \end{bmatrix} \tag{3.3}$$

where h_x, h_y, h_z are the coordinates of the user's head and D is the distance from the origin to the projection display.[11] (see Figure 3.6) The matrix 3.3 provides a correct frustum for the front viewport, and must be adapted for the two others. As the user moves within the space, the frusta are updated in real-time to provide a correct perspective of the synthetic world (DG3).

[10]In our case, the projection plane corresponds to the actual walls.

[11]More specifically for our configuration, distance D is half the length or width of the room, that is 3 feet.

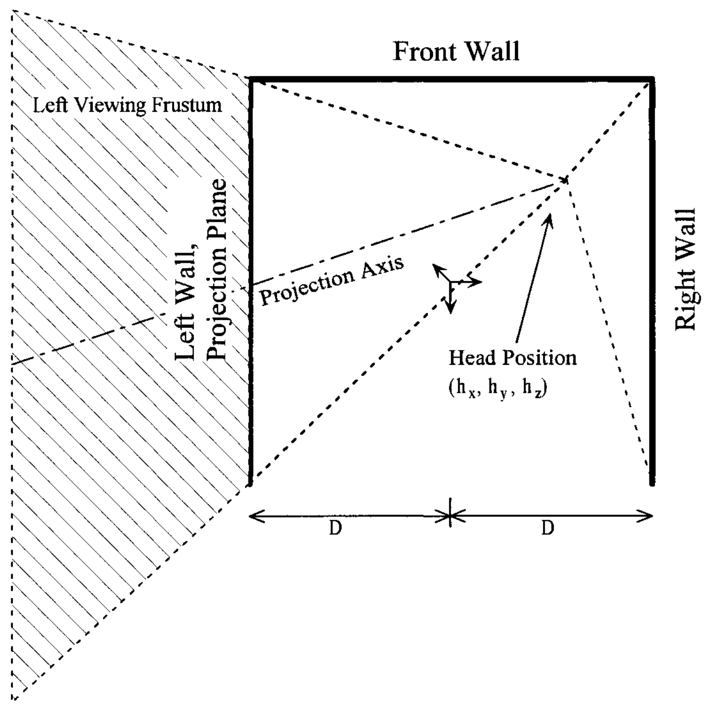

FIGURE 3.6. Off-Axis Projection in the SRE

The general OpenGL call to create a viewing frustum is (see Figure 3.3):

```
void glFrustum( GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdou-
ble near_val, GLdouble far_val )
```

In the SRE context, the appropriate call for the centre frustum is:

```
glFrustum( qave_left - headpos.x, qave_right - headpos.x, qave_bottom - headpos.y ,
qave_top - headpos.y, ABS(qave_front - headpos.z ), far)
```

where qave_front, qave_back, qave_left and qave_right are the wall locations, whereas headpos(x,y,z) are the coordinates of the user's head. Calls for the other two frusta are slight variations of the above, where the coordinate system is permuted from (x,y,z) to (-z,y,x) for the right frustum and (z, y, -x) for the left frustum.

3.1.4. Rendering Program Flow. As shown in Figure 3.1, once the initialization steps are completed, Qave spawns the I/O and the rendering threads. The rendering thread runs the GLUT main loop until the program is terminated. As opposed to normal OpenGL applications, where one simply sets the perspective and draws the objects, immersive environments with multiple displays require a slightly different approach because the projection parameters can change dynamically.

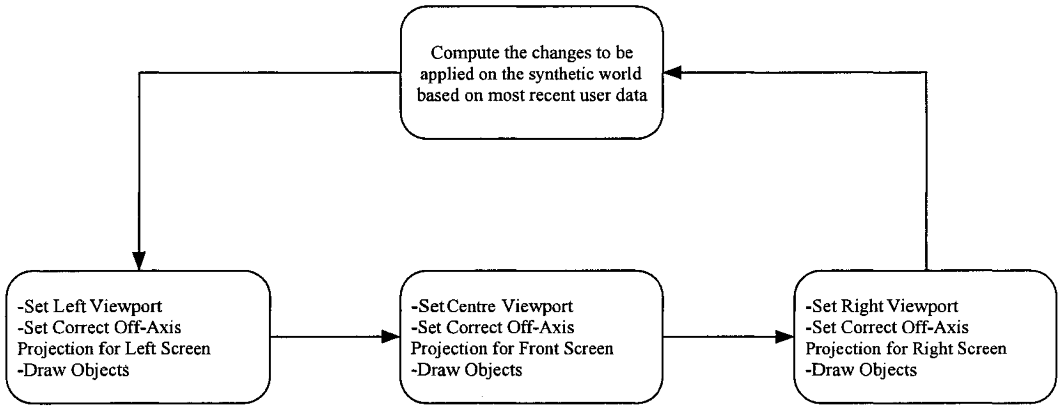
Compute the changes to be applied on the synthetic world based on most recent user data
-Set Left Viewport
-Set Correct Off-Axis Projection for Left Screen
-Draw Objects
-SetCentre Viewport
-Set Correct Off-Axis Projection for Front Screen
-Draw Objects
-Set Right Viewport
-Set Correct Off-Axis Projection for Right Screen
-Draw Objects

FIGURE 3.7. Rendering Loop

As can be seen in Figure 3.7, the rendering loop will compute a new set of viewing frusta based on the latest available user position at every frame . Once the new frusta are computed, they are rotated to match to the corresponding physical screen.

For each viewport, the next step consists of determining the changes to be made on the world. More importantly, this step is where the potential developer is expected to take over and define the actual application in terms of object interactions and events. World updates can be triggered either by user information, or by remote coherence data. Note also that because the user position is updated in another thread, the renderer always assumes it has the latest set of position data at its disposal. Due to current tracker limitations, the user data consists of the coordinates of the head and one hand of the user. From that information we can determine where the user is pointing.[12] The distance between the head and the hand can be thresholded thereby providing a clutch mechanism akin to a click on a mouse. This allows for an easy *grab-like* gesture. Within the scope of our sample application, once we know the angle of the user's pointing gesture, we cycle through a list of all selectable objects in the scene to determine if the user actually points to an item. If an item is found and if the distance between the head and the hand exceeds a certain threshold, the object is considered to be selected. In our sample application, the model then follows the hand movements until released.

Ultimately, more refined user information is needed to create a richer interaction. Sequences of hand or finger postures could be interpreted as gestures with a specific semantic and syntax. Such gestures could be used for more sophisticated bimanual controls, such as grouping or resizing. Non-obvious instructions, relating for example to object texturing or lighting, could be assigned to sequences of gestures. Such a system would require the precise detection of both hands along with the associated finger posture. Temporal information would also be needed to define the gesture boundaries. Further, an adequate parser and interpreter would be required to determine whether the gesture is meaningful and intentional.

3.1.5. Configuration File. The configuration file allows changing core Qave characteristics at run-time without the need to recompile the application. It is comprised of two parts: the first one is the Qave configuration while the second one is an initial list of models to be loaded at initialization time (see Section 3.3). The Qave configuration parameters are:

- *Window width and height.* This allows the resizing of the main window so that higher resolutions can be used in the future.

[12]The exact target of the pointing gesture cannot be determined as a 3D point, but the general pointing direction can be estimated by tracing a line from the user's head to his hand.

- *Viewport Number.* This switch indicates how many viewports are used during the rendering. More specifically, it determines whether the engine will run in immersive mode, in which case the number of viewport is 3, or in simulator mode (see Section 3.4), if the viewport number has any other value.
- *Screen positions in the Qave coordinate system.* These determine where the screens are placed, and thus affect the frustum computation. Note that the screens are assumed to be orthogonal to each other; more specifically that the left and right screen are z-axis aligned while the front screen is x-axis aligned.

After the Qave configuration variable, the user can define a list of models to be initially loaded at run-time. This is useful for including new models without the need to recompile the application. The syntax to include a model is:

```
<model_name> <model_id>
```

The model_name is the name of the file containing the model data, while the model_id is a unique identifier (that is there could be multiple objects from the same model file).

3.1.6. Video Stream Blending. As mentioned in Chapter 1, remote collaboration and telepresence are two fields of great interest. It has been shown that the quality of the remote user's representation is "crucial in situations where problems need to be solved", and that "the use of video enhances the collaboration" [**14**]. It is thus important for the Qave to support video embedding in the synthetic world. Furthermore, Garau et al. have shown that the quality of a semi-photorealistic or photorealistic avatar played an important role for gaze inference [**15**] in computer mediated human-human interactions. Gaze inference and awareness is a very important non-verbal cue which normally allows the identification of the interlocutors in a discussion.

Technically, we rely on the ServerLib binary data transport to receive the video stream. At every framebuffer swap, we take the most recent video frame received and extract a texture from that data (DG7). For better immersion, the incoming video frames usually go through a background removal process. This isolates the distant users from their physical surroundings. Pixels marked as background are then treated as transparent during the texture creation. The obtained texture is then pasted on a transparent polygon. This allows the blending of the remote video in the synthetic scene (see Figure 3.8).

A transparent physical material, such as the one on which the incoming video is pasted, shows objects behind it as unobscured and does not reflect light off its surface. To correctly render the transparent material, we must make use of the blending function in OpenGL:

```
void glBlendFunc( GLenum sfactor, GLenum dfactor )
```

where sfactor and dfactor respectively specify how the source and destination blending factors are computed. In our case, we set sfactor to GL_SRC_ALPHA and dfactor to GL_ONE_MINUS_SRC_ALPHA. With these settings, the incoming colour is modified by its associated α value[13] and the destination colour is modified by $(1 - \alpha)$. The sum of these two colours is then written back into the framebuffer, hence creating a translucent polygon. Because of this mechanism, correct results for transparent rendering are only guaranteed if the primitives are sorted and rendered from back to front with depth testing enabled.

[13]The α value of a pixel determines its level of transparency.

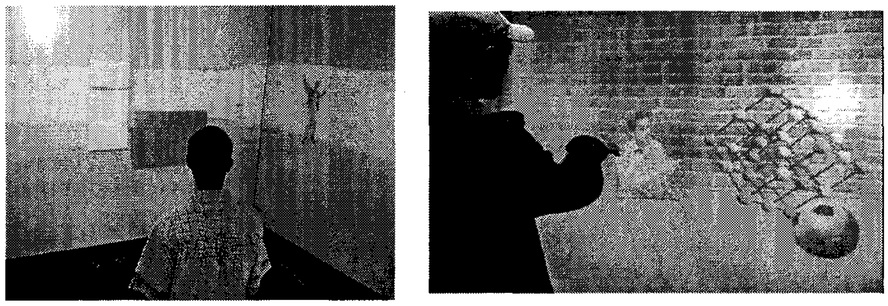

FIGURE 3.8. Video Embedding Examples in the SRE

3.2. Communication Thread

The communication thread manages all data input and output. The thread starts a ServerLib Server that can accept messages from other applications (DG5). For the message passing mechanism to work properly, the Qave must register with a nameserver. The nameserver receives the registration data, stores the IP addresses of its clients and associates them with a unique ID. Once the registration is successful, the application sends the nameserver a list of commands (along with the associated parameters) it can accept. If another registered application tries to call an invalid (i.e. undeclared) function, an error will occur and no message will be transmitted to the target application.

Qave has a set of pre-defined commands that are required for core functionality. These commands are normally invoked either by a tracker or by the WorldServer. A developer can easily add commands by declaring them in the command list and defining *ad hoc* callbacks (DG8). The predefined commands are listed in Table 3.1.

3.3. Model Object

OpenGL is a powerful vertex-based rendering library, but creating complex object by hand is tedious as one has to define all the vertices along with lighting and material data. Qave provides built-in support for automating the process and allowing for

setperson pos <person no> <value1> <...>	Sets the position values for head (in tracker coordinate system). This function is normally used by a tracker.
lookat eye<xyz> center<xyz>	Sets the position values for direction of user look (in Qave coordinate system), with eye<xyz> being the eye point coordinate and center<xyz> being the position of the reference point. This call is the equivalent of calling a gluLookAt(). This function is only defined in simulation mode, where the user can change the view orientation of the single viewing frustum.
touch <xyz>	Sets position values hand (in Qave coordinate system). This call is often used to quick debugging purposes. It is easier to use than the sethandpos because it takes values in Qave coordinates.
sethand pos <person no> <value1> <...>	Sets the position values for hand (in tracker coordinate system). This function is normally used by a tracker.
receivevideo1	Receive video frames for person1. This starts the reception of the remote video stream.
share <model_id>	Instruct Qave to be coherent with its representation of model <model_id> with other Qaves. This call is used in the context of the WorldServer.
update <model_id><xyz-pos><rot_matrix><xyz-scale>	Updates the data about a shared object. "*" may be used as *don't cares.* Updates targeting a non-shared object are ignored.
reqUpdate <model_id>	Requests an update about a shared object. This call is usually made by the WorldServer.
getviewerpos	Prints the position of head and hand to stdout in Qave coordinates

TABLE 3.1. Pre-defined Qave Remote Calls

pre-designed models to be imported in the scene as Wavefront[14] objects. Wavefront OBJ (object) files[15] are used by Wavefront's Advanced Visualizer application to store geometric objects composed of lines, polygons, and free-form curves and surfaces. OBJ files are extensively supported in the CAD industry, and converters to and from other model formats are widely available.

Robbins[16] wrote a OBJ importer for OpenGL which has been adapted for use in Qave. In particular, Qave uses an array of object structures, where each node in the list maintains its own set of data (see Table 3.2). The reason why we maintain all the model information in a data structure is that it allows the processing of all the world updates before the actual rendering takes place. A batch function then reads the information from the data structures and renders all the models with the appropriate ModelView matrix transformation. This speeds up the rendering pipeline. This also allows for greater synchronism among the objects because all the world updates are processed within a smaller time frame.[17] Such a structure also makes it easier to manage the coherence data obtained from the WorldServer (DG6)(see Chapter 4).

Note that the rotation data is not recorded in the angle-axis format but in full matrix representation, thereby easing the processing of subsequent rotations applied to a model. Note finally that Qave provides two ways of manipulating the orientation of the models. One can set an absolute angle that the model should have; in this case the rotation matrix in the data structure is simply overwritten. Alternatively, one can use an incremental angle change, in which case the old rotation matrix will be multiplied with the matrix representation of the increment.

[14]See http://www.alias.com/eng/index_flash.shtml for more details
[15]See http://www.dcs.ed.ac.uk/home/mxr/gfx/3d/OBJ.spec for the full format specifications
[16]http://www.pobox.com/~nate
[17]This is particularly evident if an object is moved with respect to another one while an update changes the location of the reference, in which case the update will not affect the object already rendered.

clicked	Determines if the object is being clicked on.
shared	Determines if the object is being shared.
x, y ,z	Position in 3D of the model.
rot_matrix	Holds the full 4 by 4 rotation matrix.
model_id	Contains the object unique id string.
xscale, yscale, zscale	Contains the scale along the x, y and z axis.
pmodel	Pointer to the structure containing the actual vertex data of the object as well as its filename and other data.

TABLE 3.2. Object Structure Data Fields

3.4. Simulator Mode

Instead of making use of the immersive setup, it is often useful to preview a scene on a regular computer (DG4). Regular PCs are much more common and easier to setup than a full-fledged immersive environment. When in simulator mode, the engine uses an alternative core rendering loop adapted for use on single screen setups. The simulator mode is thus an integral part of the graphics renderer, and not, as is the case for other engines [**45**], a completely separated application. This allows the developper to verify the correctness of his application in the same engine that would be used in an immersive setup. Consequently, to start the simulator, one does not need to recompile or launch a different application, but simply to edit a configuration file (see Section 3.1.5) and change the number of viewports. At that point, the rendering will not be made on three viewports but on one. This is the equivalent of only rendering the front display in the immersive case. Because we use on-axis projection in that configuration, we can make use of the built-in OpenGL perspective calls to simplify the projection matrix computation.

Note also that since the simulator actually *is* the same program, it is also possible to render the scene in immersive mode on a single screen. In that case, the view will be skewed because the projection matrices will not be adequate for the single-screen environment. On the other hand, this allows the developper to have the exact preview of the immersive scene on a regular PC.

Of course, the simulator should not have to depend on the trackers present in the immersive setup to process user input. A keyboard control override scheme is used to displace the hand of the user and to manipulate the objects in the scene. To obtain maximum flexibility, the keyboard override can be triggered on or off. If it is off, the simulator will process the data coming from the trackers. The controls are summarized in Table 3.3.

up, down, left, right arrow	Control the hand position in the z and x axis respectively.
page up, page down	Control the hand position in the y axis
F12	Triggers the keyboard override
END	Triggers an action (click)
a,d, s, w	When an object is selected, controls the rotation along the y and the x axis respectively
q, z	Control the uniform (that is along all the axis) scaling of the model

TABLE 3.3. Keyboard Override Key-List.

3.5. Results

Performance analysis of software can be done using three different methods: measurements, simulation and analysis [**46**]. The measurement method uses direct timing of the software. The simulation method performs the analysis on programs created to simulate the execution of the software being tested. The analytical method involves creating mathematical models to represent the software and extrapolate from the behavior of that model. The peformance analysis method presented in this section is based on the measurement method, because the software is in a fully workable condition.

The Qave engine was tested on different configurations for performance benchmarking. For analytical purposes the results of Qave are compared with those of CAVELib, the engine that previously drove our setups. The running variable used was the scene complexity, evaluated in terms of number of lit triangles in the scene. Triangle count is a commonly used complexity metric as it is the basic atomic processing unit for graphics processors. The performance was measured in frames per seconds (fps) for different scene complexities. Frames per second is the most significant figure for graphics engine capability. Other measures (such as the number of polygon/seconds drawn) are also sometimes used, but have less correlation with real life performance. All scenes were rendered using double-buffering, which is the typical usage scenario for Qave. Single-buffering often produces flickering in the display as the framebuffer is getting drawn while being written. Flickering, of course, is detrimental to the immersive experience.

A sample scene consisting of a textured floor and three identically textured walls was used as our basic world. To this world we then added different OBJ models[18] to vary the complexity parameter.

The specifications of the machines used are:

[18]The number of triangle in each model is known in advance.

- Bach: SGI Onyx2, 2 R12000 processors at 400MHz, 256MB of main memory, InfiniteReality2 “KONAL” graphics engine with 8 channels and 64MB of video RAM. OS: IRIX 6.5
- Watcher: PIII 850MHz, 256MB of RAM, with a Matrox G400 with 32MB of video RAM. OS: Linux, kernel 2.4.25
- Scarlatti: P4 2.6GHz, 512MB of RAM, with an NVIDIA FX5200 128MB of video RAM. OS: Linux, kernel 2.4.20

For our first benchmark, we tested the Qave engine in two different configurations for each machine:

- 1 viewport configuration, thus giving a total resolution of 1024x768 pixels
- 3 viewports configuration, with a total resolution of 3072x768 pixels

Figures 3.9 and 3.10 show the framerate vs. the number of colored triangles rendered of that first benchmark.

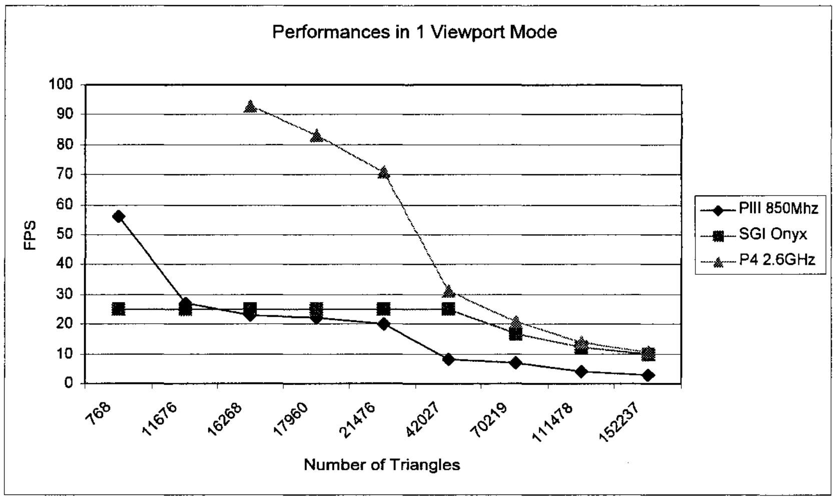

FIGURE 3.9. Qave Performance for the 1 Viewport Configuration

First, note that the first and second data points for the framerate of the higher-end PC in the single viewport configuration are off the chart and have been omitted both

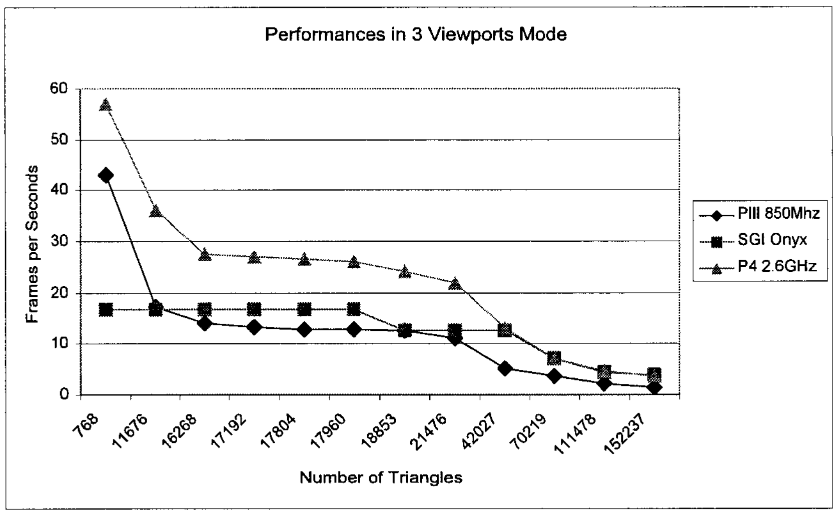

FIGURE 3.10. Qave Performance for the 3 Viewports Configuration

in Figure 3.9 and 3.12 for legibility purposes. Their respective values are 950fps and 128fps. Such high framerate is due to heavy optimization of the graphics hardware for simple scenes at low resolution.

Notice also that the maximum framerate that can be obtained on our SGI configuration at a refresh rate of 50Hz is 25 frames per second. There is such a limit because we use double buffering during our rendering, and the Onyx needs the vertical sync (VSYNC) enabled to ensure a consistent display on the three screens. Because of that constraint, the InfiniteReality engine has to wait until the next refresh cycle to swap the buffers. Hence, the consequence of the waiting time due to buffer swap is that the framerate in double buffered mode is a multiple of 50 (i.e. the refresh rate). The framerate is thereby quantized, with the maximum framerate being 25fps, the next ones being 16.6, 12.5, 10 and so forth. It is of interest to note that regular PC graphics cards will also exhibit such quantization if the VSYNC setting is enabled, in which case, similarly to the Onyx, the framerate will be a multiple of the framebuffer refresh rate. This is important in making performance comparisons, as the Onyx is impaired in its ability to compete with other architectures on scenes of modest or low complexity, i.e. the scenes in which the performances is bounded entirely by graphics card refresh limitation.

The most interesting result of this first benchmark is that both PCs usually offer a higher framerate when the scene is simple, but the performance of the low-end PC degrades markedly to fall below that of our SGI when the scene contains more than 20000 triangles. That quick degradation is notably due to the limited onboard memory and the slow speed of the video card. On the other hand, the high-end PC manages to keep higher framerates than the SGI in all the tested cases. This shows that current off-the-shelf PCs can indeed perform as well as older specialized visualization computers.

The other interesting result is that the low end PC actually performed better than the Onyx for simple scenes. Such a result is probably due to the hard-coded framerate limit of the Onyx (25fps).

The minimum framerate usually considered as acceptable for interactive systems is 10fps [**34**]. From the data shown in Figure 3.10, we can determine the most complex scene that can be rendered in the Qave in three viewports mode with acceptable framerate should not contain more than 50000 lighted triangles on an SGI. The same figure also holds for our higher-end PC, which by no means is the most performant PC available. One can speculate that a top-of-the-line processor and a matching graphics card could easily outperform the Onyx.

The second test compares the performance of Qave and CAVELib. Both programs were run on the Onyx2 (see Figure 3.11) and on a P4 running Linux (see Figure 3.12). On the Onyx2, both programs were set to use three viewports for a total resolution of 3072x768 pixels. On the P4, one viewport and a resolution of 1024x768 were used.

Figure 3.11 shows that CAVELib and Qave have similar performance on the Onyx. Because of quantization effects, both engines exhibit performance degradation

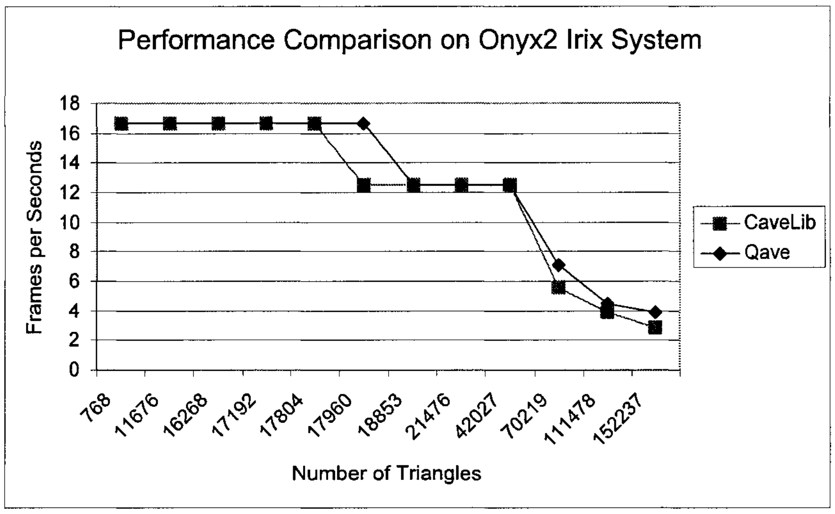

FIGURE 3.11. Performance Comparison between Qave and CAVELib on Onyx2

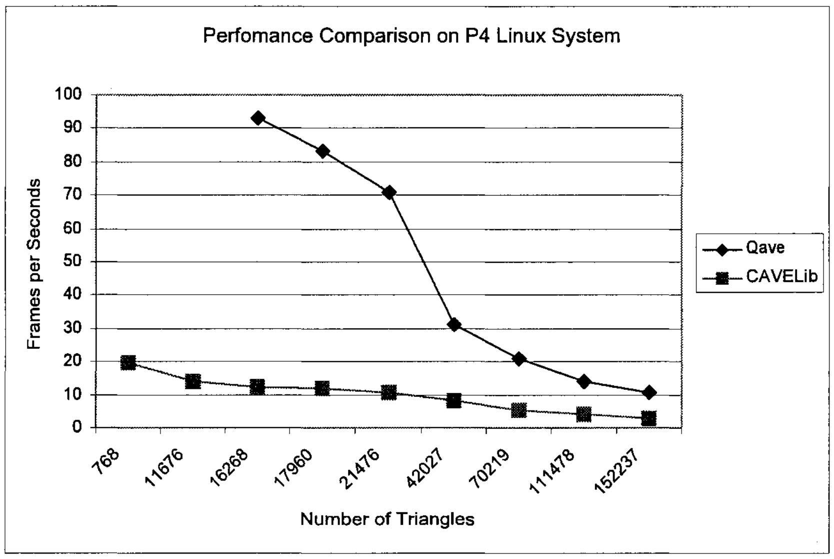

FIGURE 3.12. Performance Comparison between Qave and CAVELib on a P4

at around 18000 lit triangles, with Qave providing a slightly better performance, overall, to that of CaveLib on the SGI.

When running under Linux (see Figure 3.12), Qave proves to be a much better performer than CAVELib, reaching framerates up to four times higher. This dramatic difference under Linux is probably due to the fact that CAVELib was specifically designed to take advantage of the SGI platform, notably in terms multi-processing and shared memory; under Linux, such optimizations were not available.

Taking again the 10fps limit as the usability limit, Figure 3.12 shows that CAVELib on Linux can process a maximum of 21000 triangles. Qave, on the other hand, can go as high as 150000 triangles, seven times more. Given the fact that our new setup will use regular PCs running Linux for rendering, the Qave is clearly the preferred solution.

To summarize, Figures 3.9 and 3.10 compare the performance gradient of the Qave engine on different configurations, while Figures 3.11 and 3.12 show how Qave and CAVELib compare on similar setups. The first set of data shows that Qave scales nicely with more powerful PCs, both in single- and three-viewports modes. Further, this data also suggests that for scenes of relatively modest complexity Qave performs better on off-the-shelf PCs than on our SGI platform. The second set of data shows that the our newly implemented Qave engine performs better than its predecessor, CAVELib. This is notably true when both platforms are benchmarked on PC systems. On the Onyx2, the increase in peformance can mostly be seen by a larger range of data at which the graphics engine renders at a structurally-bound top speed, which, in most cases, is lower than what can be achieved on a reasonably recent PC. This is of particular significance for the research community as it shows that today's consumer-level PCs can outmatch older specialized graphics workstations. Most recent research in VR platform now use clustering methods to reach even higher framerates with complex scenes. Such methods are usually complex to setup and require multiple machines. Our results show that for relatively simple scenes, a regular PCs is more than capable of rendering for immersive spaces.

CHAPTER 4

WorldServer

"The multitude which is not brought to act as a unity, is confusion. That unity which has not its origin in the multitude is tyranny."
—Blaise Pascal

The WorldServer's purpose is to maintain information about a coherent world that can then be broadcast to different Qaves. Each Qave uses that information to synthesize its own rendering of the *same* world. Sharing a coherent space notably allows distant users to collaborate as if they were in the same physical location.

A client/server model was used for the implementation of the transactions, where the server acts as a middleware and holds the world data which is then transmitted to clients. Another paradigm for maintaining the world coherence among Qaves is to adopt a peer-to-peer architecture, whereby each Qave directly talks to all the other Qaves. Such a paradigm is a reasonable solution for a pair or a small number of Qaves, but it does not scale well when the number of peers increases. With this transaction model, conflict resolution is a non-trivial issue.[1]

For example, a conflict could occur if multiple users located at different nodes decide to simultaneously throw a sheet of paper on the same stack. In a peer-to-peer

[1]Conflict occurs if two or more Qaves end up rendering a different world.

model, each Qave represents its *own* sheet as being at the bottom of the stack because it receives the data concerning the other sheets after it has rendered its own sheet. Clearly, the world is not in a consistent state anymore, as each user sees a different reality. A non-trivial arbitration among all the peers must then take place in order to have all the Qaves agreeing on a state as being what is *real.* In a client/server model, all the Qaves send notice to the WorldServer that they have put a sheet of paper on the stack. Then, the WorldServer makes the decision of the sheet ordering (most likely, but not necessarily, based on message arrival order) and broadcasts that decision to all the Qaves, thereby maintaining the world in a coherent state with minimal effort. Each Qave receiving the update contradicting its own version of the world will experience a temporary out-of-sync world before the update arrives, but that situation is much more desirable than having to resolve a conflict to determine which peer has the correct representation of the scene.

In Section 4.1, we describe the world data model, while in Section 4.2 we go over the transaction model and the communication protocol between the different modules of the framework.

4.1. Object and World Models

The WorldServer maintains data structures holding the position, orientation and scale of objects in the world (DG9). The Qaves are responsible for rendering their own local environments and the coherence paradigm we use is that only the necessary data is explicitly shared. More specifically, any module may initiate the sharing of objects which will then be coherent among all the Qaves. As such, Qave users may initiate the sharing of an object themselves, or another process or script might declare the share automatically at launch time. Ensuring completely similar worlds then requires running the same application on all the Qaves. However, Qaves that are sharing data have no formal requirement to do so. This scheme provides maximum flexibility

x, y ,z	Position in 3D of the model.
rot_matrix	Holds the full 4 by 4 rotation matrix.
model_id	Contains the object unique id string.
xscale, yscale, zscale	Contains the scale along the x, y and z axis.
id	WorldServer local object number id.

TABLE 4.1. Object Attributes.

x, y, z	Contains the 3D position of the Qave in the virtual world.
id	Contains the local WorldServer id for the client.
qave_name	Contains the name that was registered to the nameserver of the client

TABLE 4.2. Qave Clients Attributes.

and adaptability, as Qaves might not all have the same rendering capacities and might prefer to render a plain background instead of a complex scene to maintain performance and responsiveness. Table 4.1 shows the attributes maintained for each object in the WorldServer. Note that these attributes are essentially a subset of those of objects in a Qave (see Table 3.2).

In addition to object positions, the WorldServer also maintains information about the registered client Qaves, as is shown in Table 4.2.

4.2. Transaction Model

The transactions follow a client/server model. The ServerLib is again used to support the low-level message passing.

4.2.1. Operation Modes. WorldServers can operate in one of two modes for message broadcasting: synchronous or asynchronous. In synchronous mode, the server blocks after each message send out until an acknowledgment from the recipient is received. This is a useful mechanism to prevent Qaves from being overwhelmed by messages if the WorldServer broadcasts too many updates too fast. The downside of the synchronous mode is that there is a potential loss of performance in terms of update propagation speed. Further, it may not scale well when the number of clients increases. In summary, this mode produces a slower update rate, and should be used when it is known that a client might get flooded by updates.

In asynchronous mode, the WorldServer simply broadcasts the updates to all the Qaves as soon as it can. It is the mode that offers the lowest update latency, and its use is recommended if the Qaves can support the higher update rate. It must also be noted that message creation is event-based. This means that unless a Qave has something to broadcast, no world update will be sent out.

4.2.2. Message Protocol. The main guideline used while designing the message protocol was that objects in Qaves should not be connected unless they are explicitly shared. This means that two Qaves may be running totally dissimilar environments containing many different objects, and only share a few. The sharing of the objects is triggered by sending a message containing the object ID to the WorldServer. Anyone may send such a message (i.e. sharing may be initiated at startup by a script, or on-the-fly by clients themselves), and the WorldServer will then broadcast the appropriate instructions to all its registered clients (DG10). Explicitely declaring the shared object requires a bit more setup and initialization than forcing the sharing of the whole world data, but greatly enhances the flexibility.

As was described in the object models (see Table 4.1), each object is identified by a string that must be unique within the scope of the local Qave. For example, one

register <qave_name>	Registers a qave in the world and responds with and id number.
share <model_id>	Instruct all the registered Qaves to be coherent with their representation of model <model_id>.
update <model_id><xyz-pos><rot_matrix><xyz-scale>	Updates and broadcasts the data about a shared object.
reqUpdate <qave_name><model_id>	Requests client <qave_name> to send the server an update about model model_id.

TABLE 4.3. WorldServer Command List.

Qave could not have two objects with id "object_1", but two separate Qaves could both use the string as an identifier. Objects having the same id between Qaves can be linked together to exhibit the same properties at all times. This is normally the case when users run the same program or use the same object section of the configuration file. Table 4.2.2 describes the command list that the WorldServer uses.

Figure 4.1 shows an example of the message protocol used to register a number of Qaves and share data about objects. A typical scenario is the following. We have two Qaves, A and B. Assume that these Qaves want to share two objects, obj_1 and obj_2. Qave A and B register with the WorldServer by sending the messages **register QaveA** and **register QaveB** respectively. The WorldServer then fills the client list with information given by the Qaves. Qave A initiates the sharing of obj_1 by sending the message **share obj_1**. Qave B does the same with obj_2. At that point, Qave A and B share two objects, and **update** messages are sent to the WorldServer when the user of a Qave makes changes to either one of these objects. Each message is then broadcast by the WorldServer to all the registered Qaves.

A third Qave, C, registers. After having updated its client list, the WorldServer tells Qave C that two objects are already being shared and supplies Qave C with data updates about obj_1 and obj_2. The user of Qave C can then collaborate with the other users by manipulating obj_1 and obj_2.

It is important to note that in general the generation of update messages should be kept event-based (as opposed to being a constant stream of updates regarding all the objects for example) in order to limit the bandwith usage (DG10). On the other hand, a stream of updates would be transmitted during a continuous move to ensure the correct position of the object throughout the whole motion. Such a motion would indeed create a notable flow of data. In that case, network congestion might lead to a lower rate of update messages as the WorldServer would block waiting for the completion of the transmission to all clients. This in turn may lead to desynchronization between clients. Another option to minimize bandwidth would be to sample the continuous motion at discrete points in time and perform an interpolation between these points. This, however, might result in a loss of granularity in the gesture and create additional lag as the next update point is needed to perform the interpolation.

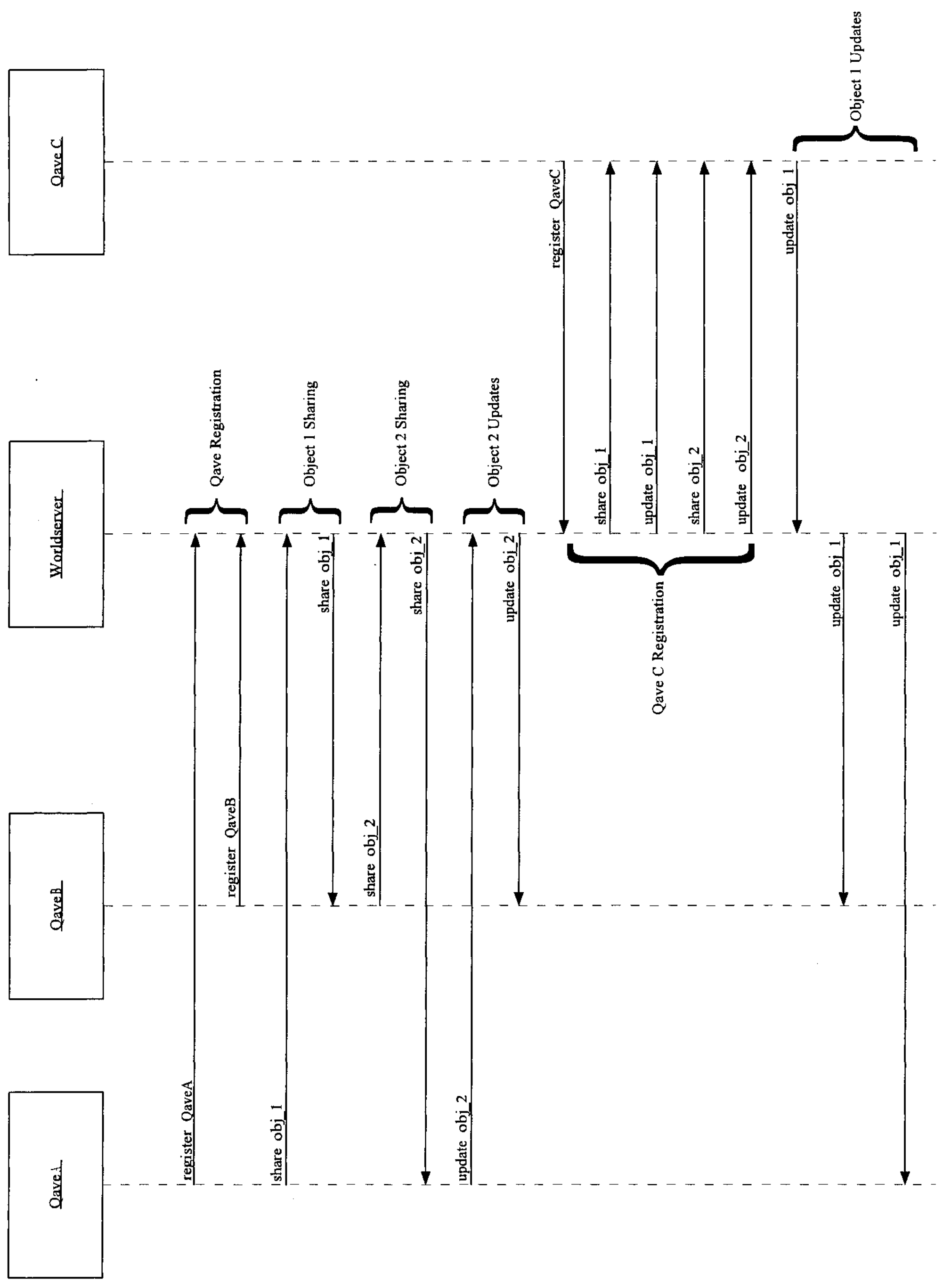

FIGURE 4.1. Communication Protocol

CHAPTER 5

Tracker and Spatialized Audio

"A gesture cannot be regarded as the expression of an individual, as his creation (because no individual is capable of creating a fully original gesture, belonging to nobody else), nor can it even be regarded as that person's instrument; on the contrary, it is gestures that use us as their instruments, as their bearers and incarnations."
—Milan Kundera

From the outset, one of our main design goals has been to avoid the introduction of any special equipment that must be worn by the user, for instance, stereo goggles, data gloves, and other tethered tracking devices. Instead, we aim to perform all our tracking and gesture analysis entirely through the use of video processing techniques. This allows for a more direct and immersive experience of the synthetic world.

The main role of the tracker is to provide information pertaining to user's gestures. In our current version of the tracker, this means data about one head and one hand. The head position data is used for the off-axis projection computations (see Section 3.1.3). It can also be sent out to a remote Qave, where the incoming sound stream can be spatialized thereby being coherent with the position of the avatar of the distant user.

The tracking system is covered in Section 5.1, while Section 5.2 describes the sound spatialization system.

5.1. Image Processing and Message Dispatching

Our tracking system is comprised of two distinct processes. The first one is in charge of the video processing (i.e. the actual tracking) while the other is in charge of dispatching the tracker data to clients. The dispatcher uses ServerLib to communicate with other applications. Our current user tracking algorithm is based on the view from a ceiling-mounted camera. It employs differencing between the current frame and an average reference image of the background.[1] The tracker can also forward processed video frames (i.e. background removed) to a Qave, where it will then be blended in the virtual scene. Both tracker and dispatcher were originally written by Stéphane Pelletier, and were adapted for use in this framework. Two main changes were applied.

First, the communication mode was set to synchronous between the Qave and the dispatcher. Because the tracking update rate is higher than the Qave framerate, positional updates could overwhelm the renderer in asynchronous mode. By making sure that the Qave sends out a reply message when an update has been received, we ensure that fresh data is transmitted while avoiding overflooding the socket buffer of the receiver.

Secondly, we wanted the possibility of transmitting the position of the head of the user to the audio spatialization system, where it could be used to compute the correct output for each speaker. The spatializer accepts data from the network in the Open Sound Control (OSC) format,[2] and a special OSC interface was implemented in the dispatcher to permit the transmission of data to the spatializer.

[1]The background image is generated prior to any users entering the space.
[2]See http://www.cnmat.berkeley.edu/OpenSoundControl/ for more details on OSC.

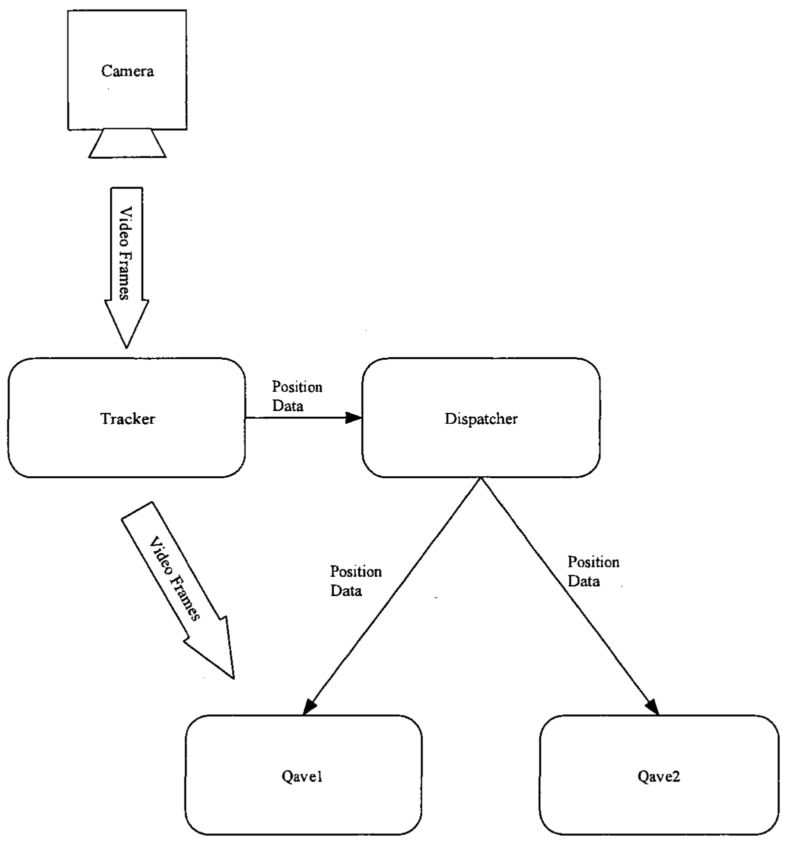

FIGURE 5.1. Tracker and Dispatcher Processes

Ultimately, head- and hand-tracking should be accomplished by multiple cameras distributed around the environment. So far, due to implementation limitations, we only use a single overhead camera, which, in conjunction with the tracker, allows for a single hand and a single head tracking. Since only one user can enjoy a correct perspective view at any time, being limited to single head detection is a fair assumption. Conversely, the ability to track multiple hands is an extension that could prove very beneficial, as bimanual interactions can greatly enrich the gestural vocabulary. Several studies [**47, 48, 49, 50, 51**] have illustrated the benefits of well-designed bimanual tasks. Not only does bimanuality aid in speed and efficiency, but the structure of the task can be made more complex without significant loss of performance. Kinesthetic feedback provided by the muscles in our arms and hands offers additional subconscious information to the user [**52**] so that less computer-generated feedback is required and in turn, the cognitive load imposed on the user is reduced.

With respect to our tracker, it is thus important to detect and follow multiple targets. As such, members of our research group are currently experimenting with new tracking algorithms, which could then subsume the present tracking system. Those algorithms notably involve CONDENSATION [**53**] and skin-color based detection.

5.2. Tracker Data for Spatialized Audio

The sonification of immersive environments is gaining an increasing amount of attention. Researchers acknowledge that effective immersion in a virtual environment implies more than filling the user's field of view with coherent data. As noted by Naef et al. [**54**], auditory perception offers a powerful complement to the visual channel. Gary Kendall et al. [**55**] describes the advantage of 3D sound as the quality of "being there." With 3D audio, one loses the feeling of having the sound being mediated by the speakers. Instead, one directly perceives the sonic environment.

As was mentioned, the SRE is equipped with eight speakers located at each vertex of the cube (see Figure 1.2). Multiple input sources[3] can then be spatialized. The spatialization process is performed by the LocalizerQ.[4] The LocalizerQ uses time delays, intensity differences and cross-talk cancellation to provide a 3D sound space. It consists of seven processing units (see Figure 5.2) :

(1) The Doppler simulator: performs pitch based shifting based on the acceleration of the sound source in relation to the listener

[3]For example, a typical sound source is a stream from a remote location.
[4]The LocalizerQ was developed by Zack Settel, http://www.Zeep.com.

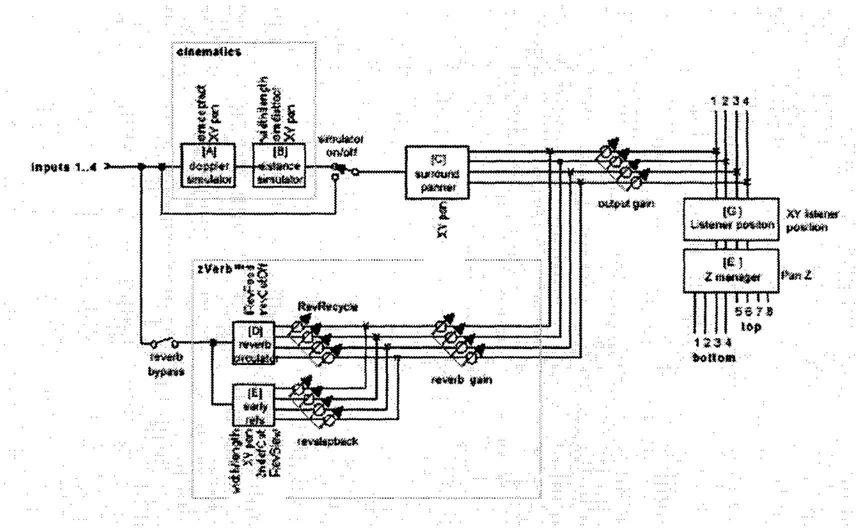

FIGURE 5.2. LocalizerQ DSP Architecture (with the authorization of Zeep.com)

(2) The distance simulator: performs amplitude attenuation based on the distance to the listener
(3) The surround panner: pans (equal-power) the simulated sound onto the speaker array
(4) The reverb engine: simulates a reverberant sound field and excites it with the simulated sound source
(5) The early reflections: calculated based on the actual position of the source
(6) The height manager: positions the sound in the z axis
(7) The listener position: controls the global output energy balance

Figure 5.3 presents an overview of how the spatializer integrates in the framework. The dispatcher can send the head position obtained from the tracker to the spatializer. The spatializer uses this information in the listener position processing unit, which creates the illusion of a static audio point source as the user moves about in the space.

/Pan/PanX,Y,Z	Controls the position of the sound source in 3D
/Pan/Length,Width	Controls the size of the virtual sound space
/Listener/PanX,Y	Indicates the position of the listener in the space
/Pan/OutputGain	Controls the overall volume of the sound source

TABLE 5.1. LocalizerQ Command List Sample.

The LocalizerQ receives command in the OSC format, and Table 5.2 shows a subset[5] of the parameters (in OSC format) that can be passed to the LocalizerQ.

The concept of spatialized sound is not new one, nor is its implementation. However, the ability to associate a 3D sound source to a dynamic object on the screen is an important feature for user engagement in the virtual world. As noted by Hollier et al., spatial audio has a vital role to play in enhancing speech communication performance and naturalness in synthetic spaces by giving directionality to individual sound sources and providing feedback about the virtual space in which the user is located [**56**]

[5]Other parameters that have not been listed here for conciseness control the reverberation factors of the virtual room.

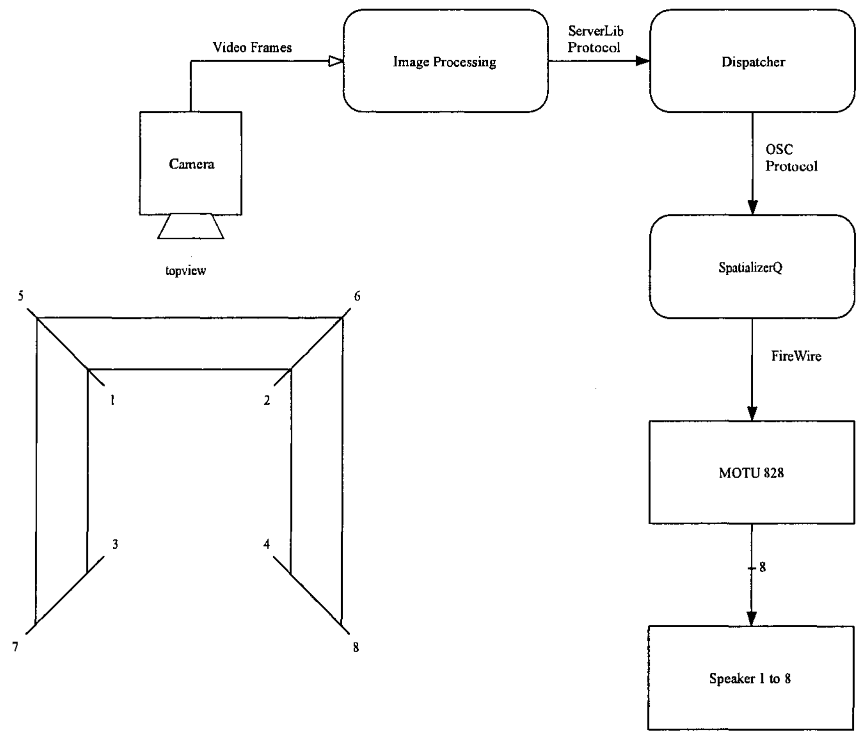

FIGURE 5.3. Spatializer Information Flow

CHAPTER 6

Conclusions and Future Work

6.1. Conclusions

The proposed framework has been successfully used both for our initial application and by the other members of our research group [**6, 5**]. The overall goals of flexible immersive rendering and the integration of coherence between multiple Qave clients were achieved, thereby allowing collaboration between users in remote immersive environments. Distant users can also be blended in the synthetic scene both through video and spatially coherent audio streams. This is an important feature that benefits computer mediated human-human interactions.

In particular, the Qave engine, which was specifically designed for the SRE framework, has proven to be faster than the previously used CAVELib, especially under Linux. Another important point is that the Qave engine only uses widely available APIs, and can be run on virtually any Unix-based platform. This allows Qave to run both our VR setups. Additionally, the engine architecture is simple enough that it can be extended by other researchers, which means that other components can be easily integrated in the engine. Furthermore, the simulation mode makes it faster and easier for the application programmers to see the progress of their work.

Finally, the inherent modularity of the framework means that any one component may be upgraded in the future without disrupting the rest of the system.

6.2. Future Work

The framework was designed to be flexible and easily upgradeable module by module. For the Qave engine, an interesting expansion would be to support more 3D file formats, especially those that include animation data. A logical extension would then be to add support for some sort of scripting language that could pre-define the interactions between objects. Further, our trackers need to be improved to detect and follow multiple targets with great accuracy. The integration of multiple camera views into an accurate 3D model would be very helpful in that task. Such a 3D model could be used to determine the posture of tracked users. Camera view integration in turn requires good background/foreground segmentation along with the precise calibration of the different cameras. Additionally, being able to recognize fine grained gestures involving, for example, hand rotations or finger movements would greatly enhance the range of possible actions of the users, and hence their control over the virtual scene. When detected, hand and finger postures could also be applied to the hand model, providing the user with a greater sense of familiarity with the synthetic world. These challenges all constitute active research projects in our laboratory.

Regarding the framework as a whole, the use of a faster networking library such, as Bronto,[1] could lead to better video framerate of the remote user and potentially faster message passing between clients. This in turn would result in a better sense of immersion for the user and hence in a potentially richer interaction between remote participants. Finally, as with all frameworks, the most interesting part is application development, which, we hope, will further knowledge in fields such as bimanuality in immersive environments and telepresence.

[1]Bronto is developed by Stephen P. Spackman, see http://ultravideo.mcgill.edu/technical/ for more details.

REFERENCES

[1] C. Ware, K. Arthur, and K. S. Booth, "Fish tank virtual reality," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 37–42, ACM Press, 1993.

[2] M. N. Hilario and J. Cooperstock, "Occlusion detection for front-projected interactive displays," in *Proceedings of Pervasive 2004*, (Vienna, Austria), April 2004.

[3] C. Jaynes, W. B. Seales, K. Calvert, Z. Fei, and J. Griffioen, "The Metaverse: a networked collection of inexpensive, self-configuring, immersive environments," in *Proceedings of the workshop on Virtual environments 2003*, pp. 115–124, ACM Press, 2003.

[4] I. Poupyrev, S. W. andd M. Billinghurst, and T. Ichikawa, "Egocentric object manipulation in virtual environments: Empirical evaluation of interaction techniques," in *Proceedings of Eurographics*, vol. 17, 1998.

[5] Y. Boussemart, F. Rioux, F. Rudzicz, M. Wozniewski, and J. R. Cooperstock, "A framework for 3D visualization and manipulation in an immsersive space using an untethered bimanual gestural interface," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM Press, November 2004.

[6] F. Rioux, F. Rudzicz, and M. Wozniewski, "The modellers' apprentice – the toolglass metaphor in an immersive environment," in *Proceedings of the 18th British HCI Group Annual Conference*, September 2004.

[7] I. Sutherland, "The Ultimate Display," in *Proceedings of IFIP Congress*, pp. 506–508, 1965.

[8] I. Sutherland, "A head mounted three dimensional display," in *IFIPS Conference Proceedings*, pp. 757–764, 1968.

[9] C. Cruz-Neira, D.J.Sandin, and T. DeFanti, "Surround-screen projection-based virtual reality: the design and implementation of the CAVE," in *Proceedings of SIGGRAPH '93*, pp. 135–142, ACM Press, 1993.

[10] B. Wei, C. Silva, E. Koutsofios, S. Krishnan, and S. North, "Visualization research with large displays," *IEEE Computer Graphics and Applications*, vol. 20, pp. 38–44, July-August 2000.

[11] D. Guilford, "Virtual Design: as clay fades, GM shifts toward digital imagery," July 2004. http://www.autoweek.com/.

[12] A. Gaitatzes, D. Christpoulos, A. Voulgari, and M. Roussou, "Hellenic cultural heritage," in *Proceedings of the 6th international conference on Virtual Systems and Multimedia*, 2000.

[13] S. J. Gibbs, C. Arapis, and C. J. Breiteneder, "TELEPORT – towards immersive copresence," *Multimedia Syst.*, vol. 7, no. 3, pp. 214–221, 1999.

[14] G. Goebbels and V. Lalioti, "Co-presence and co-working in distributed collaborative virtual environments," in *Proceedings of the 1st international conference on Computer graphics, virtual reality and visualisation*, pp. 109–114, ACM Press, 2001.

[15] M. Garau, M. Slater, V. Vinayagamoorthy, A. Brogni, A. Steed, and M. A. Sasse, "The impact of avatar realism and eye gaze control on perceived quality of communication in a shared immersive virtual environment," in *Proceedings*

of the conference on Human factors in computing systems, pp. 529–536, ACM Press, 2003.

[16] M. Deering, "High resolution virtual reality," in *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pp. 195–202, ACM Press, 1992.

[17] S. Feiner, B. Macintyre, and D. Seligmann, "Knowledge-based augmented reality," *Commun. ACM*, vol. 36, no. 7, pp. 53–62, 1993.

[18] T. Furness, *Harnessing Virtual Space*, pp. 4–7. Society for Information Display Digest, 1988.

[19] J. Butterworth, A. Davidson, S. Hench, and M. T. Olano, "3DM: a three dimensional modeler using a head-mounted display," in *Proceedings of the 1992 symposium on Interactive 3D graphics*, pp. 135–138, ACM Press, 1992.

[20] R. Pausch, T. Crea, and M. Conway, "A literature survey for virtual environments: military flight simulator visual systems and simulator sickness," *Presence: Teleoper. Virtual Environ.*, vol. 1, no. 3, pp. 344–363, 1992.

[21] R. Azuma and G. Bishop, "Improving static and dynamic registration in an optical see-through hmd," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 197–204, ACM Press, 1994.

[22] J. Liang, C. Shaw, and M. Green, "On temporal-spatial realism in the virtual reality environment," in *Proceedings of the 4th annual ACM symposium on User interface software and technology*, pp. 19–25, ACM Press, 1991.

[23] P. Richard, G. Burdea, G.Birebent, D. Gomez, N. Langrana, and P. Coiffet, "Effect of frame rate and force feedback on virtual object manipulation," *Presence: Teleoperators and Virtual Environments*, vol. 5, no. 1, pp. 95–108, 1996.

[24] E. Patrick, D. Cosgrove, A. Slavkovic, J. A. Rode, T. Verratti, and G. Chiselko, "Using a large projection screen as an alternative to head-mounted displays

for virtual environments," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 478–485, ACM Press, 2000.

[25] E. Lantz, "The future of virtual reality: head mounted displays versus spatially immersive displays (panel)," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 485–486, ACM Press, 1996.

[26] M. Czernuszenko, D. Pape, D. Sandin, T. DeFanti, G. L. Dawe, and M. D. Brown, "The ImmersaDesk and Infinity Wall projection-based virtual reality displays," *SIGGRAPH Comput. Graph.*, vol. 31, no. 2, pp. 46–49, 1997.

[27] W. Kruger, C.-A. Bohn, B. Frohlich, H. Schuth, W. Strauss, and G. Wesche, "The Responsive Workbench: a virtual work environment," *Computer*, vol. 28, no. 7, pp. 42–48, 1995.

[28] B. Ullmer and H. Ishii, "The MetaDESK: Models and prototypes for tangible user interfaces," in *ACM Symposium on User Interface Software and Technology*, pp. 223–232, 1997.

[29] J. Leigh, A. E. Johnson, T. A. DeFanti, and M. D. Brown, "A review of tele-immersive applications in the CAVE research network," in *VR*, pp. 180–, 1999.

[30] Paul Rajlich, "CAVEQuake." http://brighton.ncsa.uiuc.edu/~prajlich/caveQuake/.

[31] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira, "VR Juggler: a virtual platform for virtual reality application development," in *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, p. 89, IEEE Computer Society, 2001.

[32] R. Pausch, "ALICE." http://www.alice.org/.

[33] H. Tramberend, "Avocado – a distributed virtual environment framework," 1999.

[34] R. Held and N. Durlach, "Telepresence, time delay and adaptation," pp. 232–246, 1993.

[35] R. Samanta, J. Zheng, T. Funkhouser, K. Li, and J. P. Singh, "Load balancing for multi-projector rendering systems," in *HWWS '99: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pp. 107–116, ACM Press, 1999.

[36] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin, "Net Juggler: running VR Juggler with multiple displays on a commodity component cluster," in *Proceedings of the IEEE Virtual Reality Conference 2002*, p. 273, IEEE Computer Society, 2002.

[37] W. Gropp and E. Lusk, "The Message Passing Interface (MPI) standard.," 1999. http://www-unix.mcs.anl.gov/mpi/.

[38] J. Allard, V. Gouranton, G. Lamarque, E. Melin, and B. Raffin, "Softgenlock: active stereo and genlock for PC cluster," in *Proceedings of the Joint IPT/EGVE'03 Workshop*, (Zurich, Switzerland), May 2003.

[39] A. Bierbaum and C. Cruz-Neira, "ClusterJuggler: A modular architecture for immersive clustering," in *VR-Cluster'03- Workshop on Commodity Clusters for Virtual Reality*, IEEE Virtual Reality Conference 2003, March 2003.

[40] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, "Chromium: a stream-processing framework for interactive rendering on clusters," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pp. 693–702, ACM Press, 2002.

[41] M. Gross, S. Wurmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. V. Moere, and O. Staadt, "blue-c: a spatially immersive display and 3D video portal for telepresence," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 819–827, 2003.

[42] K. Isakovic, T. Dudziak, and K. Kchy, "X-rooms," in *Proceeding of the seventh international conference on 3D Web technology*, pp. 173–177, ACM Press, 2002.

[43] L. P. Soares and M. K. Zuffo, "JINX: an X3D browser for VR immersive simulation based on clusters of commodity computers," in *Proceedings of the ninth international conference on 3D Web technology*, pp. 79–86, ACM Press, 2004.

[44] O. A. R. Board and D. Shreiner, *OpenGL 1.4 reference manual (Blue Book).* Addison Wesley Professional, 2004.

[45] E. Frécon and M. Stenius, "Dive: A scaleable network architecture for distributed virtual environments.," *Distributed Systems Engineering Journal*, vol. 5, pp. 91–100, September 1998. Special Issue on Distributed Virtual Environments.

[46] R. K. Jain, *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley, April 1991.

[47] W. Buxton and B. Myers, "A study in two-handed input.," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, vol. 27, pp. 321–326, 1986.

[48] K. Hinckley, R. Pausch, D. Proffitt, and N. Kassell, "Two-handed virtual manipulation," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 5, no. 3, pp. 260–302, 1998.

[49] A. Leganchuk, S. Zhai, and W. Buxton, "Manual and cognitive benefits of two-handed input: An experimental study," *ACM Transactions on Computer-Human Interaction*, vol. 5, no. 4, pp. 326–359, 1998.

[50] P. Kabbash, I. MacKenzie, and W. Buxton, "Human performance using computer input devices in the preferred and non-preferred hands," in *Proceedings of SIGCHI conference on Human factors in computing systems*, pp. 474–481, 1993.

[51] R. Balakrishnan and G. Kurtenbach, "Exploring bimanual camera control and object manipulation in 3D graphics interfaces.," in *Proceedings of ACM CHI 1999 Conference on Human Factors in Computing Systems*, pp. 56–62, ACM Press, 1999.

[52] R. Balakrishnan and K. Hinckley, "The role of kinesthetic reference frames in two-handed input performance," in *ACM Symposium on User Interface Software and Technology*, pp. 171–178, 1999.

[53] M. Isard and A. Blake, "CONDENSATION – conditional density propagation for visual tracking," 1998.

[54] M. Naef, O. Staadt, and M. Gross, "Spatialized audio rendering for immersive virtual environments," in *Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 65–72, ACM Press, 2002.

[55] G. Kendall, "Directional hearing and stereo reproduction," October 2002. http://music.northwestern.edu/classes/3D/pages/sndPrmGK.html.

[56] M. P. Hollier, A. N. Rimell, and D. Burraston, "Spatial audio technology for telepresence," pp. 40–56, 1999.