2004

Dynamic Shared State Maintenance In Distributed Virtual Environments

Felix George Hamza-Lup
University of Central Florida

DYNAMIC SHARED STATE MAINTENANCE IN DISTRIBUTED VIRTUAL ENVIRONMENTS

by

FELIX GEORGE HAMZA-LUP
B.S. Technical University of Cluj-Napoca, 1999
M.S. University of Central Florida, 2001

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2004

ABSTRACT

Advances in computer networks and rendering systems facilitate the creation of distributed collaborative environments in which the distribution of information at remote locations allows efficient communication. Particularly challenging are distributed interactive Virtual Environments (VE) that allow knowledge sharing through 3D information.

In a distributed interactive VE the dynamic shared state represents the changing information that multiple machines must maintain about the shared virtual components. One of the challenges in such environments is maintaining a consistent view of the dynamic shared state in the presence of inevitable network latency and jitter. A consistent view of the shared scene will significantly increase the sense of presence among participants and facilitate their interactive collaboration.

The purpose of this work is to address the problem of latency in distributed interactive VE and to develop a conceptual model for consistency maintenance in these environments based on the participant interaction model.

A review of the literature illustrates that the techniques for consistency maintenance in distributed Virtual Reality (VR) environments can be roughly grouped into three categories: centralized information management, prediction through dead reckoning algorithms, and frequent state regeneration. Additional resource management methods can be applied across these techniques for shared state consistency improvement. Some of these techniques are related

to the systems infrastructure, others are related to the human nature of the participants (e.g., human perceptual limitations, area of interest management, and visual and temporal perception).

An area that needs to be explored is the relationship between the dynamic shared state and the interaction with the virtual entities present in the shared scene. Mixed Reality (MR) and VR environments must bring the human participant interaction into the loop through a wide range of electronic motion sensors, and haptic devices. Part of the work presented here defines a novel criterion for categorization of distributed interactive VE and introduces, as well as analyzes, an adaptive synchronization algorithm for consistency maintenance in such environments.

As part of the work, a distributed interactive Augmented Reality (AR) testbed and the algorithm implementation details are presented. Currently the testbed is part of several research efforts at the Optical Diagnostics and Applications Laboratory including 3D visualization applications using custom built head-mounted displays (HMDs) with optical motion tracking and a medical training prototype for endotracheal intubation and medical prognostics. An objective method using quaternion calculus is applied for the algorithm assessment. In spite of significant network latency, results show that the dynamic shared state can be maintained consistent at multiple remotely located sites. In further consideration of the latency problems and in the light of the current trends in interactive distributed VE applications, we propose a hybrid distributed system architecture for sensor-based distributed VE that has the potential to improve the system real-time behavior and scalability.

To my parents, my grandmothers who have supported me all the way since the beginning of my studies, and in loving memory of my grandfathers.

To my friends. You keep my spirit alive!

To all those who believe in the richness of learning.

ACKNOWLEDGEMENTS

GOD created us and when doing it He endowed us with both the ability to reason and that to feel. He gifted us with ‘a brain’ enabling us to think, to learn, to remember and to perceive the world around us, and He gifted us with ‘a heart’ which helps us love and care. Well equipped with brains and heart we strive along our life path to quench our thirst of knowledge, to make our dreams come true, but also to fulfill our commitments and carry out our mission on Earth. What are we here for if not to enjoy, through our Savior, life eternal, solve what problems we can, give light, peace and joy to our fellowmen, and leave this dear planet a little better and healthier than when we were born. These have been the guiding principles in my life and at this most important moment of it I would like to take the opportunity to thank, with all my heart and soul the following honorable people for having supported me in the pursuit of achieving my ideals.

My deepest gratitude is now directed to:

- My advisors, Professor Jannick P. Rolland and Professor Charles E. Hughes, who have guided me throughout my research and challenged me to expand my point of view and see “the bigger” picture.
- My committee members: Michael Moshell, Frank Biocca and Kien Hua for their comments and suggestions and for treating me with respect and timely advice.
- My wonderful parents, Georgeta “Mica” and Lucian “Ticu”, who deserve a vast amount of credit for supporting me both emotionally and financially throughout my education and

life, and most of all for believing in my intellectual capacity. Their encouragement has helped me to find the path in the difficult moments when "hope was a lost friend".

- Nicoleta, all of your ceaseless support moving to Florida is deeply appreciated and my sister, Georgiana, you have always been supportive, caring, and helpful. Both of you have been by my side and patiently shared with me the shiny days of peace and calmness and those full of worries and restlessness.

To my dear friends from USA who, from behind the scenes, have encouraged and supported my endeavor and my work. I am grateful to them for investing time and energy discussing ideas with me and tolerating my many digressions.

- Larry Davis Jr., for his friendship and understanding.
- Marc Smith, for his kind advice and friendship.
- Paulius Micikevicius, for healthy and constructive criticism and honesty.
- ODALab members, past and presence, for assistance at various stages in my work.

I will never forget my mentors and friends from Romania, especially:

- Professor Ioan Salomie for his advice and guidance during my undergraduate research at the Technical University of Cluj-Napoca.
- Professor Kalman Pusztai for his extraordinary dedication in pushing the research forward at all levels.

And last but not least, my family and friends from Oradea and Cluj-Napoca who have made me wish to turn back time and live again so many unforgettable moments:

- My cousin Ioan Abrudan "Nelu" for his guidance in life and for his unreserved friendship and love of nature, for his inspiration and trust in the unmatched beauty of Transilvania's forests. I recall now our philosophical escapades on the hills of Totoreni near Beius. Thank you for taking care of a little "gibonel" and for reciting Mihai Eminescu with such a passion.
- My uncles Ghita and Cornel, my aunts Florica and Eugenia, my dear cousins Marius, Adelina and Monica, my nephew Horatiu who have all surrounded me with their love.
- My grandparents Tati and Bunu (God rest them in peace), Mami and Buni, for loving me unconditionally and for giving me beautiful childhood memories. You have a very special place in my heart! God bless you.
- To anybody I missed who deserves a mention.

All these people have been part of my life and the dear thought of them brings into my mind William Shakespeare's memorable lines:

"*What a piece of work is man! How noble in reason, how infinite in faculties, in form and moving, how express and admirable in action… THE BEAUTY OF THE WORLD*."

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AOI	Area Of Interest
AR	Augmented Reality
ARC	Augmented/Artificial Reality Center
ASA	Adaptive Synchronization Algorithm
AV	Augmented Virtuality
CBT	Core-Based Tree
CORBA	Common Object Request Broker Architecture
CPO	Control Package Object
CRT	Cathode Ray Tube
CSCE	Computer Supported Cooperative Environment
DARE	Distributed Augmented Reality Environment
DIVE	Distributed Interactive Virtual Environment
DVS	Distributed Virtual System
ETI	EndoTracheal Intubation
ETT	EndoTracheal Tube
HPS	Human Patient Simulator
HMD	Head Mounted Display
IPD	InterPupilary Distance
IRED	InfraRed Emitting Diode
ISO	International Standard Organization
LAN	Large Area Network
LCD	Liquid Crystal Display
LOD	Level Of Detail
MASSIVE	Model Architecture and System for Spatial Model of Communication
MIMD	Multiple Instruction Multiple Data
MOM	Message Oriented Middleware

MPI	Message Passing Interface
MR	Mixed Reality
NTP	Network Time Protocol
OLED	Organic Light Emitting Display
OMG	Object Management Group
OOM	Object Oriented Middleware
ORB	Object Request Broker
OS	Operating System
OSI	Open System Interconnection
P2P	Peer-To-Peer
PARADISE	Performance Architecture for Advanced Distributed Interactive Simulation Environments
QoS	Quality of Service
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RT	Real Time
RTOS	Real-Time Operating System
THMD	Teleportal Head-Mounted Display
TCP	Transport Control Protocol
UDP	User Datagram Protocol
VE	Virtual Environment
VR	Virtual Reality
WAN	Wide Area Networks

1. INTRODUCTION

Computers and computer networks are the main ingredients in the current development of powerful information systems bound to be completely transformed in the near future into knowledge sharing systems.

Knowledge is embedded in people and unlike information, knowledge creation occurs in a process of social interaction. As our service-based society is evolving into a knowledge-based society, there is an acute need for more effective collaboration and more effective knowledge sharing systems for use by geographically scattered people. The current communication systems are limited by physical factors (e.g., signal propagation time, noise) and make distributed interactive application challenging. The heterogeneity of the application deployment environment, the packet loss rate, the bandwidth limitations, traffic collisions and congestions are several factors that influence the latency.

1.1 Remote Collaboration

Current technological advances have led to an increased interest in distributed collaborative environments. These environments have the potential to significantly change the way activities (e.g. research, business, education etc.) are carried out. The latest trends in distributed

collaboration technologies allow people to move across organizational boundaries and to collaborate with others within/between organizations and communities.

"Collaboration" is a broad area of research involving wide-reaching issues such as knowledge representation, interaction methods, and many others. Through collaboration individuals gain maximum benefit from the community of users who share similar goals.

Technological advances in optical projection and computer graphics allow us to augment reality with computer generated three-dimensional objects. Moreover the distribution of these three-dimensional objects at dispersed locations allows efficient communication of ideas and concepts. A distributed interactive virtual environment (VE) can enhance the level of this communication by transforming current computer networks into navigable and populated 3D spaces.

As the pioneer of computer supported collaborative environments, D. C. Engelbart mentioned [1]: *"Three people working together in this augmented mode seem to be more than three times as effective in solving a complex problem as is one augmented person working alone--and perhaps ten times as effective as three similar men working together without this computer-based augmentation. It is a new and exhilarating experience to be working in this independent-parallel fashion with some good men. We feel that the effect of these augmentation developments upon group methods and group capability is actually going to be more pronounced than the effect upon individuals methods and capabilities."*

This research concentrates on a subset of computer supported collaborative environments, particularly environments in which collaboration is achieved through interactive mixed and virtual reality paradigms i.e. distributed interactive VE. The application domain for these environments range from entertainment and business to engineering and medicine including the entire virtuality continuum [2] and evolving into potential infospaces [3].

1.2 Synchronicity - Consistent Dynamic Shared State

Have you ever been amazed by the perfect flight formation of a flock of birds? Where does the power of a set of small entities come when they act together as one, at a particular moment in time? Synchronicity is the answer and the examples are unlimited.

Restricting the synchronization application domain to remote collaboration and furthermore to distributed interactive VE we can replace the synchronization paradigm with another one, the ***consistent dynamic shared state***. From a virtual environment perspective the ***dynamic shared state*** constitutes the changing information that multiple, distributed machines must maintain about the shared virtual components of the environment. From a distributed systems perspective consistency is an inherent problem due to data replication. Several consistency models have been proposed in the literature and will be investigated on the course of this work.

Now, let's briefly analyze these words: "interactive remote collaboration". The fact that we talk about "collaboration" implies that two or more entities will be involved in the experience. From

the perspective of this work we will restrict these entities to human beings, which bring in important issues related to human factors (e.g. perceptual, conceptual, and motor cycle time) [4]. "Remote" comes into play and brings in the main advantages and the motivations for building such systems (e.g. reduced travel time and costs, as well as reduced risks). "Interactive" implies that each participant is able to make its actions visible to the other participants in real time and hence the need for consistency. Actions based on inconsistency judgments are undoubtedly inconsistent.

1.3 Inconsistency Factors

The interactive and dynamic nature of a collaborative VE is constrained by many factors including latency. Latency generators in distributed interactive VE can be roughly grouped in two categories: ***computing system*** latency and ***network infrastructure*** latency.

In a VE, in describing the equipment (e.g. head-mounted display) that provides stereoscopic visualization and body parts tracking, the latency is increased with the time elapsed from detecting the body part motion to the time the appropriate image is displayed on the appropriate interface. The computing system latency includes rendering delays (e.g. image-generation delay, video sync delay, frame delay and internal display delay), mismatches in data speed between the microprocessor and input/output devices, sensor delays (e.g. tracker delays) and inadequate data buffers [5]. However, rapid advances in hardware technology are making computing system

latency much smaller than the one caused by the network infrastructure. Hence, one of our focuses is reducing the network's contribution to potential inconsistencies.

In a network, latency a synonym for *delay*, is an expression of how much time it takes for a packet of data to get from one designated point to another. The contributors to network latency include:

- *Propagation time*: This is simply the time it takes for a packet to travel between one place and another, close to the speed of light. Today and probably in the near future it will be impossible for the signal to travel at the speed of light. Even if that would be possible it would take 16 ms for the light to make the trip from coast to coast.
- *Transmission time*: The medium itself (whether optical fiber, wireless, or some other) introduces some delay. The size of the packet introduces delay in a round trip since a large packet will take longer to receive and return than a short one.
- *Router and other processing times*: Each gateway node takes time to examine and possibly change the header in a packet (e.g. changing the hop count in the time-to-live field).
- *Other computer and storage delays*: Within networks at each end of the journey, a packet may be subject to storage and hard disk access delays at intermediate devices such as switches and bridges. (in backbone statistics, however, this kind of latency is probably not considered.)

Table 1 gives the MCI (UUnet) average backbone latency between US and different countries for the past year [6].

Table 1. MCI Backbone Latency Statistics in (ms)

	2004	2003						
	January	December	November	October	September	August	July	June
Hong Kong to US	196.031	200.98	179.865	168.785	153.485	157.28	159.3	152.79
Singapore to US	214.553	206.62	208.585	217.085	211.80	207.80	207.075	206.375
Australia to US	165.816	169.49	173.155	160.55	162.615	163.27	163.935	163.69
Panama to US	59.997	66.45	66.27	68.51	64.99	64.19	69.69	72.9
Argentina to US	147.991	150.37	151.65	155.36	153.13	153.46	151.07	150.89
Chile to US	114.597	118.76	125.07	119.18	120.26	120.43	148.64	169.17

Factors that affect latency are the network infrastructure *bandwidth*, *traffic congestion*, *error rates* and *communication protocol characteristics*.

The *theoretical bandwidth* is the maximum capacity of a communication line. While theoretical peak bandwidth is fixed, actual or effective bandwidth varies and can be affected by high latencies. Too much latency in a short period of time can create a bottleneck that prevents data from "filling the pipe", thus decreasing effective bandwidth.

Another latency factor is congestion. In the points of congestion, packets will be stored for limited periods of time in the communication infrastructure (i.e. buffers) and sometimes dropped

if the storing space is exceeded. The latency resulting in such cases is very hard to predict and compensate.

Latency is affected by the signal-noise ratio which translates to errors. Errors influence the packet loss at different levels from the physical level up to the application level in the protocol stack. Error detection and correction algorithms are implemented at different levels and have a major impact on latency.

The higher level protocols used for communication influence the latency too. Over IP networks the UDP (User Datagram Protocol) simulates a packet switching connection with best effort. The packets are sent from source to destination without acknowledgement, thus they may arrive out of order or they might not arrive at all (packets may take different paths to arrive at the destination). Some applications can cope with this scenario (e.g. video streaming, chat etc.) The TCP simulates circuit switching (similar to a phone connection). Once the connection is established all packets from source to destination follow the same path and each packet is acknowledged by the destination. If a packet is not acknowledged it must be resent. TCP provides a more reliable connection at the expense of latency if the connection between the source and destination is not reliable.

Closely related to the network latency is the network *jitter* which refers to the variation in network latency. Under high jitter conditions, packets are not received at a steady rate at the destination even if they were transmitted at fixed intervals from the source.

1.4 Motivation

Envision a world where people from remote locations actively participate in a live three-dimensional experience instead of just watching a broadcast. Imagine being able to learn concepts by manipulating three-dimensional objects that represent those concepts.

One of the challenges in distributed interactive VEs is maintaining a consistent view of the shared state. It is only recently that researchers have begun to examine systematically the effects of consistency on the sense of presence. A consistent view in a shared scene may facilitate participants interaction and thus significantly increase the sense of presence among them [7, 8]. One way in which interaction is related to presence is its ability to strengthen participant's attention and involvement [9].

Virtual Reality (VR) environments and more importantly Mixed Reality (MR) (i.e. including Augmented Reality) environments must bring in the loop human users which interact through a wide range of electronic motion sensors and devices. This work aims at improving distributed interactive VE spanning the entire virtuality continuum by proposing a novel criterion for categorization of distributed VE applications as well as an algorithm and a distributed system architecture for ameliorating the effects of latency.

1.5 Research Summary

We address the problem of latency in distributed interactive VE and develop a conceptual model for consistency maintenance in these environments based on the participant's interaction model.

Our specific objectives are:

- A classification of the distributed interactive virtual (MR/VR) applications based on the participant interaction frequency.
- The design, implementation and assessment of a distributed dynamic shared maintenance algorithm for latency compensation.
- A hybrid distributed system architecture and data distribution scheme for interactive VE containing real-time sensors.

1.6 Dissertation Overview

Chapter 2 gives an overview of distributed systems paradigms and of the technologies for building VEs. We advance the notion of dynamic shared state in distributed VR environments and the related research efforts. A brief description of the fundamental models and paradigms for distributed systems is provided in Section 2.1. The most relevant architectural systems models are further investigated. Section 2.2 concentrates on the technologies involved in the development of VEs. The state of the art in three dimensional display devices and tracking

systems as well as the challenges for building interactive VEs are synthesized. The research towards shared state maintenance in distributed interactive VEs has passed through various stages in its development. Section 2.3 contains a survey of the dynamic shared state maintenance techniques for VE.

A theoretical formulation of the consistency problem within a distributed interactive simulation is presented in Chapter 3 which leads to a broader conceptualization of the consistency paradigm and a novel categorization criterion for participant interaction in distributed VEs. We analyze a distributed interactive environment from two perspectives: the application perspective Section 3.1 and the overall system perspective Section 3.2. A classification of the interactive VE applications is proposed based on the participants' interaction patterns. The proposed adaptive synchronization algorithm follows as well as a conceptual architecture for sensor-based distributed interactive VE.

Chapter 4 is dedicated to the hardware and software implementation and starts with a system overview in Section 4.1, followed by the testbed hardware and software description in Sections 4.2 and 4.3, respectively.

The experimental design and setup is described in Chapter 5 followed by the experimental results and analysis in Chapter 6.

The research contributions, the implications and limitations of the work as well as proposed directions for further research are summarized in Chapter 7, together with a medical training application prototype that would benefit from this work.

2. BACKGROUND AND RELATED WORK

2.1 Distributed Systems

A distributed system is a collection of (possibly heterogeneous) nodes whose distribution is transparent to the user so that the system appears as one local machine. The distribution transparency is in contrast to a network, where the user is aware that there are several machines, and their location, storage replication, load balancing and functionality are not transparent.

It is hard to capture all aspects of a distributed system in one definition. A more feasible approach is to discuss distributed systems referring to specific characteristics of distribution. One characteristic is the presence of a computer network. Such a system is built up from components that communicate and coordinate their actions by passing messages. The main characteristics of distributed systems are: openness, resource sharing, concurrency, modularity, scalability, transparency and graceful degradation [10]. The concurrency of components, the lack of a global clock and the independent failure of components relate the concept of distribution to disciplines such as fault-tolerance, security, real-time, system management.

As distributed networks become faster and parallel machines tend to look more like fault-tolerant distributed systems, distributed computing and parallel computing tend to be viewed as one concept characterized by concurrency.

2.1.1 Interaction Models

Interacting processes perform all the activity in a heterogeneous distributed system [11]. At each point in time there are an arbitrary number of active processes over a fixed or variable number of physical nodes. Each of these processes has a set of data that it can access. The communication performance and the lack of a global clock are the most important limiting factors that have an impact on the processes' interactions. Two important interaction models can be distinguished: synchronous and asynchronous. We can also consider a hybrid, quasi-synchronous model.

Synchronous distributed systems have several important constraints [12]:

- Each message exchanged among the distributed system's nodes is received within a known bounded time
- The time to execute each step of a process has a known lower and upper bound
- There is a local clock for each process with a known drift rate from the real time due to hardware differences.

At the other end on an *asynchronous* distributed system:

- There is no bound on the message transmission delays
- From the process execution point of view each process may take an arbitrary amount of time
- There are no bounds on the clocks' drift rates.

A synchronous distributed system has several advantages over the asynchronous one. First the processes are kept in synchronization with each other; they are progressing in a lockstep fashion. Second, more robust algorithms can be designed since the level of determinism is higher than in an asynchronous system. On the other end the asynchronous design is an attractive model because it has no timing constraints. Timing constraints are sometimes hard to meet because of the infrastructure limitations (e.g. limited network bandwidth and network latency). However, the weakness of the model consists in the fact that it does not allow interaction in real-time. Quasi-synchronous design is a hybrid design in which synchronization does not occur at a fixed time interval as in the fully synchronous one. Each node runs asynchronously until it needs to coordinate its activity with another node. At this point synchronization is necessary.

Since the research described in this dissertation revolves around distributed interactive virtual environments the *synchronous* and the *quasi-synchronous* design are of particular interest.

2.1.2 Coordination Models

Besides interaction, another fundamental model of a distributed system is the coordination model. The coordination model is closely linked with several characteristics of a system like scalability, architecture and performance. A distributed system brings the advantages of data and/or computation distribution. Any computation aspect is transparently viewed by the user as a service. While centralized services, data and algorithms impose scalability limitations [13], distribution ameliorates these problems at the expense of increased system complexity.

Centralized services are implemented by means of a single server running on a specific node in the system. Obviously the server will become a bottleneck as the number of users grows. *Centralized data* has the same effect on the scalability potential of a system. In a large distributed system, a significant number of messages are routed over many lines. From a theoretical point of view, the optimal way is to collect information about the load on all nodes and lines, and then run a graph theory algorithm to compute the optimal routes. However collecting and transporting all the input and output information would overload the system and would lead to performance degradation. *Distributed services and data* approach allows the improvement of the scalability attribute of a system. However in this case, the ***coordination model*** is the source of the increased system complexity.

A taxonomy of the coordination models for mobile agents is given by [14]. The taxonomy can be applied on a general distributed system if we view each process as being an agent.

Table 2. Coordination Models

		Temporal	
Referential		**Coupled**	**Uncoupled**
	Coupled	Direct	Mailbox
	Uncoupled	Meeting -oriented	Generative communication

- In the direct coordination model the processes are coupled from the referential and temporal aspect, i.e. the processes that are communicating know about each other and they run at the same time. This is the "strongest" coordination model.

- The Mailbox coordination model allows temporal uncoupling of processes, i.e. even if the processes know about each other, they do not have to run at the same time hence they exchange messages in a mailbox fashion.
- Meeting based systems are usually implemented using events in a publish/subscribe fashion, i.e. even if the processes do not know about each other (are anonymous), they participate at the same time in a communication session.
- Generative communication is the "weakest" coordination model from the referential and temporal point of view. Generative communication was introduced in the Linda programming environments [15, 16]. In this case processes make use of a shared persistent data space of tuples. Later IBM released an implementation of the tuple-space IBM T-Spaces [17]. Other implementations are JavaSpaces [18] and GigaSpaces [19] sometimes referred as space based middleware.

A distributed interactive VE will most likely use a direct (strong) model in which coordination is achieved through message passing.

2.1.3 Consistency Models

Data replication is one of the most important issues in distributed systems. Particularly in a distributed VE data are replicated to enhance reliability and improve performance. Two issues arise, the first issue is related to the actual distribution of updates, which concerns placement of replicas and the second issue is how replicas are kept consistent.

2.1.3.1 Data-Centric Consistency Models

Data-centric consistency models aim at providing a system wide consistent view on a data store on which several concurrent processes may perform updates. The consistency models range from *strict consistency*, where any "read" operation on a data item returns a value corresponding to the result of the most recent "write", to *weak consistency* which enforces consistency on a group of operations not on individual "reads" and "writes".

An important model in-between is the *causal consistency* [20] which makes a distinction between events that are potentially causally related and those that are not. In other words, actions (i.e. "writes") that are potentially causally related must be seen by all participants in the same order. Concurrent actions (i.e. "writes") may be seen in a different order by different participants. Implementing causal consistency requires keeping track of which participants (i.e. processes) have seen which actions (i.e. "writes"). It means that a dependency graph of which action is dependent on which other action must be constructed and maintained.

2.1.3.2 Client-Centric Consistency Models

Client-centric consistency models originate from the work on the Bayou database system [21]. In essence, this model of consistency provides guarantees for a single client concerning the consistency of access to a data store by that client. No guarantees are given concerning concurrent accesses by different clients. The client centric model enhances the *eventual*

consistency using "Monotonic Reads and Writes" as well as "Writes Follow Read" and "Read Your Writes" consistency. Since the constraints on consistency are relaxed the client centric consistency model may be used in a mobile environment.

2.1.4 Architectural System Models

The architectural models are closely linked with the coordination and consistency models. The strong coordination model is usually implemented on client-server architectures while the weak coordination model is associated with the atomistic peer-to-peer architectures. The diversity of applications makes developers take an architectural approach that falls in between these ranges.

In Chapter 3 we propose a novel architecture and data distribution scheme for sensor-based distributed interactive VE that falls in-between the atomistic peer-to-peer model and the traditional client-server model. Each node is autonomous and fully manages its resources and connectivity. The dynamic behavior of the nodes can be dictated by the participants that manipulate the virtual components of the environment through these sensors (e.g. motion tracking sensors).

In what follows we briefly discuss the most popular architectural approaches. The scope of this discussion is to give an overview of the advantages and disadvantages brought by different architectures as they pertain to the development of distributed interactive applications.

2.1.4.1 From Client-Server to Distributed Object Model

An ideal system architecture does not exist; it is usually driven by the application domain and users requirements. The Information Technology (IT) industry has been practicing a simple form of client/server (C/S) computing since the initial foundation of the mainframe. A mainframe host and a directly connected terminal represent a one-tier C/S system. In the two-tier C/S architecture, the client communicates directly with the server, commonly a database server. The application or business logic either resides on the client side or on the database server in the form of stored procedures. Discussions on the C/S model can be found in [22].Two-tier C/S model first began to emerge with the applications developed for local area networks in the late eighties and early nineties, and was primarily based upon simple file sharing techniques. C/S based systems for file sharing like: SUN Network file Systems, Andrew File Systems (AFS) as well as a variation of AFS called CODA, developed at Carnegie Mellon University are compared in [13].

Fat Clients and Fat Servers

The two-tier model initially involved a non-mainframe host, (a network file server) and an intelligent "fat" client where most of the processing occurred. This configuration did not scale well however it was enough to facilitate large or even mid-size information systems. With the emergence of the Graphical User Interfaces (GUIs), client component became more complex ("fat") and required additional computational power. Moreover, the network "footprint" using fat

clients, became very large affecting the scalability of the systems running on a limited bandwidth.

An alternative "thin" client / "fat" server configuration was introduced. The user invokes procedures stored at the database server. The "fat" server model, is more effective in gaining performance, because the network footprint, although still heavy, is lighter than the fat client approach. The down side of this approach is the stored procedure dependency. This dependency had a negative impact on the business logic's flexibility since a change in the business logic implies changes in each database containing the procedure. Two-tier (C/S) systems could not scale beyond several hundred users. Overall, these architectures are typically not well suited for distributed interactive applications.

The N-Tire Model, Middleware

As the C/S model continued to evolve, more sophisticated multi-tier solutions appeared where client-side computers began to operate as both clients and servers. Currently, the industry appears to be rapidly moving toward N-Tier architectures. N-Tier computing accomplishes a synergistic combination of computing models, by providing centralized common services in a distributed environment. The architecture typically leans heavily upon object oriented methodologies to gain as much flexibility and interchangeability as possible. The distributed object systems represent the ultimate generalization of the C/S model [23] and a multitude of frameworks and tools have been proposed and have evolved in what is known today as *middleware*.

Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. In a way, middleware has evolved from a software point of view in the same manner as the client-server applications have evolved from 2-tier to N-tier architectures. This evolution was necessary for supporting the development of complex applications on top of heterogeneous networks. Middleware can be divided into three classes:

- ***Event-Based Middleware*** where the focus is on distributed systems exhibiting event-based architectural style. This class is particularly suited to the construction of non-centralized distributed applications that must monitor and react to changes in their environment. A taxonomy of the event-based middleware can be found in [24] and descriptions of the issues with event-based middleware are presented in [25] .
- ***Message Oriented Middleware*** is a specific class of middleware that supports the exchange of general-purpose messages in a distributed application environment. MOM supports data exchange and request/reply style interaction by publishing messages and/or message queuing in a synchronous and asynchronous manner [26].
- ***Object Based Middleware*** offers synchronous, typed communication between components of a distributed program. Developed out of the need to extend the Object-Oriented programming paradigm to distributed systems, the middleware consists of a mechanism to allow methods to be invoked on remote objects, plus services to support the naming and location of objects in a system-wide manner (e.g. Java RMI/JINI, CORBA, Microsoft's DCOM).

The evolution from simple client-server to complex middleware is the effect of the proliferation of distributed applications is a wide range of domains. While numerous lessons can be learned from the latter developments, distributed interactive VEs design entail particular attention. The interactivity of such an environment is affected by the overall system latency. The latency is increased by the software complexity; therefore a middleware approach might not be the best solution for a distributed interactive VE.

We continue our system architecture review with a brief discussion of the controversial peer-to-peer architecture and the advantages/disadvantages brought to the distributed collaborative environments arena.

2.1.4.2 Peer-to-Peer architectures

The peer-to-peer (P2P) technology enables a combination of distributed storage to meet peak demands without saturation and the replication of frequently requested information in locations nearer larger groups of users. P2P assumes end-to-end connectivity, whether this connection is entirely unmediated or partially assisted by centralized services. A P2P distributed system containing "*n*" nodes requires $(n^2-n)/2$ connections hence it has an upper bound on the number of connections of $O(n^2)$.

Applications based on peer-to-peer (i.e. point-to-point) (P2P) architectures came into existence relatively late, even if the potential to build them on such architectures existed since ISO/OSI

[27] was defined. Some of the most successful examples are Gnutella, Napster, Pointera, FreeNet, Chord [28] used mainly for file distribution. A comparison is available in [29] and a comprehensive taxonomy of P2P applications and platforms is introduced in [30]. An analytical model for P2P file sharing system, based on random-graph theory, is proposed in [31].

From an architectural point of view Leuf [32] categorizes P2P platforms on a scale ranging from the Atomistic (AP2P) model to the User Centric (UCP2P) and Data Centric (DCP2P) models. In the AP2P each node is autonomous and fully manages its own resources and connectivity. For peer-discovery a broadcast protocol is used to send a query to join while awaiting a response. The UCP2P adds to the basic atomistic model a form of central server mediation. In its simplistic form the server component adds a directory based on unique user identifier to simplify the way nodes find each other. Clients register with the directory service to announce their availability. The DCP2P is similar with the UCP2P with the distinction that the central server maintains an index over available resources, not individual users.

Some of the advantages of P2P architectures as related to distributed interactive VEs are: *geographical locality***,** they can provide data, resources and services where they are needed; *less resource and central administration bottlenecks*, initial setup of peer workgroups by the participants is easy, project groups can form and change quickly. Moreover the *network fragmentation* attribute allows direct peer connectivity, dynamically forming virtually connected interest groups as users gravitate towards a common purpose.

The main disadvantage of the peer-to-peer model is the *complexity of the control*. Since resources and users are distributed it is difficult to collect and generate a unified view of the entire system at particular moments in time. As related to distributed interactive VEs, issues with *consistency* are inherent in systems that lack central control. A recent study of consistency issues in P2P systems can be found in [33].

In spite of the disadvantages that the peer-to-peer architecture carries, its use in distributed interactive VEs comes from the potential of direct communication between two nodes. The fact that the information does not have to pass through a central server is in favor of the interactive/near real-time behavior of the environment.

2.1.5 Real-Time Distributed Systems

In interactive applications participants make their judgments according to the situation presented to them by the human-computer interfaces. Their actions are spontaneous and random which implies that these applications should run in real-time or at least at interactive speeds. In such a system, the correctness of a computation is defined both in terms of the logical results and the time at which they are provided.

"*A Real-Time System is a system whose progression is specified in terms of timeliness requirements dictated by the environment. Real-Time is not about having a lot of bandwidth and computational power, and even if it were, we would always find ways of exhausting it.*" [10]

There is confusion in associating real-time concept with performance. That is because real-time is not necessarily about performance but more about predictability. If a system executes all its actions within the necessary (i.e. predetermined) amount of time and if its behavior is consistent, then the system is real-time.

Advances in sensors and computer networks have triggered an increase in the number of potential interactive VEs that entail large amounts of information from external devices (e.g. motion tracking systems). These sensors capture information in real time and for interactive behavior the distributed system has to disseminate the data in real-time. As we pointed out in Section 1.3 the latency inherent in the system infrastructure does not allow real-time data distribution however we can take advantage of the human perceptual limitations to develop useful distributed interactive VEs.

2.2 Virtual Environments - Mixed, Augmented and Virtual Reality

2.2.1 Background

Introduced by Milgram [2] in 1994 the Reality-Virtuality Continuum paradigm allows categorizations of systems which employ virtual reality techniques. Systems based on this

paradigm range from those that completely create and/or recreate an environment to those that augment the real environment with synthetic (virtual) components.

Definition: "Virtual Reality (VR) paradigm defines a computer simulation of a real or imaginary system (a virtual world) that enables a user to perform operations on the simulated system and shows the effects in real time" [34].

VR environments require immersive displays, which only provide a synthetic view of the world [35-38]. On the other hand, the *Mixed Reality* (*MR*) paradigm is associated with systems used to enhance the perception of the real world (*Augmented Reality, AR*) or the perception of the virtual world (*Augmented Virtuality, AV*). Visually, AR means that the real scene a person sees is augmented with computer-generated objects. These virtual objects are seen by the participant (e.g. using special displays) in such a way that the computer generated information appears in a specific location (e.g. superimposed or attached) with respect to the real objects in the scene.

On the other hand, AV means that the virtual scene a person sees is augmented with real objects. A common technique for AV is *blue-screening* or *chroma-keying* [39]. Chroma keying is used to create an overlay effect e.g. to insert a false background, such as a weather map or scenic view, behind the real object. The real object is filmed against a background having a single color or a relatively narrow range of colors, usually in the blue or green. When the phase of the chroma signal corresponds to the preprogrammed state or states associated with the background color, or range of colors, behind the real object, the signal from the alternate i.e. false, background is

inserted in the composite signal and presented at the output. In this way the virtual scene is augmented with a real object. Figure 1 describes how real and virtual worlds can be combined in different proportions.

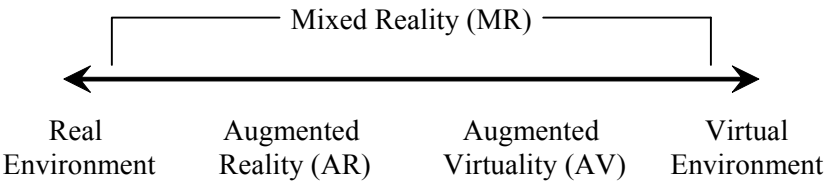


Figure 1. Virtuality Continuum

Since MR is an open interval between completely real and completely virtual, to capture the entire spectrum of applications that involve virtual components we refer to MR/VR environments as VEs. In what follows we describe the main hardware components used in the development of an interactive distributed VE.

2.2.2 Hardware Components for Interactive VEs

The first design decision in building an interactive VE is finding the way to accomplish the combination of real and virtual. We are providing a brief review on the main hardware components for the development of such environments.

2.2.2.1 3D Display Systems

One way to implement MR and particularly AR is with a see-through Head Mounted Display (HMD). Two types of HMDs that let the participant see the real world with virtual objects superimposed are: *optical see-through* and *video see-through HMDs* [40].

The video see-through HMD combines a closed-view HMD with one or two head-mounted cameras. The video cameras provide the user's view of the real world. The video from these cameras is combined with the graphic images created by a scene generator, merging the real and virtual as illustrated in Figure 2.

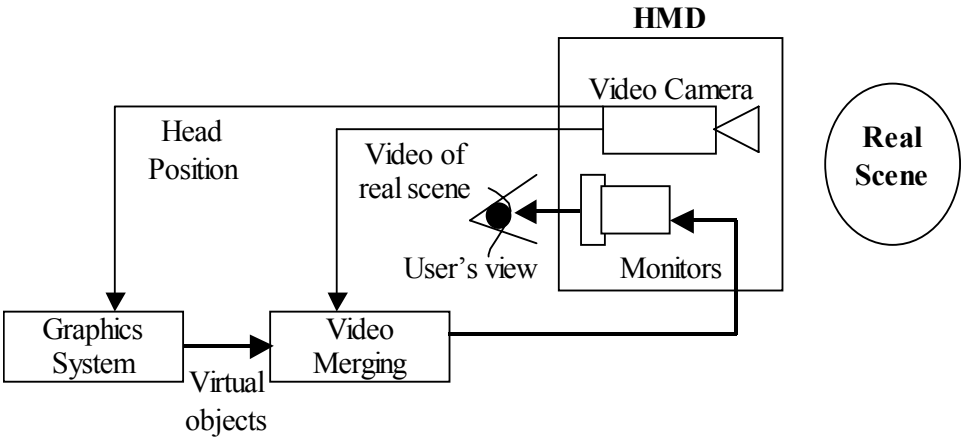


Figure 2. Video See-through HMD

Optical see-through HMDs work by placing optical combiners in front of the user's eyes. These combiners are partially transparent so that the user can look directly through them to see the real world. The combiners are also partially reflective, so the user sees virtual images bounced off the combiners from the head-mounted monitors as illustrated in Figure 3.

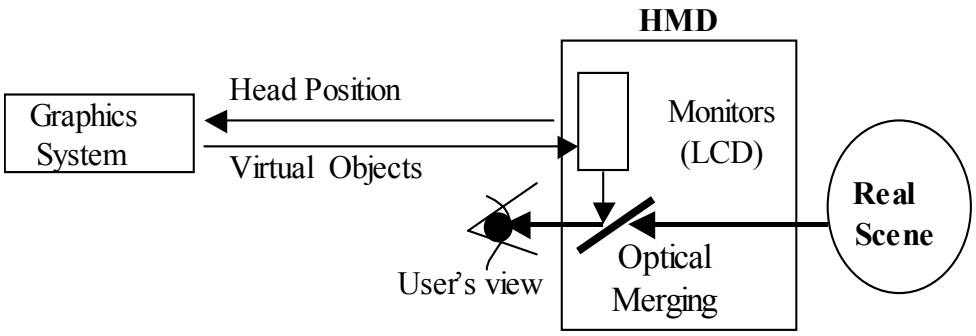


Figure 3. Optical See-through HMD

Alternative approaches to visualization are the projection-based systems. In early 90's, Carolina Cruz-Neira created what will become the most popular 3D projection based visualization system, the project-based Cave Automated Virtual Environment or the CAVE™ [37]. The CAVE™ is a room-sized, high-resolution, 3D video and audio environment. It is often implemented as a cube of approximately 12 feet on each side (Figure 4) and it uses four CRT projection systems and Crystal Eyes shutter glasses. Stereoscopic images are projected on the front and on the two sidewalls.

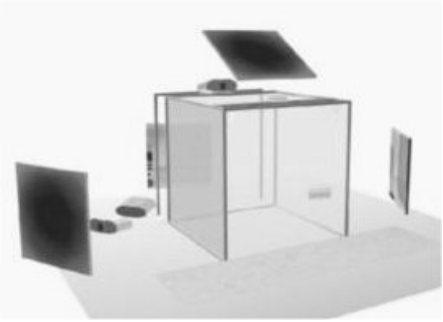


Figure 4. The CAVE

In addition to these, the CAVE™ usually uses a tracking system to dynamically adjust the computed viewpoint according to the tracked head's location in the 12^3 cubic foot volume. As a viewer moves within its display boundaries, the correct perspective and stereo projections of the environment are updated, and the image moves with and surrounds the viewer.

Two major disadvantages of the CAVE™ system are its high cost and the fact that in the case of multiple users only the user tracked sees the correct stereo perspective. Several projects have tried to diminish these disadvantages [41]. In the PlatoCAVE project, the cost of the system is decreased by using only one wall for projection. To address the second problem, a nominal fixed viewpoint is selected in the middle of the room. Hence all the users will have a slightly distorted stereo perspective but the extreme distortion perspective cases are avoided.

2.2.2.2 Sensors for Motion Tracking

Tracking systems are important components especially in an MR/AR environment. These systems are able to provide (with some accuracy) the position and/or orientation of different objects in the real world scene. Feeding their output data to algorithms, one is able to register the computer-generated objects at the correct position and orientation in the real scene. Tracking systems have been classified in a number of surveys [42-44].

Several types of tracking are available on the market today. The most widely spread commercial sensor based tracking systems are based on:

- *Magnetic tracking***.** By circulating an electric current in a coil, a magnetic field is generated. To measure the position and orientation of a receiver in space, the emitter must be composed of three coils with orthogonal magnetic fields. These systems have traditionally been prone to large amounts of error and jitter due to interference from metallic structures in the environment; however they are popular because of their robustness and the lack of constraints on user motion.
- *Optical tracking*. Optical tracking systems have been separated into two categories:
    - *Pattern recognition systems* [45] that sense an artificial pattern of lights and use this information to determine position and/or orientation.
    - *Image-based systems* [46] that determine the position by using multiple cameras to track predetermined points on the moving objects within a working volume.

- *Inertial tracking* [47]. Inertial sensors use physical phenomena to measure acceleration and rotation relative to the inertial reference frame of the earth.
- *Acoustic tracking* [48]. These can determine the position through either time-of-flight and triangulation or phase-coherence. Phase-coherence trackers determine distance by measuring the difference in phase of a reference signal and an emitted signal detected by sensors.
- *Hybrid tracking* [49-51]. These consist of combinations of the above approaches.

In most MR/AR applications, the dynamic superimposition procedures that bring the virtual objects in register with particular regions in the real scene use one of these types of tracking sometimes combined with vision techniques to increase the registration accuracy.

2.2.3 Distributed Interactive VEs Survey

A distributed interactive VE is a computer supported environment that provides an advanced form of collaboration. Collaboration can be roughly categorized on the following two dimensions: *space* and *time* both playing a determinant role in the system architecture and consistency:

- *Time-wise* collaboration can be *synchronous* when the participants involved are active at the same time or *asynchronous* when the participants involved are not active at the same time.

- *Space-wise* collaboration can be *local* if the participants are sharing the local environment or *remote* if the participants are sharing a distributed environment. The most popular ways in which synchronous computer supported collaboration is achieved are through video conferencing, chat channels and more recent, interactive virtual environments.

In reviewing the research literature attention is being devoted to distributed interactive environments developed using VR paradigms. The review of the literature is edifying in understanding the evolution of these environments in different domains as well as the evolution of the methods employed for consistency maintenance. There is a growing body of research in distributed collaborative environments which forces us to state that this is not an exhaustive review as new systems and methods are developed daily.

2.2.3.1 Distributed Interactive VEs in the Defense Industry

The most important initiator in distributed interactive environments based on the VR paradigms was the US Department of Defense (DoD). The primary objectives of these systems were training and strategy evaluation. The systems had to be scalable and exhibit a real-time behavior.

One of the first and most intensive efforts in this direction was the SIMNET project started in 1983 [52] followed a few years later by the Naval Postgraduate School's NPSNet [53]. Important contributions of these systems included the object-event architecture, which means that objects generate update events, the protocol data unit (PDU) that allowed the distribution of simulation

data among participants and dead-reckoning algorithms to reduce the number of packet updates. Later the Distributed Interactive Simulation (DIS) project improved the PDU and led to the emergence of the IEEE 1278 Standards for Distributed Interactive Simulation and its follow-on IEEE P1516 (see [54] for a discussion of these efforts). DIS also extended the dead reckoning mechanism, and the DIS standard defines nine dead reckoning algorithms.

2.2.3.2 Distributed Interactive VEs in the Entertainment Industry

The entertainment industry has been the second early player in the distributed environments through the development of networked games such as SGI Flight simulators. ID Software (www.idsoftware.com) has been an early promoter of distributed interactive games like Doom™ and the Quake™ series. Released in 1996, Quake was the first game to provide true six-degrees of freedom distributed interactive environment.

Recently, the interest has focused on multiplayer games giving birth to a new genre known as the Massively Multiplayer Online Role Playing Games (MMORPG) [55]. These games (e.g. Ultima Online, www.origin.com) allow development of social structures in which people can cooperate in large numbers and can form guilds and complex relationships. Ultimately players can share knowledge.

2.2.3.3 Distributed Interactive VEs in the Academia

The DoDs distributed environments had two major drawbacks: lack of generality, most of the simulations involved military vehicles and lack of availability, since most of the development efforts had to be kept secret.

The transition to desktop based VR started in early 90's when the academic community reinvented some of the systems used by the military, documented the systems and made the results available to the research community. A review of the most preeminent distributed collaborative environments based on VR paradigms follows.

One of the most important and longest running academic group is the NPSNET research group. Spawn from a military background, NPSNET covers all areas of research including consistency, scalability, dynamic extensibility, composability and interoperability. The latest version is the NPSNET V [56].

Early research efforts in collaborative environments were merged in the "Model, Architecture and System for Spatial Interaction in Virtual Environments" (MASSIVE) project developed at University of Nottingham [57]. Several versions have been developed in the last fifteen years. One of the goals of the last version MASSIVE-3 was to improve data consistency by ameliorating the effects of network latency using algorithms developed at the University of

Reading [58]. Each environment database is fully replicated and consistency is maintained through effective combinations of centralized updates and ownership transfers.

Another milestone, the "Distributed Interactive Virtual Environment" (DIVE) [59] developed at the Swedish Institute of Computer Science is an internet-based multi-user VR system where participants navigate in 3D space and see, meet and interact with other users and applications. The first DIVE version appeared in 1991 and concentrates on consistency and concurrency.

In the DEVA3 VR [60] each entity is composed of a single "object" that represents what the entity does, and a set of "subjects" that represents how the entity looks, sounds, and feels. The object position is updated only when the subject has moved a certain distance from its previously synchronized location.

Other academic implementation of note: SPLINE, developed at Mitsubishi Research Laboratories, introduced the idea of subdividing the world into smaller, more manageable areas known as locales [61]; ATLAS [62, 63] which introduces the ideas of personalized information filtering and self-reconfigurability, PARADISE [64], Bricknet [65]. Several research initiatives in Europe (e.g. COVEN) have underlined the importance of distributed collaboration based on VR paradigms.

The early Distributed Virtual Environments (DVEs) provided a custom solution for distributing the application state. Subsequent research efforts have been directed towards distributed scene

graphs, where the graphical scene description is used to encode the application state. Examples of such frameworks in an object oriented approach are Repo-3D [66] and AVOCADO [67].

In recent work, Schmalstieg and Hesina presented Studierstube, which uses a distributed shared scene graph to keep track of the actions applied on the shared scene by multiple participants [68]. The authors show that multiple concurrent operations on the objects in the scene may lead to inconsistent views. As communication delays increase, the inconsistency among remote participants grows even further.

Previous research has investigated the impact of network latency upon the consistency of distributed VEs. A review the research efforts for consistency maintenance as well as the most important groups of techniques available follow.

2.3 Survey of Consistency Maintenance Techniques in VEs

The main challenge encountered by distributed collaborative environments designers is the dynamic nature of the environment. The attributes of the virtual and/or real components of the scene are changed as an effect of the participants' interactions. The interactions and information exchanges generate a state referred to as the *dynamic shared state* that has to be maintained consistent on all sites.

The consistency-throughput tradeoff states that: *"It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of the state."* [69]

The diversity of the application field makes distributed applications span over a large spectrum. At one end of this spectrum are applications that attempt to guarantee shared state consistency at all participants while at the other end are applications that attempt to maximize the potential rate of the shared state updates in spite of some inconsistencies.

Table 3 summarizes how the consistency-throughput tradeoff affects the characteristics of a distributed VR system [69].

Table 3. Spectrum of Dynamic Shared State Management

System Characteristics	**Absolute Consistency**	**High Update Rate**
View Consistency	Identical at all hosts	Determined by data received at each host
Dynamic data support	Low: Limited by consistency protocol	High: Limited only by available bandwidth
Network infrastructure requirements	Low latency, high reliability, limited variability.	Heterogeneous network possible
Number of participants supported	Low	Potentially high

Shared state consistency can be decoupled based on the application characteristics in positional accuracy, behavioral accuracy and structural accuracy [70]. Positional accuracy implies that

virtual shared entities have synchronized position and orientation. Behavioral accuracy implies that the animation of the shared entities should be alike at each participant. And finally structural accuracy means that the remote rendering reflects real-time changes to the shape of non-rigid shared entities.

The next sections contain a review of the research efforts and techniques for managing the shared state. We discuss these techniques in the context of entity position however these principles apply to any other type of information maintained in a virtual environment. The techniques can be roughly grouped in three categories: centralized information repositories, dead reckoning algorithms, and frequent state regeneration.

2.3.1 Centralized Information Repositories

The approaches grouped in this category attempt to provide absolute consistency among the participants in a distributed interactive VE by ensuring that all the participating nodes contain the same values for the shared state at all times.

A common approach is to keep the state for each participant in a file. A networked file system can be employed to provide distributed access to the centralized information. Examples of such systems can be found in [71],[72]. The concurrency problem arises when two or more participants wish to change some attributes of the shared state simultaneously. The concurrency issue is solved in these cases by the networked file systems' locks. The main disadvantage of the

networked file system is the access time and the scalability of the approach. When remote participants access or modify the shared state the system performs disk input/output operations to read/write the data in the file system. These operations are very slow compared with memory access. Therefore another approach to deal with data access is to keep the information in a shared memory.

The server approach is an improvement over the networked file system in terms of data access speed; however this approach is still limited by the centralized data access and by the fact that the server is the single point of failure. If the server fails, the entire state of the virtual environment is lost. The server approach is exemplified and discussed in [73].

An important aspect of the centralized information repositories is the state update initiator. There are two major strategies, the *pull strategy* when each participant pulls state updated from the central repository and the *push strategy* when the server regularly pushes state updates to the participants. The push strategy has been employed in the Shastra system [74]. In this approach each participant node maintains a cache that preserves the shared state. When a participant updates the shared state, the server pushes the data to all the other participants. An advantage of this approach is an enhanced scalability by the reduction of the bottleneck given that the data is stored at each node.

All the above approaches consider a traditional client-server architecture. The centralization disadvantages could be eliminated using a peer-to-peer architecture. In such a system the state

maintenance is accomplished through a distributed consistency protocol. An example of such a system is the early versions of DIVE [75] . DIVE uses a distributed data management model implemented on top of the ISIS communications library [76] and offers enhanced fault tolerance since it eliminates the single point of failure. DIVE uses distributed locking and reliable multicast protocols to maintain a consistent view of all the entities in the virtual environment. A participant can directly update the position of any object by performing the following operations: obtain a distributed lock on that object in the database, update the local copy, multicast the change and release the lock. The major drawback is the significant communication overhead caused by the locking mechanism.

Enhancements of the centralized approach are presented in the BrickNet toolkit [77]. The toolkit offers different levels of data consistency ranging from reliable, in-order updates to unreliable, unordered updates. The central server ensures that data is forwarded to receivers according to their consistency requirements.

While traditional distributed VEs separate graphical and application state, several research projects like Studierstube [68], Repo-3D [66] and Avocado [67] try to simplify the development of such systems by unifying the graphical and non-graphical state into a single data structure shared over the network. In DIV [78] for example, distribution is performed implicitly through a mechanism that keeps multiple local replicas of a scene graph synchronized without exposing this process to the application programmer. The scene graph changes are propagated using reliable multicast.

2.3.2 Dead Reckoning Algorithms

Several research efforts have been directed towards dead-reckoning algorithms to address the update rate problem by maintaining a loose consistency of the shared state. The idea behind dead-reckoning is to transmit state updates less frequently and to use the information contained in the available updates to approximate the true shared state. These state prediction techniques enhance the scalability of the distributed VE at the expense of the accuracy of the shared state.

Dead-reckoning protocols for distributed VE imply two phases: *prediction* and *convergence*. In the prediction phase the current state is computed at each participant based on the previous information. In the convergence phase the inaccuracies in the predicted state are corrected and a smooth transition is assured.

In the first phase, derivative polynomials are commonly use to predict the shared state attribute values [79]. For example first order, second order or third order polynomials can be used to predict the position of a component of the shared scene based on its previous location, velocity or acceleration. Large distributed VEs like SIMNET [80], NPSNet [81] and the DIS protocol rely on position, velocity and acceleration updates to produce remote animations. An improvement over the polynomial prediction method is the hybrid polynomial prediction. The Position History-Based Dead Reckoning [70] employed in the PARADISE (Performance Architecture for Advanced Distributed Interactive Simulation Environments) system represents an example of the

hybrid approach that dynamically chooses between first-order and second-order derivative polynomials based on the information available in the preceding updates.

To obtain better prediction results several research efforts have been concentrating on matching the virtual entity characteristics with the prediction algorithm to obtain an object-specialized prediction. Examples of such approaches can be found in [82] (specialized prediction for aircrafts making military maneuvers), [83] (specialized prediction for ground based military vehicles), and [84] (a specialized protocol to remotely predict the position of drumsticks played above a sensor pad).

In the second phase convergence algorithms are employed to correct the predicted state without creating noticeable visual distortions to the participant. The simplest form of convergence is the zero-order or the snap convergence, in which the prediction is corrected immediately without any regard to the visual distortion seen by the participant. The zero-order convergence has undesirable visual effects. For example the entities in the shared scene seem to change position abruptly defying the laws of physics. An improvement over the zero-order convergence is the linear convergence. The linear approach results in a slower convergence on the convergence path providing the participant with continuity along the visual path.

The linear approach does not take into account the entity acceleration. For improved transition smoothness, the "quadratic" method can be employed to account for acceleration a, and velocity v. The entity's location at a particular time t can be approximated using the well known quadratic function:

$$x(t) = x_0 + v(t)\Delta t + \frac{a(t)\Delta t^2}{2}$$ Equation 1

Instantaneous acceleration can be computed as:

$$a(t) = \frac{dv}{dt} = \frac{d^2x}{dt^2}$$ Equation 2

The quadratic method also fails because even though an entity's motion is represented more realistically, its final velocity is likely to be incorrect. The quadratic function employs instantaneous acceleration and does not consider the variation in acceleration or "jerk" defined as:

$$j(t) = \frac{da}{dt} = \frac{d^3x}{dt^3}$$ Equation 3

In spite of the additional computational complexity, the "cubic spline" approach offers one of the most realistic methods for convergence by accounting for the entity's starting and ending position and velocity. As a result, the entities that follow a cubic spline path have no jitter, unless network lag is especially severe. Using a cubic spline to create a path is a matter of simple algebraic equations. The input for these equations are four (x,y) coordinates. The first coordinate represents the object's starting position (x_0,y_0). Similarly, the fourth coordinate (x_3,y_3) signifies the object's ending position. Usually the end position is a new (x,y) coordinate that has just

arrived in a data packet. The most important coordinates are the second and third; they represent the object's velocity. For the second coordinate (x_1,y_1), we calculate where the object will appear after 1 second with its current velocity. For the third coordinate (x_2,y_2), we reverse the object's end velocity, and calculate where it would appear after 1 second. Below is the set of parametric equations defining the spline:

$$x = A_x t^3 + B_x t^2 + C_x t + D_x$$
$$y = A_y t^3 + B_y t^2 + C_y t + D_y$$

Equation 4

where

$$A_x = x_3 - 3x_2 + 3x_1 - x_0$$
$$B_x = 3x_2 - 6x_1 + 3x_0$$
$$C_x = 3x_1 - 3x_0$$
$$D_x = x_0$$
and
$$A_y = y_3 - 3y_2 + 3y_1 - y_0$$
$$B_y = 3y_2 - 6y_1 + 3y_0$$
$$C_y = 3y_1 - 3y_0$$
$$D_y = y_0$$

Equation 5

With the increase of the distributed system's nodes processing power we see an increase potential for dead reckoning algorithms as an improvement to network traffic. However dead reckoning introduces several limitations. First of all it does not guarantee that all participants share identical state about each entity; second, simulations that rely on dead reckoning protocols are more complex to develop, maintain, and evaluate. Furthermore, dead reckoning algorithms must be customized based on the object behavior to obtain precise remote modeling.

2.3.3 Frequent State Regeneration

For some distributed VEs absolute consistency is not required. The consistency protocol can be replaced in these cases with a frequent state update system based on the fact that if slight inconsistencies in the shared state appear they will be temporary and limited. One example of such an application is remote video streaming. Video streaming could be compared with a frequent state regeneration system where each frame is a new update. Loosing some frames in a 30 FPS (frames per second) video stream would be almost imperceptible for a human [85].

2.3.4 Resources Management Strategies

Resource utilization is directly related to consistency maintenance in interactive VEs. The amount of data and computation that has to be distributed in such an environment will affect the interactivity of the application and the environment consistency. Resource management strategies can be grouped in four categories as described in the following section. An in depth discussion of these strategies can be found in [86].

2.3.4.1 Communication Protocol Optimization

Communication protocol optimization can be achieved through packet compression and packet aggregation. Through packet compression, the size of the packets transmitted in the virtual

environment is reduced. Compression might be either *lossless*, i.e. the information is compressed/decompressed without loss (e.g. run-length encoding, RLE algorithm), or *lossy* i.e. some of the data is lost in the process of compression. Another division in compression techniques is *external* vs. *internal compression*. While external compression manipulates the packet data considering the content of the previous packets, internal compression manipulates a packet based only on its internal content.

Another general strategy for optimization is *packet aggregation* which seeks to reduce the number of packets that are actually transmitted by combining information from several packets into one packet, assuring bandwidth savings. On the other hand packet aggregation introduces tradeoffs between the bandwidth reduction and the interactivity of the system since the first packet in the aggregation queue has to wait for the last packet in the queue before the aggregation process can proceed.

2.3.4.2 Visibility of Data Management

The optimization techniques in this category reduce the bandwidth consumption in the distributed system by reducing the number of message received. Data flow optimization techniques like area-of-interest (AOI) filtering [57] and multicasting (e.g. PARADISE [87], Diamond Park [61]) allow packets to travel only to parts of the network that contain interested participants.

These techniques impose a tradeoff between data partitioning and multicast grouping. A significant body of research has been developed to create a balance between a fine-grain data partitioning and multicast grouping, and hybrid multicast aggregation techniques have been proposed [88]. Multicasting protocols can be designed at the application level, allowing the development of *overlay networks* on top of the current network protocol stack.

2.3.4.3 Human Perception Limitation

Humans have limited perceptual abilities in space and time. Such limitations can be exploited in a tradeoff between *ideal consistency* and *perceived consistency*. In Chapter 3 we will discuss in more depth a time-space consistency model that takes into account human perceptual limitations.

Information about the virtual entities in the environment can be provided at different levels of detail (LOD) and different rates (i.e. exploiting visual perception limitation). The timeliness characteristic of information received by participants can be improved exploiting the temporal perception.

Particularly interesting is the *temporal contour* technique [89] that creates a temporal map in which every point P in the virtual space has a t value assigned corresponding to the network latency experienced by the local participant L for updates originating at P. The t value at any point P can be computed as

$$t(P) = \Phi_L \times \sum_{i=1}^{n} (d_i e^{\frac{-r_i^2}{2v_i}}) \qquad \text{Equation 6}$$

There are n remote participants. Each remote participant i has a locally perceived network delay d_i and is located at position P_i relatively to user L. Φ_L is a constant that assures that $t(P)$ is null at the local participant position L. r_i is the distance between point P and point P_i, and v_i is a variance value that affects the flatness of the contour around P_i. The contour equation calculates a composite of the participants' network latencies by weighting each one by the corresponding entity's distance from the active participant. The main limitation of this technique is the considerable computation requirements particularly in the presence of many active participants.

2.3.4.4 Systems Architecture

An important decision in designing a distributed interactive environment is the system architecture. While the system architecture is driven by the application requirements and the current evolution of systems architectures (see Section 2.1.4), two major trends exist. The first trend investigates clustering and partitioning techniques evolving mainly around the client-server paradigm, while the second trend investigates combinations between the atomistic peer-to-peer and the client server paradigms.

3. SHARED STATE MAINTENANCE

A number of problems arise when designing algorithms for distributed interactive VEs. Current applications tend to employ and are more dependent on sensory information. As discussed in Section 2.2.2, a VE might include a variety of position sensors. Sensor data is especially required to render the position and orientation of the virtual components of the scene in applications where a soft blending of the real and virtual is desired. To obtain an enhanced view of the real environment, participants could wear HMDs to observe three-dimensional computer-generated objects superimposed on their real-world view or vice versa. The position and orientation of each participant's head must be obtained to render the computer-generated objects from the correct viewpoint, at the correct depth [90]. Because virtual and real objects must be placed into register, i.e., spatial coincidence, we require accurate motion tracking not only for the head of the participant but also for other objects.

From an application perspective, the need for real-time sensors that update the environment at high rates is associated with several problems. The distributed system nodes must collect and distribute each sensor's data in real-time and at the same time maintain a *consistent view* of the environment. While interaction with the virtual components of the shared scene is still discrete, new levels of interaction arise through the use of sensor-based interfaces.

From a system perspective the *dynamic membership properties* of a VE have an impact on the system architecture. New data distribution schemes are necessary to cope with the interactivity and the dynamic properties of these environments.

In the next sections we explore two viewpoints of distributed interactive VEs. The first viewpoint is an *application perspective* in which we present a formulation of the consistency problem and propose a new categorization criterion for applications, as well as a dynamic shared state maintenance algorithm. The second viewpoint is a *system perspective* in which, modeling the distributed interactive system as a graph, we propose an approach to derive the minimum delay communication sub-graph. We then present and analyze a novel data distribution architecture targeted towards sensor-based interactive VEs.

3.1 Distributed Interactive Virtual Environments - Application Perspective

3.1.1 Consistency Model

There is no general solution for dynamic shared state maintenance in interactive VEs. Currently, the combination of techniques employed is dictated by the distributed application characteristics. A formal categorization [91] divides distributed interactive applications based on two characteristics/criteria: the way an entity in the application changes its state, *continuous* vs. *discrete* and the way the participants apply actions, *turn-based* vs. *concurrent*. The cross product

of these characteristics leads to four categories: *continuous concurrent* applications, *continuous turn-based* applications, *discrete concurrent* applications and *discrete turn-based* applications. From this perspective an interactive VE application can be seen most of the times as a *continuous concurrent* application.

The dynamic shared state inconsistency problem can be translated to a *Time-Space* inconsistency problem. For example, because of message transmission delays, different participants may see the same moving virtual object located at different positions at the same (i.e. reference or wall-clock) time *t*. In what follows we will expand on the definitions and theorems from [92] to provide a theoretical framework for reasoning about consistency as it pertains to distributed interactive VEs.

Let's consider a distributed system consisting of *n* nodes for a defined time interval (we assume *static membership*, i.e. the number of participants in the environment does not change; the *dynamic membership* property will be discussed later). We assume that each node is associated with one and only one participant. The participants at each node can apply consecutive actions on the virtual objects in the shared scene.

A collaborative VE is populated by one or more virtual objects (e.g., a tree is one object, an avatar might be another). Each object has a set of attributes (e.g., position, color, velocity), and each attribute has a set of possible values. The state of an object at a particular time is defined by the values of the object's attributes at that time. Each object can be categorized as *static* or

dynamic. While the attribute values of a static object are fixed, the attribute values of a dynamic object change over time.

Definition: State of an object

Given an object θ, the state of θ is defined by its attributes (e.g., position, orientation, color) at a particular moment in the reference time t. Let $S_{\theta}^{i}(t)$ be the state of the object θ at node i at time t, where $1 \leq i \leq n$.

One of the reasons for the existence of dynamic objects is the participants' interaction. The participants interact on the shared scene by applying a set of actions on the objects. The effect of these actions propagates in the entire system and affects other participants' judgment and other objects' behavior.

Definition: Action times

Let a denote an action. The action applied by the participant has defined attributes (e.g., if we consider orientation or position change, the action may have a particular direction, velocity and acceleration). Let $u(a)$ be the node where the action is generated, $\tau_i(a)$ the time at which the action is generated at site i (i.e. $i=u(a)$), and $o(a)$ the shared object on which the action is applied. Moreover we denote by $arrive_i(a)$ and $execute_i(a)$ the arrival time and execution time, respectively, of action a at node i.

For example, to express the fact that the action is generated at the same site we can use the following notation: $\tau_i(a) \rightarrow u(a)=i$. To express the fact that actions having an arrival time were generated at a remote site we can use: $arrive_i(a) \rightarrow u(a) \neq i$.

Further we define the causal precedence relation, a powerful concept for reasoning, analyzing, and describing inferences about a distributed computation.

Definition: Happened-Before

Given two actions a_i and a_j we define the causal order of precedence (or happened-before relation) as a_i happened-before a_j denoted as $a_i \rightarrow a_j$ if and only if one of the following occurs:

- $u(a_i)=u(a_j)=k$ and $\tau_k(a_i) < \tau_k(a_j)$; in other words, both actions were instantiated at the same node k and a_i was instantiated first.
- $u(a_i) \neq u(a_j)$ and $arrive_k(a_i) < \tau_k(a_j)$ where $k=u(a_j)$; in other words the actions were instantiated at different nodes, however the action a_i arrives at node k before action a_j is instantiated at this node.
- there exists an action a_m such that $a_i \rightarrow a_m$ and $a_m \rightarrow a_j$

From the definitions above we can describe the absolute consistency for the distributed VE.

Definition: Absolute Consistency

Given an object θ, the object is absolute consistent at any time t, if and only if $(\forall)\ 1 \leq i,j \leq n,\ i \neq j$, $s^i_\theta(t) = s^j_\theta(t)$. We say that the system is absolute consistent if and only if $(\forall)\ \theta$ and $t>0$, θ is

absolute consistent. In other words, a distributed VE is absolute consistent if and only if all the shared objects are consistent at all times.

Definition: Average Delay

Let t_{ij} represent the average delay from node i to node j. Since the nodes are interconnected on an arbitrary network we make here the assumption that the messages experience similar delays on the path from i to j and on the path from j to i.

Given the above definition, several significant theorems follow. These theorems will help us reason and categorize distributed interactive VEs based on the action frequencies applied on the objects in the scene.

Theorem 3.1. A distributed system, starting from an initial consistent state, is absolute consistent if and only if any action generated by a participant is executed simultaneously at all nodes.

Proof. ($\rightarrow$) We shall prove that if any action is executed at the same time at all nodes, then the system is consistent. The initial state is consistent and each operation is executed at all nodes at the same time, so each participant sees the updates at the same time. Therefore, the system is absolute consistent.

($\leftarrow$) We prove by contradiction that if the system is absolute consistent, then any action must be executed at the same time at all sites. We assume there exists an action a applied on an object θ,

where $u(a)=i$ and $t=\tau_i(a)$, which is not executed simultaneously at all sites. If node i executes action a at time t_1 and node j executes the same action at time t_2, $t_1>t_2$, then $s^i_\theta(t_1) \neq s^j_\theta(t_1)$ or $s^i_\theta(t_2) \neq s^j_\theta(t_2)$, that contradicts the assumption that the system is consistent. *Q.E.D.*

Theorem 3.2. Given an action a issued on object θ, where $u(a)=i$, if action a is executed at site i immediately after it is issued, then the system is not absolute consistent at time $\tau_i(a)$.

Proof. The theorem states that indeed the inherent delays in a distributed system are non null. To prove it, suppose $t = \tau_i(a)$ and $(\exists)\ \delta>0$ such that the system is absolute consistent at time $t-\delta$. If at time t one node j updates the state of an object θ with action $a' \neq a$, then $(\forall)\ 1 \leq i,j \leq n,\ i \neq j$, $s^i_\theta(t) \neq s^j_\theta(t)$ hence the system is not absolute consistent at time t.

Secondly we prove that at time t all other nodes ($i \neq j$) do not change the state of the object θ so they become inconsistent. Since action a will update the state of object θ at time t, we have $s^i_\theta(t) \neq s^i_\theta(t-\delta)$. Because the message takes time t_{ij} to propagate between two nodes, the state at another node j remains unchanged at time t, i.e. $(\forall) 1 \leq i,j \leq n,\ i \neq j,\ s^j_\theta(t) = s^j_\theta(t-\delta) = s^i_\theta(t-\delta)$, and from here, $s^j_\theta(t) \neq s^i_\theta(t)$. Hence the system is not absolute consistent at time t. *Q.E.D.*

Theorem 3.3. Given an action a issued on object θ, to keep the system absolute consistent

$$execute_i(a) \geq \tau_i(a) + MAX^{n}_{\substack{j=1 \\ j \neq i}}(t_{ij}) \qquad \text{Equation 7}$$

The proof of this theorem follows from the previous one.

Continuing our reasoning about distributed interactive VEs, it is clear that a method for maintaining a consistent shared state would be to delay the application of the action on the local scene until all nodes have been notified. Such a synchronization barrier will have negative impact on the responsiveness of the application, since the local participant will perceive his local actions as delayed. Clearly, there is a tradeoff between responsiveness and consistency, and thus between interactivity as perceived by the participants and consistency. The key to this problem lies in the delays among the nodes of the system, t_{ij}. These delays are inherent in a distributed system infrastructure.

Importantly this work proves that, by combining a distributed monitoring system with an interactive distributed VE, the consistency can be significantly improved. The Adaptive Synchronization Algorithm (ASA) described in Section 3.1.4 represents our proposed, and we believe, elegant solution. Prior to presenting this contribution we discuss the time-space inconsistency model and the associated human factors issues.

3.1.2 Human Factors - Response Times & Sensors

The *time-space inconsistency* [91] for distributed interactive applications can be defined as

$$\Omega = \begin{cases} 0, if \mid \Delta \mid < \varepsilon \\ \int_{t_0}^{t_0+\tau} \mid \Delta(t) \mid dt, if \mid \Delta \mid \geq \varepsilon \end{cases} \qquad \text{Equation 8}$$

where $\Delta(t)$ represents the difference between a shared object's local attribute (e.g., position/ orientation) and the value of this attribute as estimated on a remote node. ε is the minimum difference between attribute values (e.g. a position drift) that the human player can distinguish in the application. In the above expression, Δ is a variable over the reference time t, while t_0 is the instance at which the difference starts, and τ is the duration for which the difference persists.

If $\Delta(t)$ is constant over time, the time-space inconsistency, Equation 8, can be simplified to

$$\Omega = \begin{cases} 0, if \mid \Delta \mid < \varepsilon \\ \mid \Delta \mid \tau, if \mid \Delta \mid \geq \varepsilon \end{cases} \qquad \text{Equation 9}$$

In a distributed VE the $\Delta(t)$ value can be influenced by the rendering characteristics of the nodes, as well as by the nature of the action's attributes. For example, if we take into consideration the time derivatives (e.g. acceleration) as an action's attribute, then $\Delta(t)$ will not be constant;

however it will be predictable. Given the preceding context, the ε variable has particular relevance to our discussion. ε is dependent on human-computer interaction factors and human perception. If $|\Delta(t)| \leq \varepsilon$, the difference in the attributes cannot be perceived by the participants, so it does not influence their judgments. The values of ε are dependent on the object's attributes (e.g. speed, size) as defined by the application domain. As an example, in a distributed aircraft battle application ε might be several tens of meters while in a distributed running competition ε might be several centimeters.

It is conceivable that there are other factors that influence the time-space inconsistency. We hypothesize that time-space consistency is also related to additional human-computer interaction factors. For example, the fastest human-computer response time includes perceptual (i.e., user perceives the items on the display or auditory signals), cognitive (i.e., user retrieves information from his own memory) and motor cycle times, which can add up to an average of 240ms [4]. In the next sections we extend this model and provide a novel categorization of distributed interactive VE applications based on the interaction frequency.

3.1.3 Classifying Interactive VE Applications - Action Frequency Patterns

Let's consider the following application scenario. A surgeon located in an office building is analyzing a 3D model of the mandible of a patient. This physician would like to discuss the surgical procedure that will follow shortly with a colleague, whose office is in another building.

As part of their discussion, they have to analyze the 3D model of the patient's mandible. They use the 3D distributed visualization platform implemented on the hospital's local area network. For stereoscopic visualization, each office is equipped with HMD [93] and a sensing glove [94]. In this scenario, the distributed visualization platform allows one participant to modify the position and orientation of the 3D model from a mouse-driven graphical user interface (GUI, Figure 5) or through the sensing glove shown in Figure 6.

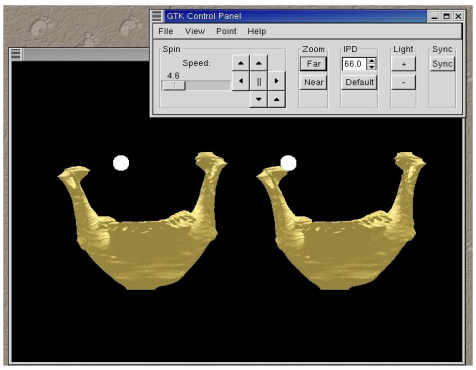


Figure 5. GUI based interaction

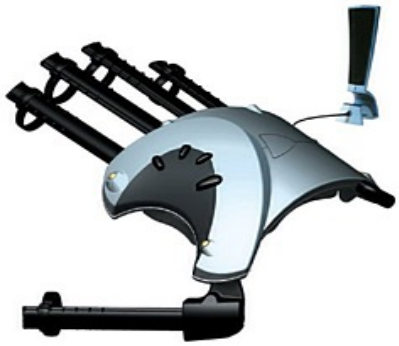


Figure 6. Sensor based interaction

There are two problems that arise in this scenario. The first is related to the network latency. As one of the participants manipulates the 3D model, the network latency desynchronizes their common viewpoints. Moreover, since network jitter is also present, the position/orientation drift among the views increases in time, while the participants are not aware of the inconsistency of their viewpoints. The second problem pertains to the nature of the interaction with the objects in the shared scene. The 3D model can be manipulated either from the GUI through discrete and predictable actions, or using the glove-like peripheral device, which updates the environment at a

higher rate and allows relatively unpredictable actions. The participant acting on the GUI through the mouse, for example, cannot exceed a certain frequency of actions mainly because of his motor reaction time. At the same time, since the position and the orientation of the object are set through the interface, predictable actions are applied on the object (e.g. by pressing the GUI's "Rotate around OX axis" button). In contrast to the GUI, the glove-like peripheral device is usually tracked at high frequencies (e.g. a P5™ glove [95] has an optical tracking system attached that has a refresh rate of 60Hz) which is going to capture the participant actions at a higher frequency than the one attainable through the GUI. As a result, we have two types of interaction with the 3D model that have distinct patterns. While the network latency problem is well known in distributed interactive simulations, the second problem is more subtle and requires further analysis. Based on the above observations, we propose a novel criterion for categorization of distributed interactive VE applications.

Two cases can be established; the actions frequency is either lower or higher than the inverse of the network delay between two interacting nodes. To study these issues, we define two frequencies in the following paragraph: the *upshot frequency* and the *action frequency*. We then show how these frequencies can be used to categorize distributed interactive VE applications as *high* or *low* frequency applications.

Let t_{xy} be the average network delay between two participating nodes X and Y, as defined in the previous sections. We assume, for now, that there is no jitter. The jitter compensation is described later. Let's consider an instance of a distributed VE that allows interaction with m

virtual 3D objects. The shared scene produced must be displayed at all the participating nodes. Let n be the number of participating nodes. We assume that all virtual objects are rigid and we restrict the actions applied on them to rotations and translations. The discussion can be further extended to arbitrary affine transforms and non-rigid, deformable 2D/3D objects.

The participants interact with the virtual objects in the scene. Each participant's interaction can be seen as a sequence of actions applied on the objects in the scene. An action is identified by a name, a direction described by a vector, and a velocity. The model can be extended to higher order space-time derivatives. Since real-life interactions are spontaneous, the action duration is not known when the action is applied. As defined in the previous sections we know only $\tau_i(a)$, $arrive_i(a)$ and $execute_i(a)$. The action duration is known only after the next action is initiated, the actions being generated in a consecutive sequence. Considering two consecutive actions a_1 and a_2 the duration of a_1 can be computed as $\tau_i(a_2) - \tau_i(a_1)$.

Definition: Action Frequency

We define the *action frequency*, $\nu_k(\theta)$ of a node k, as the number of actions performed by node k on one object θ in the shared scene per unit time (i.e. 1 second).

The action frequency is measured in actions per second and can be estimated in the following way. Let $b_{\theta k}$ be the number of actions applied on the object θ in the scene by participant k during Δt. Participant k may interact with any object in the shared scene. The total number of actions applied by k on all m objects in the scene during Δt will be given by

$$\sum_{\theta=1}^{m} b_{\theta k} \quad .$$

Definition: Average Action Frequency

Let v_k be the average action frequency obtained by computing the average number of actions applied by k on an object in the scene, given by

$$v_k = \frac{\sum_{\theta=1}^{m} b_{\theta k}}{m \cdot \Delta t} \qquad \text{Equation 10}$$

v_k can be used as an estimate of the action frequency for participant k on an object in the shared scene, i.e. $v_k(\theta)$

$$v_k(\theta) \cong v_k \qquad \text{Equation 11}$$

The longer the Δt, the more accurate is the action frequency estimation. Moreover this estimation can be done for each participant in the distributed VE application.

As an observation, in the estimation of the average action frequency, only objects on which the participants interact are considered. Another observation is that the interaction pattern on the

objects in the environments is fairly uniform (i.e., the participant interaction frequency on a particular object is not very high as compared to his/her interaction on the rest of the objects). If this is not the case, the maximum interaction frequency on a particular object might be used for determining the application's characteristics.

Definition: Upshot Frequency

Furthermore, let ν_0 be the *upshot frequency* between two nodes *X* and *Y* defined as

$$\nu_0 = \frac{1\ action}{t_{xy}} \qquad \text{Equation 12}$$

with actions per second as measurement unit.

The upshot frequency is dependent on the network delay between two participating nodes and limits the interactivity potential between two nodes. For example, if t_{xy} is 100 milliseconds, the upshot frequency between the nodes will be 1/0.1 or 10 actions per second. Computing the average network delay between each pair of participants, the corresponding upshot frequency (ν_0) can be computed. Based on the above definitions, two cases can be distinguished: the action frequency is less than the upshot frequency, $(\nu_k < \nu_0)$, or the action frequency is greater than or equal to the upshot frequency, $(\nu_k \geq \nu_0)$.

To illustrate the discussion above, consider a simple case in which there is only one virtual 3D object in the shared scene and two participants, node *X* and node *Y*, located on a network. Node *X* can change the object orientation by applying arbitrary rotations around the object coordinate axes. Since it is a distributed application, the shared scene must be maintained consistent (i.e. both participants should see the same orientation for the object). The first case $(\nu_k < \nu_0)$ usually corresponds to a distributed interactive VE application involving either low update frequency devices (i.e. when compared with the upshot frequency) or participants who perform actions on the objects in the shared scene through a GUI. The fastest human-computer response time adds up to an average of about 240ms [4]. Under the later assumptions, distributed VE applications that fall in this category should not be deployed on a network that has an *upshot frequency* lower than 4.16 (actions/second), in other words the network delay has to be lower than 240ms.

In most of the cases when the distributed VE application is deployed on a local area network, the frequency of the actions applied by the participant through a GUI on the objects in the shared scene will be below the upshot frequency. Moreover, some actions will generate continuous movements that can be predicted. For example, the participant might spin an object for an indefinite time period with a specific velocity around a specific axis. In this case, ν_0/ν_k tends to infinity and once the nodes are synchronized no additional network traffic is necessary until another action is applied. The drift can be accurately computed if we know the network delay t_{xy} and the action (e.g. rotation) attributes (e.g. direction, velocity).

We emphasize this scenario with an example. A timing diagram is shown in Figure 7 that contains two actions applied by node *X* on an object in the shared scene and the propagation of these actions to another participant, node *Y*. The first action, a_1, takes 14 time units, the second action, a_2, takes 8 time units, and the network delay between nodes *X* and *Y* is 1 time unit. The synchronization algorithm proposed in Section 3.1.4 accounts for the network latency. In Figure 7, the shaded areas represent the time intervals when *X* and *Y* are synchronized.

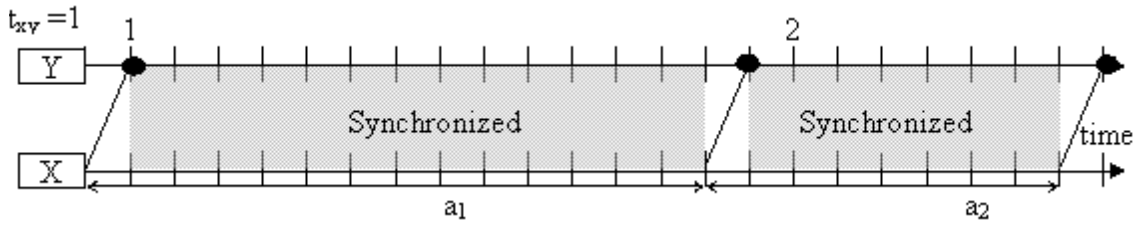


Figure 7. Action frequency is less than the Upshot frequency ($\nu < \nu_0$), delay $t_{xy}=1$

The second case ($\nu_k \geq \nu_0$) corresponds to a distributed VE application containing a fast updating device like a tracking system or a high latency network connection. In this case, a sequence of actions might take place at node *X* before node *Y* is notified about the first action in this sequence. This scenario can be described with a simple example. Below is a timing diagram that contains 10 actions and their respective durations: actions a_1, a_2, a_3, a_5, a_6, a_7, take 1 time unit; action a_4 takes 2 time units, and actions a_8, a_9, a_{10} take 3 time units. The network delay between nodes *X* and *Y* is 3 time units.

In this case, high quality of synchronization cannot be reached since node *Y* will continuously try to "*catch up*" with node *X*. By the time node *Y* has compensated for the drift, node *X* has applied new actions on the object. Node *Y* is not aware of those actions at that time.

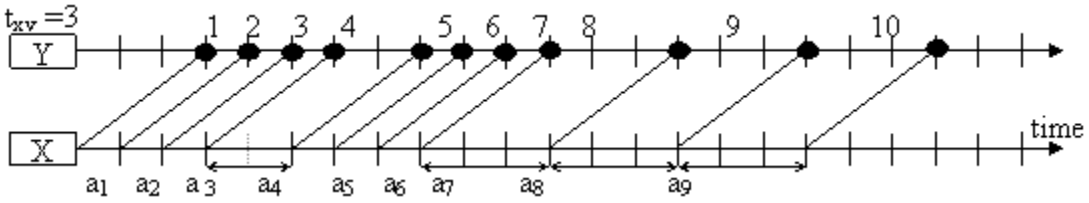


Figure 8. Action frequency is greater or equal than the Upshot frequency ($\nu \geq \nu_0$), delay $t_{xy}=3$

3.1.4 The Adaptive Synchronization Algorithm

The Adaptive Synchronization Algorithm proposed is targeted towards distributed interactive VE applications which fall in the first category ($\nu < \nu_0$) as defined in the previous section. In other words, the algorithm assumes an event-based mechanism, triggered either by the participant actions on the shared scene or by a sensor (e.g. a motion tracking system) whose update cycle time is comparable or higher than the network latency. Such assumption is generally true as high-speed networks and optical routing are becoming increasingly available.

To control the position and orientation of the objects in the shared scene, each 3D virtual object has a control packet (CPO) associated with it. The CPO contains information about the position and orientation, as well as information regarding the actions associated with each object:

rotation, translation or scaling. The small size of the CPO ensures low transmission delays. As the CPOs flow through the network, the ASA uses their information to keep the shared scene among participants consistent. In other words, the information carried by the CPOs is distributed to each participating node allowing them to compensate for the network latency.

Definition: Drift Value

Let's define the *drift value* for a particular object θ in the shared scene and a remote observer (i.e. the node that does not apply an action on the object), node k as the product between the action's velocity applied on the object by the node $u(a)$ and the network delay $t_{u(a)k}$, i.e. the delay between the node where the action was initiated $u(a)$ and the node k.

The fact that the action is generated at a remote site can be expressed as

$$arrive_k(a) \rightarrow u(a) \neq k$$

Also actions are executed as they are received at both nodes

$$arrive_k(a) = execute_k(a)$$

and

$$execute_i(a) = \tau_i(a)$$

If we denote M_t as the number of virtual objects in a shared scene of N_t participating nodes, all at the same time t, a drift matrix $D(M_t,N_t)$ can be associated with the distributed VE system at a particular time t and it can be computed as

$$D(M_t, N_t) = S \cdot T^{transpose} \qquad \text{Equation 13}$$

where S and T are both column vectors, S containing the action velocities for each object currently in the shared scene, and T the network delays vector from each action/update producer node i to each participating node j. $T^{transpose}$ represents the transpose of T. The action velocity is extracted from each object's CPO, while the network delay is measured by each observer node (i.e. the node that does not apply an action on that object) using an adaptive probe. The S matrix is stored locally at each node and updated when the scene changes.

A decentralized computational approach partitions the drift matrix into N column vectors, denoted here *drift vectors*, which contain the drift values of all the virtual objects in the shared scene for each node. The drift vectors are updated when a new 3D object is inserted or removed from the shared scene by adding or removing the entry associated with it from all nodes. The drift vectors are also updated when the participants perform actions on the objects. Whenever an action is applied to an object (e.g. a rotation), a CPO is broadcasted to all the nodes.

The information (e.g. action velocity) from the CPO is the first component used for synchronization. The second component accounts for the network latency. At specific intervals, each node "pings" the other nodes to estimate the average network delay (i.e. t_{ij}) and computes the drift vectors associated with the objects in the scene as the product between the propagation delay and the objects' actions velocities. Each delay measurement between nodes triggers a local node's *drift vector* update.

A pseudo-code sketch of the ASA is described in what follows. The *ComputeNodeDelay()* function returns the delay between two nodes t_{ij}. The *UpdateDrift()* function updates the drift values for the objects in the scene on each node. Three Boolean variables are used: *changedScene* that accounts for the changes in the scene, *newClientReques*t which is set if a new participant has joined, and *trigger*, used in accounting for the network behavior as described in the next section. Finally, the functions *ReceiveChanges()* and *BroadcastChanges()* ensure correct shared state updates among the nodes of the system. A consistent dynamic shared state is maintained over all the participants.

Observer (Consumer) node:

Initialization:

$T_n \leftarrow ComputeNodeDelay()$
$S_n \leftarrow UpdateAction();$
$D_n \leftarrow UpdateDrift()$
UpdateLocalScene();

Main:

if (trigger)
$T_n \leftarrow ComputeNodeDelay()$
$D_n \leftarrow UpdateDrift()$
end if
if (changedScene)
$S_n \leftarrow ReceiveChanges()$
$D_n \leftarrow UpdateDrift()$
end if

Action Producer node:

```
for ever listen
        if (newParticipantRequest)
                SendToNewParticipant(Sn);
        end if
        if (changedScene)
                BroadcastChanges();
        end if
end for
```

3.1.4.1 Fixed Threshold vs. Adaptive Threshold

As the traffic in the network changes, the round trip times between different nodes vary. To improve the shared state maintenance, delay measurements must be triggered at different time intervals. The delay measurements must follow the network jitter behavior yet this distributed monitoring scheme must be maintained at low levels of intrusiveness. The goal is to obtain an accurate estimate of the average delay between each pair of participants.

The algorithm uses two approaches to *trigger* the information collection. In the first approach, the measurements are triggered by each node at regular time intervals. We denoted this approach the *fixed threshold* one. Triggering these measurements too often increases the intrusiveness of the software monitoring scheme. Moreover, if the network jitter is very low, measurements are redundant. An alternative approach consists of *adaptively* triggering the delay measurements

based on the delay history, which better characterizes the network traffic and the interactive application behavior, as illustrated in Figure 9.

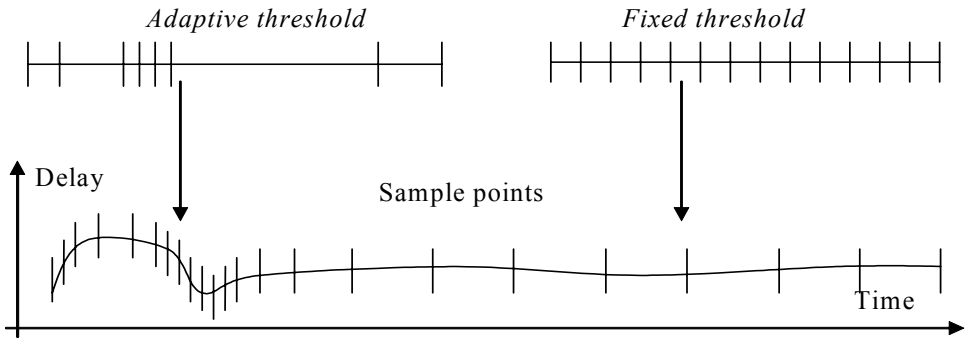


Figure 9. Adaptive vs. Fixed threshold

In the adaptive approach, a fixed threshold is initially used at each node to build the *delay history* denoted here H_p. The delay history is a sequence of p delay measurements h_i where $i=1,p$. Furthermore let σ be the standard deviation of H_p and h_{mean} the mean of H_p, in other words

$$h_{mean} = \frac{\sum_{i=1}^{p} h_i}{p}, \quad \sigma = \sqrt{\frac{\sum_{i=1}^{p} (h_i - h_{mean})^2}{p}} \qquad \text{Equation 14}$$

Let h_0 be the most recent delay, i.e. the last number in the H_p sequence, and γ_0 the current frequency of delay measurements, expressed as the number of measurements per second. The adaptive strategy is to decrease γ_0 with 1 unit if $h_0 \in [\, h_{mean} - \sigma\,,\, h_{mean} + \sigma\,]$ and to increase γ_0 with 1 if h_0 does not belong to this interval. Of course the values of all these variables are application dependent and can be tuned to best fit the application.

3.2 Distributed Interactive VEs - System Perspective

An important characteristic of collaborative VEs is the membership property. Most collaborative VEs have a *dynamic membership* property, i.e., the number of participants can change at any time (e.g., participants can join or leave the environment at any time). Other VEs have a *static membership* property, i.e., once the distributed session is initialized the number of participants does not change (e.g. a collaborative environment for business meetings).

3.2.1 A Distributed System Model

An interactive distributed VE can be deployed over a LAN, a private network or over a WAN infrastructure (e.g. the Internet). To provide an abstraction for the underlying infrastructure, as well as a reasoning anchor for the system, we use graph theory notation.

The entire network can be represented as a directed weighted graph $G=(V,E)$, where V denotes the set of nodes in the network and $|V| = n$. Some of these nodes can only serve as message routers while others are direct interfaces for the participants. Let V' be the set of nodes that can participate in a collaborative session, $|V'| = m$, and obviously $V' \subseteq V$ and $m \leq n$.

Because of the dynamic membership property at a particular moment in time t, the set of nodes participating $P(t)$ will be equal to or smaller than the set of nodes that have the potential to

participate, i.e. $P(t) \subseteq V'$. Hence the distributed interactive VE will communicate for finite periods of time using only a sub-graph of G denoted $G'(t) = (P(t), E')$, where E' represents the set of edges used for communication.

Let $sp_{i,j}$ be a loop-free shortest path between two participating nodes v_i and v_j, and $delay(sp_{i,j})$ be the communication delay on this path. If we denote by $delay_{max}(G'(t))$ the upper bound on the delay in the environment at a particular moment in time, we have

$$delay_{max}(G'(t)) = max\ \{delay(sp_{ij})\},\ \text{where}\ v_i, v_j \in P(t)$$

Knowing the upper bound between any two participants in the graph allows us to control the delays in the environment by accepting or rejecting other participants. Moreover, since there are multiple possible sub-graphs, we can optimize communication based on different metrics. Since our primary interest is to minimize the end-to-end delay, in the following section we will focus on a theoretical approach to perform such minimization.

3.2.2 Core-Based Tree - Minimizing the Delays among Participants

Reducing the maximum end-to-end delay between two participants will naturally improve the synchronization capabilities of the ASA. Minimizing the maximum end-to-end delay can be achieved by finding the connected sub-graph $H(t) = (W,F)$ that satisfies the condition:

$$delay_{max}(H(t)) = \min\{delay_{max}H'(t) \mid H'(t) \subseteq G'\}$$

where *H'(t)* is a connected sub-graph of *G'* .

Consider a weighted graph in which the weights represent the delays between nodes. The Core-Based Tree (CBT) [96] method guarantees that the path between any joining/new node and the core is the shortest. However it does not guarantee the optimal path between any two nodes. In Section 3.2.4.1 we introduce an additional rule to further bound the delay between two participants. A major advantage of the CBT approach is that only one multicast tree is created per group which reduces the overhead in the distributed VE.

A CBT is constructed incrementally. Initially, a node is chosen (i.e. the core) via a bootstrap mechanism. Each interested participant will send a *JoinRequest* message to the core. The address of the core node is advertised and is well known. The message is sent through the shortest path from the new participant to the core node, using the existing routing protocols. Along the shortest path the message can reach either nodes which are already part of the current communication sub-graph (i.e. the CBT) or nodes that are not part of the tree. If the message reaches a node *k* that is part of the CBT, the forwarding process will stop and the incoming link will be added to the *forwarding cache* of node *k*. An acknowledgement message will be sent back to the new participant and it will become part of the CBT. When the message reaches a node *k* that is not part of the CBT, *k* will redirect the message to the next hop along the shortest path toward the core node and will cache the incoming node and incoming interface (e.g. port

number) in a temporary storage, waiting for an acknowledgement message. Once *k* receives an acknowledgement it adds the incoming interface to the forwarding cache and redirects the acknowledgement to all nodes listed in the temporary storage. *k* also sets the node who sent the acknowledgment to be its parent node and becomes part of the CBT.

The CBT algorithm assures that each interested node can reach the core node using the shortest path. Moreover, the communication sub-graph is a tree due to its method of construction (i.e., all nodes are connected and each one has just one parent, except the initial core node which has no parent).

In what follows we describe the CBT algorithm. The initial CBT consists of one core node, i.e. the communication sub-graph is $G'(t_0)=(P(t_0),E')$, where $P(t_0) = \{v_{core}\}$ and $E'=\varnothing$. When a participant *v* joins the environment, it will send a *JoinRequest* to the adjacent node *v'* which is along the shortest path (i.e. in terms of delay) from *v* to the core node v_{core}. Below is the pseudo code for the procedure:

if $(v' \in V')$
    add v to the forwarding cache
    send JoinAck to v
else
    $temp_{v'} \leftarrow temp_{v'} \cup \{v\}$;
    send JoinRequest to first node along the shortest path to v_{core}
end if

When the node v receives a JoinAck from node v' the following procedure is executed:

$V' \leftarrow V' \cup \{v\}$
$E' \leftarrow E' \cup \{e\}$, *where e is the edge connecting v and v'*
add v' to the forwarding cache in v
set v' to be the parent of v
send JoinAck to $\forall u \in temp_v$
$temp_v \leftarrow \varnothing$

An example of the CBT construction is depicted in Figure 10. Participant p_3 wants to join the environment and sends a *JoinRequest* towards the core p_{core}. Intermediary node p_2 processes the request and sends the message towards p_{core} (Figure 10a). The core replies with an acknowledgement and it is added in the forwarding cache of p_2. In turn p_2 sends the acknowledgement to p_3. p_3 marks p_2 as its parent and also p_2 marks p_{core} as its parent as illustrated in Figure 10b. The forwarding caches of p_3, p_2 and p_{core} will be $\{p_2\}$, $\{p_3, p_{core}\}$ and $\{p_2\}$ respectively. When p_1 wants to join, its message will be captured by p_2 who is already part of the CBT. p_2 replies to p_1 directly and adds it to its forwarding cache. In turns p_1 marks p_2 as its parent as shown in Figure 10c.

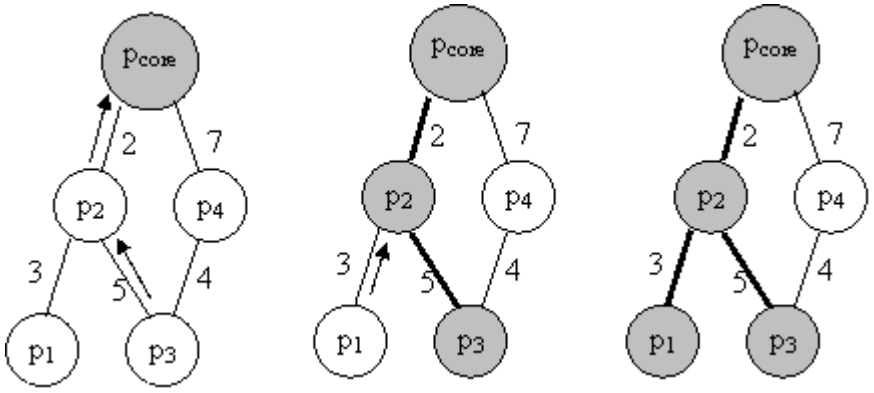


Figure 10. Core-Based Tree construction

In the next section we look at distributed interactive VE participants' behaviors and define several types of nodes based on their functionality. The generality of these definitions makes them applicable to any distributed interactive environment or simulation.

3.2.3 Hybrid Nodes with Real-Time Sensors

Distributed interactive VEs involve the interaction of several remote participants. With advances in sensor technology, we envision that in future systems a significant amount of data will be collected from sensors and devices attached to the participants. While some of the participants actively modify the shared scene, other participants are passive, in the sense that they do not interact with the shared scene. We define two categories of participants: *active participants* and *passive participants*.

An active participant triggers changes in a virtual object state from an interface (e.g., GUI or through a sensor). Passive participants do not trigger any modifications of the shared scene; they just receive visual, haptic and/or audio feedback from the environment. The active or passive attributes of the participants can dynamically change in time. An active participant may become passive and vice-versa depending on the collaboration needs.

A node in the distributed system allows a participant to interact with the components of the VE. Without loss of generality, we will consider that each node has a set of *sensors* that provide position and orientation information and other peripheral devices that allow state changes for the objects in the shared scene. The discussion can easily be extended to other types of sensors (e.g., haptic) that can be part of the distributed system's resources. The system must ensure that the data captured by each node's sensors is distributed with minimum delay to all interested participants to maintain the shared state consistency. Moreover, each node in the system will need to exchange its sensory data with all or a predefined subset of nodes.

A pure centralized data distribution approach (e.g., client-server) would not be efficient because of the additional delay associated with the data collection stage, followed by the data distribution. An atomistic peer-to-peer approach would not fit either because of the additional overhead in data distribution (see Related Work). Each node would have to exchange data with all the other nodes. As a fundamental property, the nodes in a sensor-based distributed system may act as data producers, consumers and distributors, simultaneously. Based on the above discussion we define four types or running modes for the distributed system's nodes:

- *Active nodes* - An Active (A) node represents an active participant in the virtual environment. Each active node collects data from its sensors and is responsible for making the data available to interested peers as quickly as possible.
- *Passive nodes* - A Passive (P) node does not inject any information into the virtual environment. The node uses the information provided by active and forward nodes to render the shared scene.
- *Active Forward node* - As the name indicates, a forward node forwards data. The forward node can be *active* or *passive*. An Active Forward (AF) node injects its own data into the system, as well as forwards other nodes' data.
- *Passive Forward node* - A Passive Forward (PF) node does not inject new information in the system. These nodes act as pure data forwarders.

In what follows we propose an investigation of the possible states and transitions of a node in a distributed interactive VE. Let's denote the states of a node as: {A, P, AF, PF} (i.e. "A" stands for "*Active*", "P" for "*Passive*", "AF" for "*Active Forwarder*" and "PF" for "*Passive Forwarder*"). Let's denote the conditions that trigger the change in state using a binary representation

- 00 - the state changes from *inactive* to *active*
- 01 - the state changes from *active* to *inactive*
- 10 - the state change from forwarding *off* to *on*
- 11 - the state change from forwarding *on* to *off*.

Twelve transitions may occur. Table 4 summarizes the possible transitions and the triggering conditions.

Table 4. Possible transitions for a hybrid node

Current State	Next State	Condition(s)
A	P	01
A	AF	10
A	PF	01 followed by 10
P	A	00
P	AF	00 followed by 10
P	PF	10
AF	A	11
AF	P	01 followed by 11
AF	PF	01
PF	A	00 followed by 11
PF	P	11
PF	AF	00

The behavior of a node can be represented as a state machine as illustrated in Figure 11.

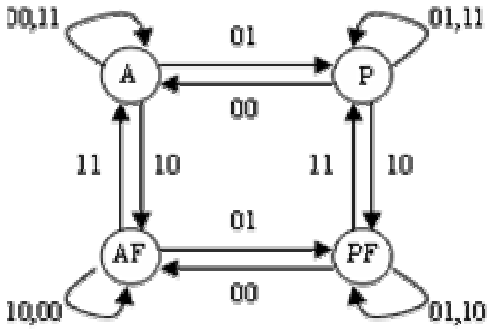


Figure 11. State machine representing the hybrid node behavior

Based on the above discussions, we now introduce a data distribution scheme targeted towards the development of sensor-based distributed interactive VEs.

3.2.4 Hybrid Data Distribution Scheme

The *dynamic membership* property of the environment, as well as the constraints of the interactive environment, pointed us towards a hybrid between a client-server and peer-to-peer model. The data distribution scheme is driven by the assumption that each active node manages a small network of potentially real-time sensors as illustrated in Figure 12. We build an overlay network at the application level, in which we employ the CBT techniques for multicast tree construction as discussed in the previous section.

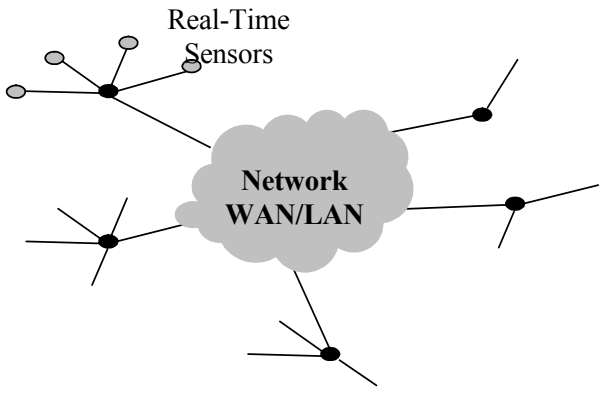


Figure 12. Distributed interactive VEs nodes and sensors

The first node that becomes active (i.e., an "*A*" node) will advertise the availability of information to all participants. Interested participants will join the multicast tree cored at the active node. At a particular moment in time the distributed interactive environment may contain several active nodes and one core. Figure 13 provides a snapshot in time of the distributed system.

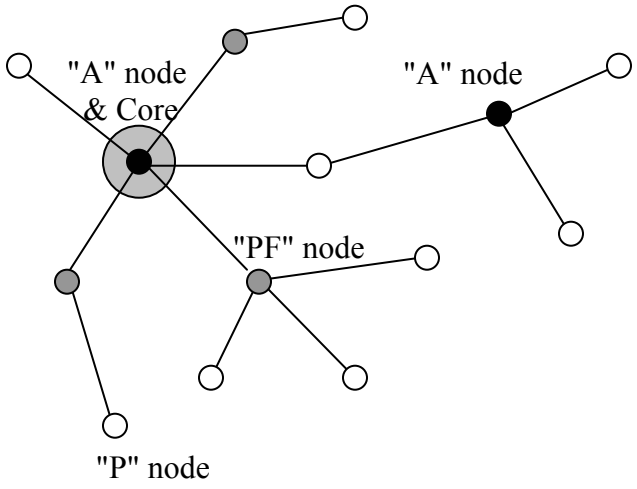


Figure 13. Snapshot of a distributed interactive sensor-based VE

3.2.4.1 The Control Protocol

To maintain its position and connections in the multicast tree, each node periodically sends a heartbeat message to its parent node. This *keep-alive* mechanism operating between adjacent on-tree nodes, allows the management of the dynamic membership behavior, as well as failure recovery. A node that is leaving the environment will inform its adjacent neighbors.

To improve the real-time behavior of the system we must maintain the minimum possible communication delay between any pair of nodes. An approach is to create an overlay network using the CBTs. The CBT approach guarantees the shortest path between the core and the other nodes. However to maintain a bound on the delay between participants we propose the following rule:

Rule. A node "*i*" cannot change its state from "*P*" to "*PF*" or from "*A*" to "*AF*" and become a forwarder for an active node "*j*" if the accumulated delay on the path to the core exceeds *MaxValue*, where *MaxValue* is an application dependent parameter.

In other words a new participant can not join the tree if the accumulated delay from the node where he will connect to the core falls over a certain threshold. This threshold is an application domain and communication infrastructure dependant value and can be obtained empirically through experiments.

3.2.4.2 Participant Joining the Interactive VE

The above model has a fundamental bootstrap problem: How to join? Without a central server there is no easy way of determining resource availability in advance. To solve the peer-discovery situation, a new node broadcasts a query (i.e. a JoinRequest) and awaits response. If the node that

replies is not already a forwarder (i.e. "*AF*" or "*PF*") it will become one as long as the *Rule* is not violated. Otherwise the new node will have to wait until the constraints are met.

A complementary problem is: How does a node register itself to receive data from an active participant? To do that the node will join the current CBT.

3.2.4.3 Participant Leaving the Interactive VE

While joining does not cause massive distortions in the environment, leaving might cause major problems, especially when it is not premeditated, e.g. in case of a node crash or a link failure. In what follows we analyze the situation for each potential state of a node. We'll start with the easier cases first.

If a "*P*" node leaves the environment it will notify its parent so that its entry can be removed from the forwarding cache. If the "*P*" node crashes, its parent will detect the absence of the *heart-beat* message and will remove it from its cache. The "*P*" nodes are always leaves in the trees so their removal does no incur any overhead.

Things become more complex with "*A*" nodes. If an "*A*" node intentionally leaves the environment or if the following state changes occur, $A \rightarrow P$ or $A \rightarrow PF$, it will send a signal to all its children. The message will propagate in the CBT allowing all the listeners to remove the unnecessary entries from their caches. If a crash or link failure occurs, it will be detected by its

children through the *heart-beat* mechanism. The changes will be propagated in the tree to all the listeners.

Let's analyze the scenario when an "*A*" node leaves or crashes. If there is only one "A" node in the VE the interaction in the environment is null, hence the virtual environment degenerates into a non-interactive one (i.e. of course with the potential to become interactive again).

A more interesting situation is the removal of "*AF*" or "*PF*" nodes. The "*AF*"/"*PF*" node crash or intentional leaving incurs additional complexity since the forwarding aspect implies that they have a non-null set of children. As in the "*A*" node case, the children will detect the node failure. Each child node has two options for failure recovery: it can either attempt to re-join the tree by sending a *JoinRequest* message to its core nodes, thus keeping the failure transparent to the rest of the down-stream branch, if any; alternatively as a result of the above mechanism failing, the child node can send a *FlushTree* message downstream allowing each node to independently attempt to re-attach itself to the tree, possibly via a better route than before.

4. TESTBED COMPONENTS AND IMPLEMENTATION

4.1 Overview

The proposed ASA and the data distribution scheme were deployed in a testbed specifically designed and implemented for carrying out research on distributed interactive VEs. In the following sections we present the testbed hardware and software components.

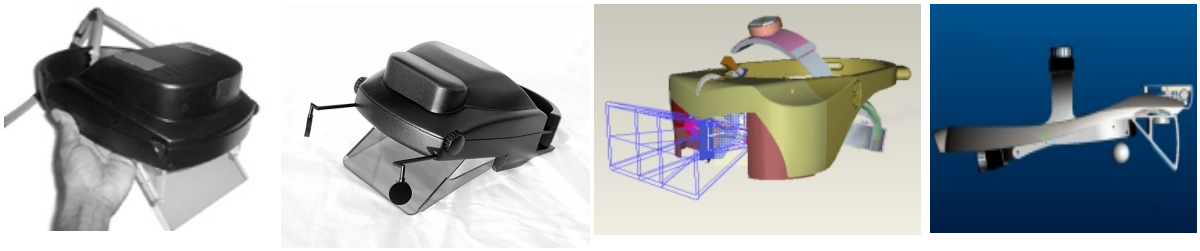
The testbed was implemented for two reasons. First, the available network simulators (e.g. NS-2[97], PARSEC[98], SSF[99], NetSim, NEST, Gosip) are limited in their design to protocol simulations on different topologies and do not support the integration of an interactive VE application. Secondly, in order to use these simulators in our experimental context an abstraction of the distributed VE must be done and assumptions must be made. These abstractions and assumptions would result in non-deterministic behavior and diminish the significance of the experimental results.

4.2 Testbed - Hardware Components

Each node consists of a HMD, a Linux based desktop/laptop, and a quasi-cylindrical room, called an Artificial Reality Center (ARC), having walls covered with retroreflective material.

4.2.1 3D Visualization Hardware Setup - HMD & ARC

The HMDs designed and integrated in the Optical Diagnostics and Applications Laboratory employ extremely lightweight (< 8g per eye) and compact custom-designed projection optics to provide computer-generated images to the participant with a field of view (FOV) that may be as large as 90 degrees. The displays were designed and intended for indoor settings. The use of projection optics as opposed to eyepiece optics is the key to compactness and negligible distortion. Optics with no distortion helps increasing image quality while keeping low cost. Figure 14 shows the HMD technology evolution since 1999 when it was first conceived [100].



(a) **(b)** **(c)** **(d)**

Figure 14. Head-Mounted Displays
(a) First prototype; (b) 2001 AR Display & Face Recording; (c) 2003 Side Mounted AR display; (d) 2004 Ultra-compact model

The first prototype (a) suffered from bulky electronics, relatively low resolution (i.e. 4 arc min), low brightness due to the use of 640 x 480 pixels backlight LCDs, and a weight of 750g which is high compared to the 8g optics per eye. The "heavy" weight was imposed by the shell of the

HMD required to package the bulky electronics. The second prototype (b) added a teleportal capability (THMD) [101], which consisted of about 1 inch convex mirrors mounted side-frontal of the user's head and coupled with temple-mounted lipstick cameras to capture stereoscopic images of the face. A recently engineered more compact system with more compact electronics and side-mounted optics is shown in (c). An even more compact prototype (<600g) based on Organic Light Emitting Displays (OLED) and extremely compact electronics is shown in (d).

The optical material placed in the real environment allows users to view computer-generated objects embedded in the environment. The material (e.g., manufactured by 3M) is retroreflective. In this manner, the material directs the stereoscopic image pairs projected from the HMD back to the eyes of the user, as shown in Figure 15, allowing stereoscopic visualization. The material is flexible and can be used to partially or completely surround users or to cover surfaces or objects of various shapes within the environment.

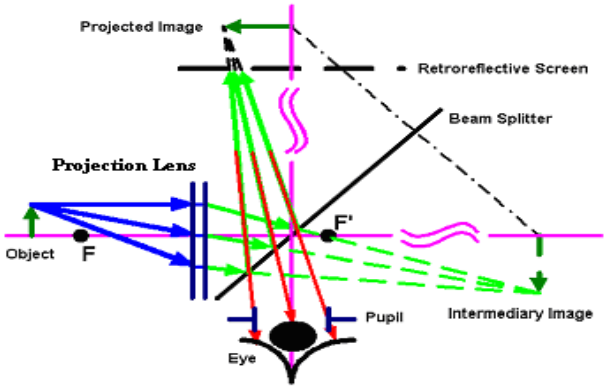


Figure 15. Image formation

In this case we have designed an Artificial Reality Center (ARC) [102], with walls consisting of a set of panels covered by the optical material as illustrated in Figure16 and Figure 17.

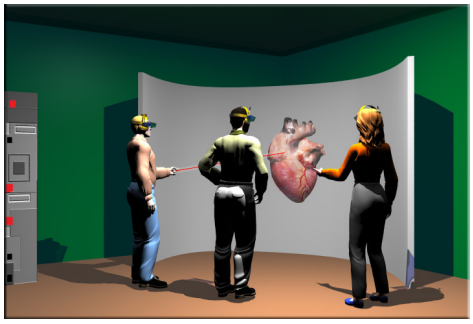


Figure 16. The ARC concept

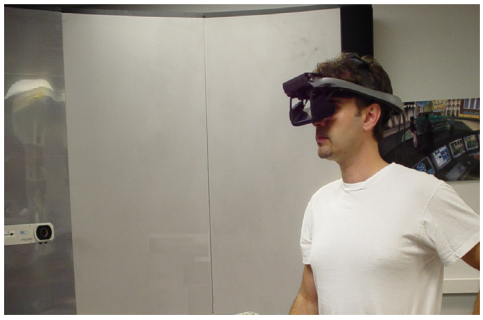


Figure 17. The ARC implementation

Several ARC rooms can be interconnected on a network as described in Fig. 18. This setup enables remote collaboration through distributed applications that span the entire virtuality continuum [103].

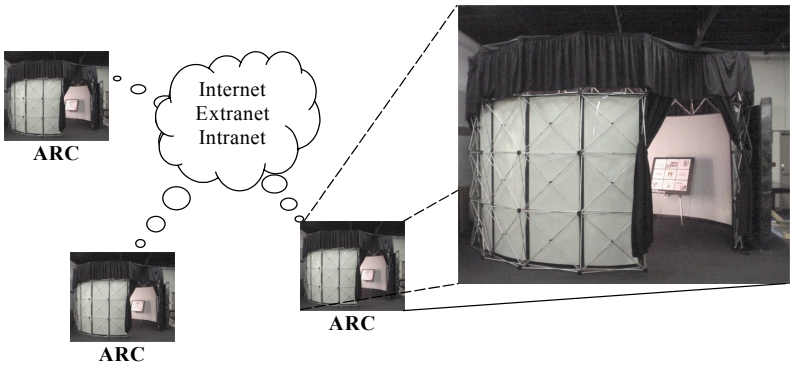


Figure 18. Artificial Reality Center

4.2.2 Sensors - Polaris NDI Optical Motion Tracking System

The sensor used is an optical motion tracking system built by Northern Digital™. Polaris© [104] is a deployable optical tracking system that provides accurate orientation and positioning (6DOF) information in real-time. The system has an update rate of up to 60 Hz.

By *tracking probe* we denote a rigid configuration of markers, as illustrated in Figure 19, that are attached on the real objects in the scene to track their position and orientation in 3D space [105].

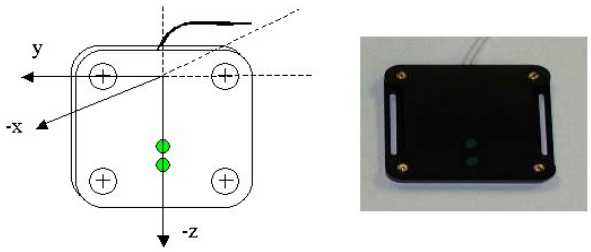


Figure 19. Tracking probe with 4 active markers

The *position sensor* shown in Figure 20 detects the position of the tracking probe while in its tracking volume. The tracking volume has a conical shape with height 1.5 meters and a base radius of 0.5 meters. Newer versions of this system provide a position sensor with a larger pyramidal tracking volume. The sensor has two infrared cameras that detect the position of the active markers (IRED) or the position of the passive markers by reflection.

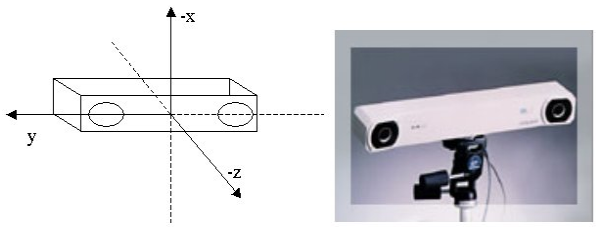


Figure 20. Polaris position sensor

4.2.3 Nodes - Heterogeneous Workstations

The nodes in the system are composed of desktops and laptops. The network cards on all nodes maintained up to 100Mbps connections. The table below contains a brief specification of each node's hardware components.

Table 5. Hardware systems attributes

Node No.	Arch.	CPU (GHz)	RAM (MB)	GPU (GeForce)	Network Card (10/100 Mbps)
1	Desktop	1.5 AMDx2	1024	4 Ti 4600	3Com 3C920
2	Desktop	1.7 AMDx2	1024	4 Ti 4600	3Com 3C920
3	Desktop	1 AMD	512	2 Mx	Netgear FA310 TX
4	Desktop	1.7 Intel	512	4 Mx 440	C.Net Pro200WL
5	Laptop	2 Intel	1024	4 Go 440	3Com 3C920
6	Desktop	2.8 AMD	512	4 Ti 4200	3Com 3C996-BT

4.3 Testbed - Software Components

The algorithm performance is slightly affected by the platform where the implementation is deployed. The Windows™ OS is challenging to control at a fine granularity level, therefore we have deployed the algorithm implementation on a Linux based platform. To create a virtual scene we are using Open GL Performer 2.5 on a Red Hat 8.0 OS platform. The GUI was developed using the GIMP Tool Kit (GTK 2.0). For a brief description of the APIs and SDKs involved see APPENDIX B.

4.3.1 Software Components Developed

We have developed a set of object oriented libraries under the name Distributed Augmented/Artificial Reality Environment (DARE) [102] see APPENDIX B.

Delay Measurement Probe

The delay measurement probe is a software module that allows inter-node delay computation. It is an important component of the ASA algorithm. Upon joining the environment, each node uses this module to compose and transmit an *EchoRequest* packet. The message contains an *ID* field, which is the Linux process ID, and a sequence number, which is an ascending integer. The first eight bytes of the data portion are used to hold a Linux "*timeval*" structure for round-trip time delay computation.

Control Package Objects

The synchronization is achieved through message exchanges. We have implemented a control package object (CPO) class that contains the attributes of each virtual object in the shared scene like position, orientation, color and actions (i.e. the set of actions applied on the object and the current values of its attributes). Whenever a new virtual object becomes part of the shared scene, an associated CPO is instantiated. If the object is composed of several parts linked by articulations, each part has its own CPO.

The CPOs circulate in the distributed system triggered by the participant's interaction on the virtual components of the scene. An event based mechanism is used. Whenever the participant interacts on the shared scene from a GUI, an event triggers the distribution of the associated CPO to all the interested participants using the data distribution scheme described previously.

Data Collection Module

To assess the ASA efficiency we employ a quaternion analysis method described in APPENDIX A. Initially, a script based on the Network Time Protocol (NTP) [106] is executed to synchronize the internal clock on each node within millisecond accuracy. Furthermore, to reduce intrusiveness, the data (e.g. quaternion components) is collected in memory during the experiments and dropped to a file at the end. This process is executed independently on each node during the simulation.

4.3.2 Hybrid Nodes Design

The proposed design falls in-between the atomistic peer-to-peer and client-server model. Each node is autonomous and fully manages its resources and connectivity through a set of software agents: a GUI agent, a sensor agent, a rendering agent and a behavior agent. These agents run on each node and trigger the node behavior, i.e. a node can switch among any of the four modes described in the previous chapter, i.e. *A*, *AF*, *P*, *PF*.

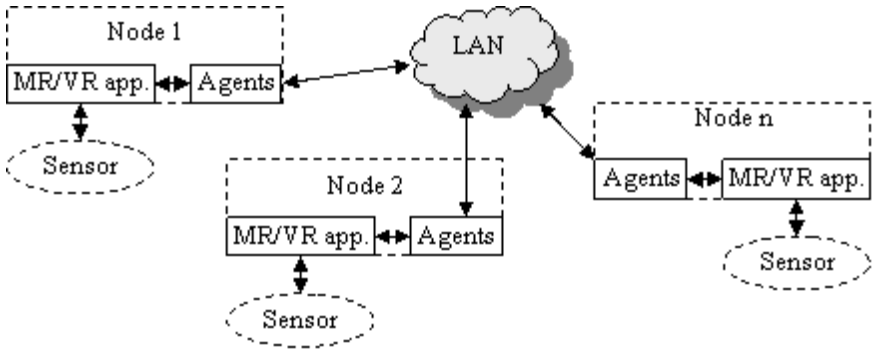


Figure 21. Distributed interactive VE deployed on a LAN

We consider two interaction scenarios common in distributed interactive VE applications: the participant uses a GUI for interaction or the participant interacts through the motion tracking sensor. In the first case, the GUI agent becomes active and makes the data available to the behavior agent; while in the second case the sensor agent pulls data from the sensor(s) attached to the node, converts it into an appropriate format and makes it available to the behavior agent. If

requested by other participants, the behavior agent will spawn a server thread making the data available to other nodes, hence propagating the local modifications to the shared scene. Otherwise, the participant's interaction will affect only the local copy of the shared scene. Regardless of the node mode, the rendering agent is active all the time, given that the scene has to be continuously rendered. In what follows we briefly describe each agent.

The GUI Agent

The GUI agent is responsible for displaying the GUI (illustrated in Figure 22) at the active nodes and collecting the participant interaction on the VE using an event-based mechanism.

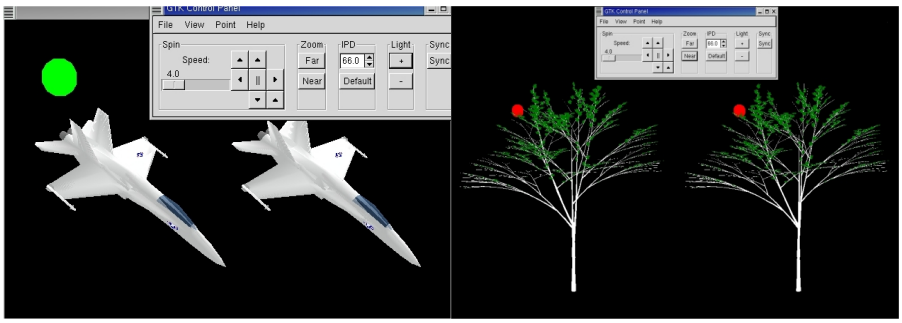


Figure 22. The GUI allows manipulation of the virtual objects and mouse based 3D pointing

The Sensor Agent

The sensor agent is responsible for collecting information given by the participant through the sensor(s) attached to the node (e.g. motion tracking sensor). The sensor agent is listening to specific ports for sensor activity. We have used the sensor agent in conjunction with the Polaris™

System to change the position and orientation of a 3D object in the scene by linking the object position in the virtual world with a tracking probe position in the real world.

The Rendering Agent

The rendering agent is responsible for rendering the shared scene on the output device (e.g. HMD) using the data collected from the participating nodes. The rendering is done using NVidia GeForce4 based cards on two channels, independently for the right and for the left eye, providing the inter-pupilary distance (IPD) as illustrated in Figure 23. Such rendering allows visualization of the virtual 3D objects through the HMD.

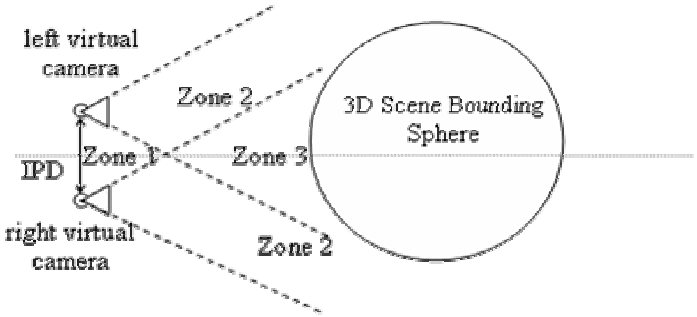


Figure 23. Virtual cameras for the left and right eye: zone 3 stereo view

The Behavior Agent

The behavior agent handles new incoming data requests for the "*A*" nodes and allows the node to switch among the four modes. When a participant interacts with the shared scene, the associated node becomes active.

The behavior agent spawns a thread that controls the activation of the server component of the node. A passive node runs in client mode "*consuming*" incoming data from the participating nodes. When a passive node becomes active, it means that it acts as a data producer and distributor for the local participant and for the other nodes interested in its data. It is the responsibility of the behavior agent to advertise that the node has available data from a particular sensor by a short broadcast to all the nodes. If a new participant is interested it will be added by the behavior agent to the node's consumers list based on the CBT algorithm, as discussed in the previous section. Each level on the path to the core implies additional delays, potentially leading to an unacceptable behavior since the system has to respond to the participant's actions at interactive speeds. Therefore the depth of the CBT can be limited based on the application requirements. If the core node cannot handle the request, or the depth of the tree is too high, the new request will be rejected and the potential participant may try joining the VE at a later time.

The behavior agent is activated by the participant interaction; hence, it maps the participant behavior onto the participant's associated node. In other words, if the participant is active, the associated node becomes active and ready to distribute interaction data. If the participant is passive, the associated node becomes passive. Figure 24 illustrates the various agents and their states on a passive as well as on an active node.

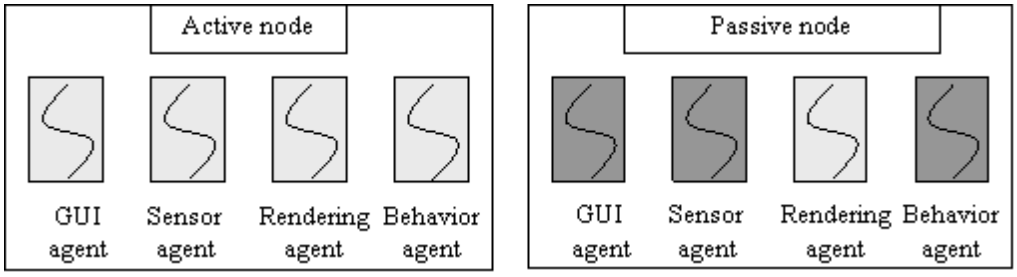


Figure 24. Dark shaded represents inactive agents, light shaded represents active agents

When the participant does not interact with the shared scene, the GUI agent and the sensor agent are deactivated and the node becomes passive.

5. EXPERIMENTAL DESIGN AND SETUP

5.1 Overview

We have performed several experiments to quantify the dependence of the algorithm efficiency on the number of participants and on the action velocities (i.e. indirectly communication latency). In what follows we describe the experimental scenario, the relationship between network latency and action velocity as it pertains to the experiments, the scalability investigation scenario as it relates to the number and type of participants and finally the assessment method.

5.2 Experimental Scenario

As participants wearing HMDs enter the ARC [103], they gradually start immersing themselves in virtuality. Initially, the participant's reality is augmented with 3D computer-generated objects; however, they can be also immersed in Virtual Reality. The virtual objects may appear to multiple remotely located participants, if they share the same scene. The participants are interconnected on a local area network.

Participants can interact with the virtual objects in two ways. Using a GUI with 3D pointing capabilities, they can manipulate these objects and they can point in the virtual space to different

parts of the objects (e.g., in the experiments we have used 3D medical models of mandibles and several 3D crosses) as illustrated in Figures 25, 26 and 27.

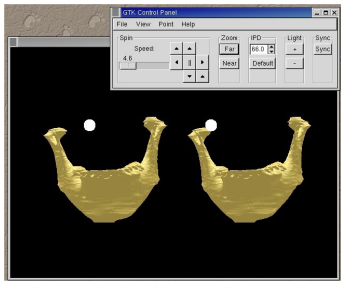


Figure 25. GUI

Figure 26. Local collaboration

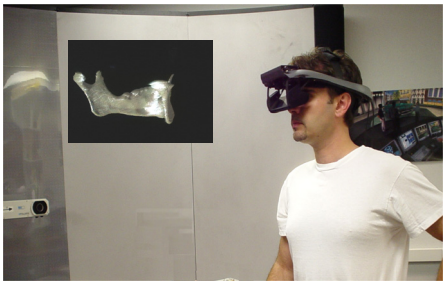


Figure 27. Remote participant

An alternative way of manipulation is through the motion tracking sensor. The participant has a tracking probe attached to his/her hand, and the position and location of the tracking probe is associated with the position and location of a virtual object in the scene.

Without loss of generality we limit the participant interaction to rotations of the object around its coordinate axis. An assumption of the experiments is that object ownership is not changeable. Active participants apply a set of consecutive actions (i.e. rotations of the object around its coordinate axis) using the GUI. The action's attributes (i.e. angular speed) are set using the interface. Each new action triggers a CPO modification that is distributed in the system.

We have compared three scenarios:

1. The information from the CPO is not used at all. As a consequence the orientation drift among different views of the participants accumulates, confirming the importance of consistency maintenance algorithms, as well as the drift level achieved with this configuration.
2. The orientation of the virtual object is corrected at each remote participant using the information from the CPO after each event is generated.
3. The orientation of the virtual object is corrected using the proposed ASA.

Chapter 6 presents the experimental results for these scenarios. Prior to describing these, we need to discuss an additional parameter that allows us to simulate the behavior of the algorithm on a higher delay infrastructure.

5.3 Network Latency vs. Action Velocity

To investigate the outcome of the network latency, we repeated the experiments at different action velocities, given that the drift value for an object is the product between the action velocity applied on that object and the network latency. For example, let's assume a simple scenario consisting of two nodes, an A node and a P node, representing an active and a passive participant. Suppose the average delay between these nodes is $t_{ij}=0.2ms$ and the active participant applies an action (e.g. a rotation around axis) on a 3D object in the shared scene with the angular velocity $\omega = 10$ *degrees/sec*. The angular drift in this case will be given by

$$\alpha_{ij} = \omega\, t_{ij} = 0.002 \text{ degrees}$$

On a higher delay infrastructure in which the delay is 20ms, the same action would produce a drift of

$$\alpha'_{ij} = \omega\, t'_{ij} = 0.02 \text{ degrees}$$

While keeping the same delay (e.g. $t_{ij} = 0.2$ *ms*), the drift increase can be simulated by increasing the action velocity, i.e., $\omega = 100$ *degrees/sec*. Therefore, in order to simulate higher latency networks in our experiments, we vary the action angular velocity from 1 degree/sec to 100 degrees/sec. In this way, we can simulate the behavior of the ASA under latencies of up to 20 ms on a 0.2 ms average delay network.

5.4 Active vs. Passive Participants, Scalability

To investigate the scalability of the algorithm as regards the number of passive participants, several sets of experiments were performed. The first set contained two nodes, an *A* node and a *P* node. The second set contained three nodes, one *A* node and two *P* nodes We gradually increased the number of passive participants to five nodes, as illustrated in Figure 29.

These experiments allow us to see how the number of participants affects the synchronization results. Since the algorithm is distributed and the drift computation is done remotely by each participant, we expect only a slight degradation as the number of passive participants increases.

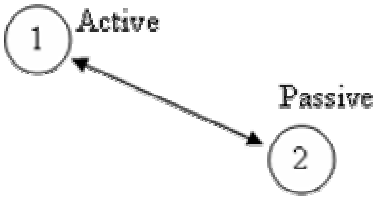


Figure 28. Two-node setup

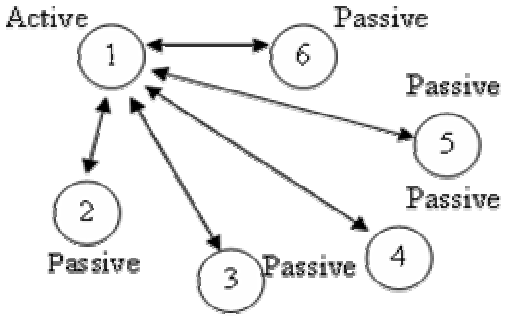


Figure 29. Six-node setup

Furthermore, to investigate the data distribution scheme we increase the number of active participants consecutively from one to six. In the last experiment the interactions of the participants at nodes 3 and 5 are captured through the optical motion tracking sensors attached to the nodes, as shown in Figure 30. The participants at nodes 1, 2, 4, 6 interact on the objects in the shared scene through a GUI. They act on different virtual objects in the shared scene, i.e., concurrent access to a virtual object is not considered.

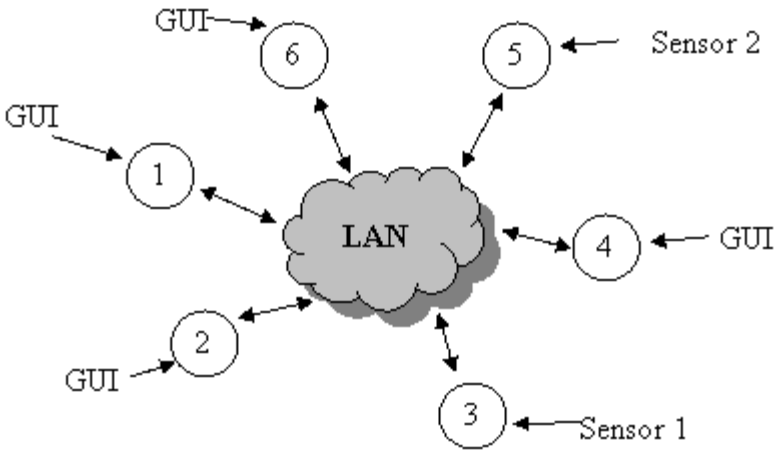


Figure 30. Six-active-participant setup

The experiment consists of seven steps:

1. All five users join the distributed interactive VE.
2. The participant at node 1 starts interacting with the virtual objects in the scene using the GUI.
3. The participant at node 2 starts interacting with the virtual objects in the scene through its GUI.
4. The participant at node 3 starts interacting with the virtual objects in the scene using the tracking probe attached to Sensor 1.
5. The participant at node 4 starts interacting with the virtual objects in the scene through its GUI.
6. The participant at node 5 starts interacting with the virtual objects in the scene using the tracking probe attached to Sensor 2.

7. The participant at node 6 starts interacting with the virtual objects in the scene through its GUI.

In each step, fifty angular drifts of one of the shared virtual objects are recorded at each node. This procedure is repeated, resulting in a total of 250 angular drift measurements between two nodes (i.e. for measurements we collect data for an arbitrary pair of nodes) for the entire simulation.

5.5 Distributed Measurements, Assessment Method

Data collection imposes additional problems. To be able to objectively compare the drift as seen by the VE participants, the data must be time-stamped. Since the data at each node is time-stamped with the local clock and since these clocks drift from each other in time, the initial step in data collection is to synchronize the clocks on the nodes. We employ the NTP to synchronize the distributed system clocks at millisecond accuracy just before the measurements are taken.

Another issue is the frequency of measurements. Taking the measurements too often will affect the distributed VE. Our strategy regarding measurements is to collect in memory the components of the orientation of the 3D object at each participating node after the participant interaction (i.e., after each new action is applied).

To assess the efficiency of the ASA, the amount of orientation drift for a shared 3D object is computed. The drift between the node that acts on the object (the *A* node) and each of the other participating nodes (nodes that see the interaction) is computed. We have focused our experiments on the assessment of the *orientation drift*. A similar assessment can be done for the object's position.

To emphasize the process we describe a simple scenario. We use two nodes (participants) sharing the same virtual 3D scene, with one acting as an *A* node and the other as a *P* node. The GUI available at the *A* node allows the participant to change the object *orientation* by applying rotations around the Cartesian axes. The participant at the *A* node generates events from the interface, and each time an event is generated, the object's orientation at both sites is recorded. Because of the network latency, different vectors at each node will describe the orientation of the object. The rotations can be easily expressed using quaternion notation.

Let q_s express the rotation of an object at the *A* node and let q_c express the rotation of the same object at the *P* node. Both participants see the same virtual scene and the object should have exactly the same orientation. To quantify the difference between the orientations of the object as rendered on the *A* and *P* nodes, we can compute the correction quaternion q_E, as shown in Appendix A, every time the user triggers a new action. The correction can be expressed as:

$$q_s = q_E q_c \qquad \text{Equation 15}$$

and thus,

$$q_E = q_s q_c^{-1} \qquad \text{Equation 16}$$

where

$$\alpha = 2\cos^{-1}(\omega_E) \qquad \text{Equation 17}$$

The angle α represents the drift between the orientations of a 3D object as rendered by the nodes.

6. RESULTS AND ANALYSIS

In this chapter we present and discuss the experimental results. Two sets of experiments were performed. In the first set we investigate the behavior of the ASA as the number of passive participants increases while in the second set we perform the same analysis as the number of active participants increases. In each set we examine three scenarios. In the first scenario there is no compensation for drift correction since we try to observe the behavior of the drift in our experimental setup. In the second scenario, the CPOs update the orientation of the shared virtual objects after each action initiated on them by the participants. The third scenario consists of employing the proposed Adaptive Synchronization Algorithm (ASA) consequently the communication latency between each pair of participants is taken into consideration.

6.1 Varying the Number of Passive Participants

6.1.1 Two Node-Setup

In the two-node setup a participant (node 1) rotates the shared virtual object by applying consecutive actions from a GUI. A remote participant (node 2) visualizes the same shared object. To simulate higher latencies the participant interactions (rotations) on the object are performed at different speeds (10, 50, 100 degrees/second). In the first set of experiments we've computed the

orientation drift as the participant applies actions on the virtual object without any compensation. The reason for the measurements is to obtain a reference for the drift magnitude and observe its behavior as a participant interacts on the shared scene. Figure 31 illustrates the variation of the orientation drift value and its trend line, while a set of fifty consecutive actions (random rotations) were applied on the virtual object.

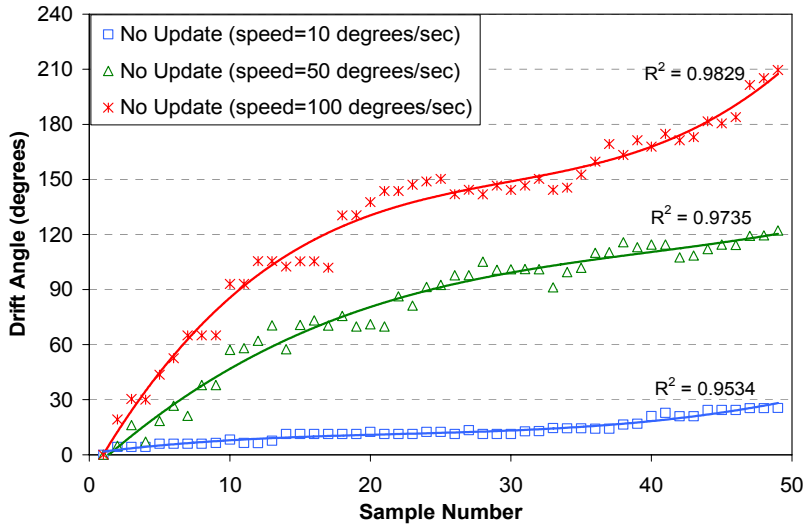


Figure 31. Drift behavior at node 2 with no drift compensation

In the second set of experiments we use the information in the CPO to update the position of the virtual object after each change in action attributes (Event Update method). These changes are generated by the participant while interacting on the virtual object. As seen in Figure 32, the orientation drift is maintained at a fairly constant value. The trend lines use a high order (sixth degree) polynomial fit.

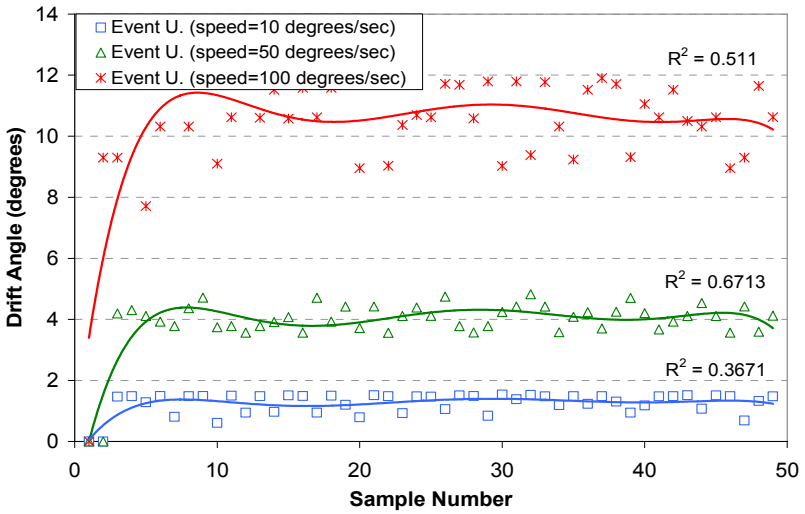


Figure 32. Drift behavior at node 2 using the Event Updates method

In the third set of experiments, we have employed the ASA to compensate for the communication delay and jitter. The information from the CPO is combined with the information carried by the delay measurement probe to compensate for the drift. As seen in Figure 33 the drift value is significantly decreased and kept at a constant level. A higher order polynomial fit was used for the trend lines. As in the case of Event Update, the trend line has a sinusoidal shape which has a negative effect on the correlation coefficient (*R*). The sinusoidal shape of the trend line can be explained as an effect of the buffering and other system threads at the network and operating system (OS) level. Moreover such a sinusoidal behavior can be associated with the result of applying a low-pass filter on a signal with noise. In this case the noise is the drift and the low-pass filter is the ASA trying to block the drift accumulation.

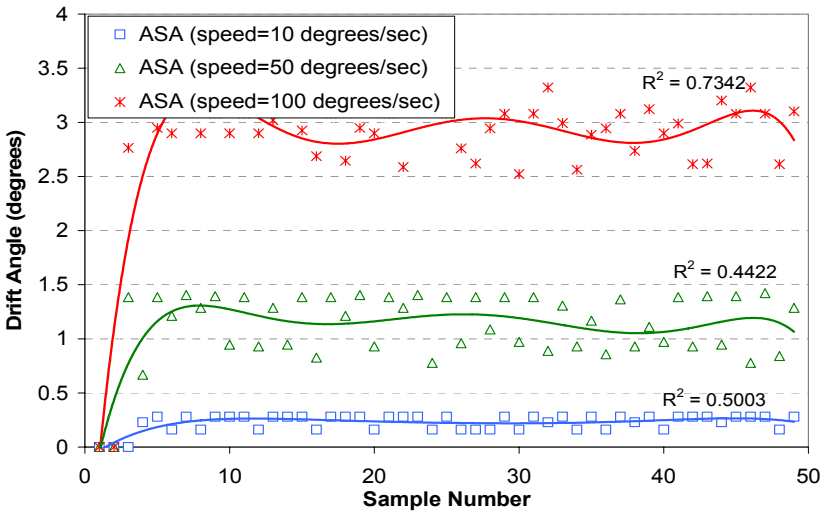


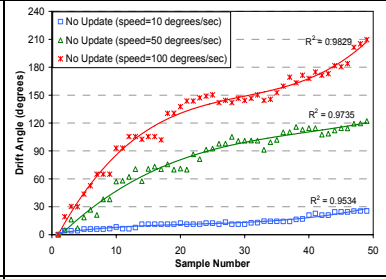
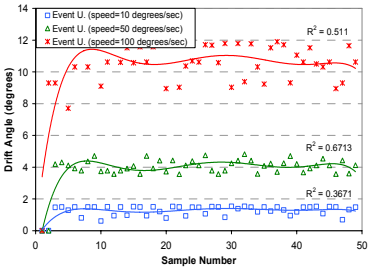
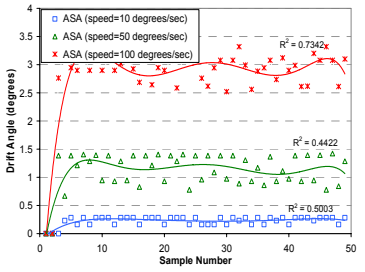
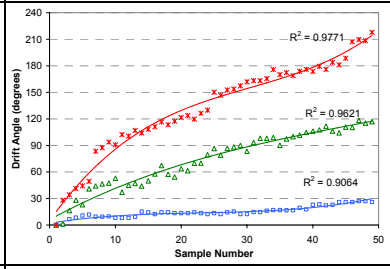
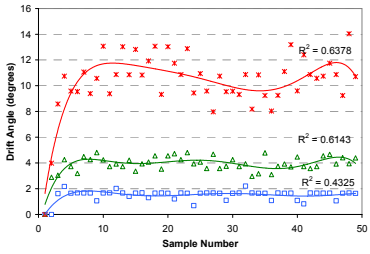
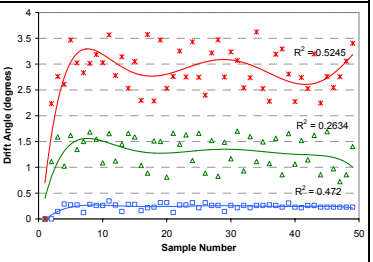
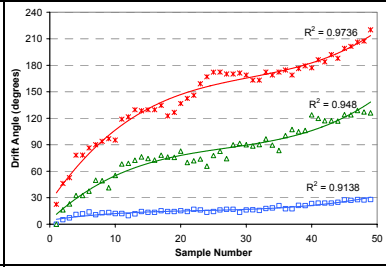
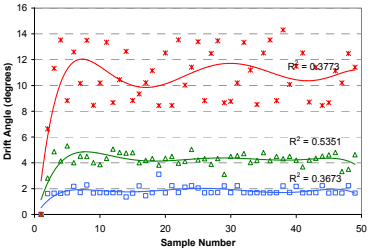
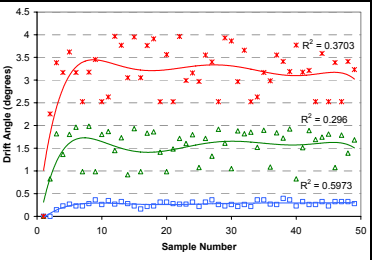
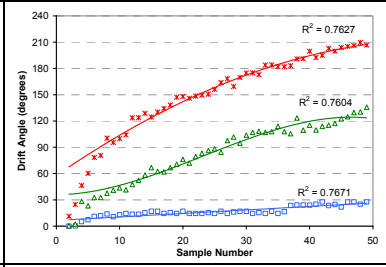
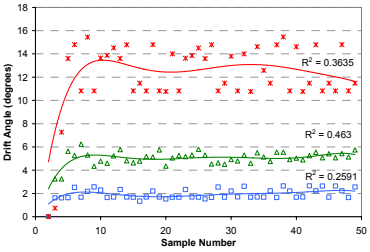
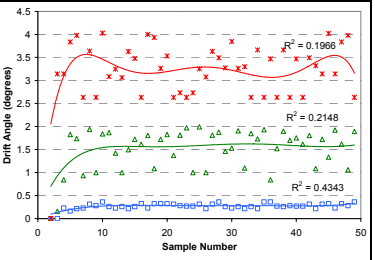
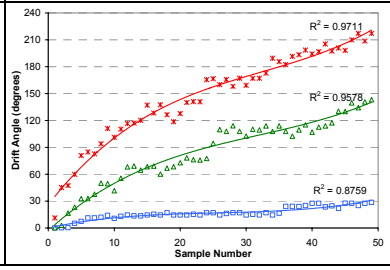
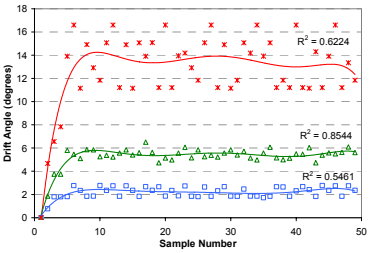
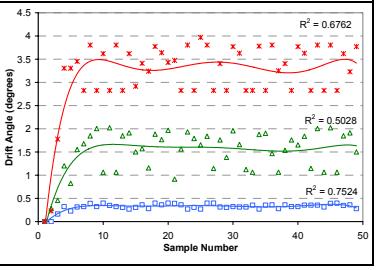
Figure 33. Drift behavior at node 2 using the Adaptive Synchronization Algorithm (ASA)

When the ASA is used, the drift angle is maintained at a constant value that is two orders of magnitude lower than the average drifts without update and approximately four times smaller than the average drift when the Event Update strategy is employed.

6.1.2 Three, Four, Five and Six Nodes Setup

We have increased the number of passive participants consecutively to two, three, four and five while maintaining the same active node (i.e. node 1, recall the hardware configuration from Section. 4.2.3). Table 6 represents the average orientation drift observed on node 2, as the number of passive participants increases. A slight increase in the drift value with the number of passive participants can be seen. The ASA keeps the drift value at constant levels.

Table 6. Node 2 drift comparison in the 2,3,4,5,6 nodes configurations

	No Update	Event Update	ASA
2			
3			
4			
5			
6			

6.1.3 Scalability Regarding the Number of Passive Participants

To investigate the scalability of the ASA, regarding the number of passive nodes, we define a metric analyzing the relationship between the number of nodes in the system and the measured orientation drift values.

Let ψ_i be the average drift value over all the participants, when $i+1$ participants are in the system. Without loss of generality, let us consider an action velocity of 100 degrees per second. In the case of a two-participant setup, results show that the average drift, ψ_1 equals 2.83 degrees, while in the case of a six-participant setup the average drift, ψ_5 equals 3.17 degrees. An algorithm with a low degree of scalability would have at least a linear increase in drift, i.e., ψ_n would equal $n^* \psi_1$. On the other extreme, a high degree of scalability would mean $\psi_n \approx \psi_1$. Using this metric, in the six-participant setup, a low degree of scalability would translate to ψ_5 equaling $5^*\psi_1$ or 14.15 degrees. However, the experimental results and trend lines in Figure 34 show that $\psi_5 \approx \psi_1$. Thus, the algorithm gives promising results in terms of scalability regarding the number of passive participants.

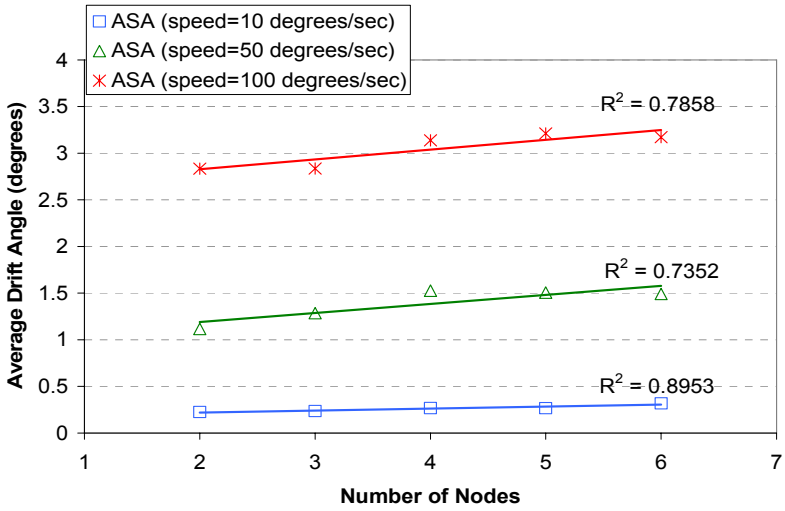


Figure 34. Orientation drift behavior as the number of passive participants increases (ASA)

In Figure 34, the average drift is potted as a function of the number of participants in the collaborative VE. The trend lines have been plotted using linear regression. The slope of the regression line increases slightly with the action speed. At 100 degrees/sec the slope value is 0.105 and it decreases to 0.097 and further to 0.021 for action speeds of 50 and 10 degrees/sec respectively. In the current scope of the experimental results, this shows that the ASA algorithm and the data distribution scheme are slightly sensitive to the network delay.

6.2 Varying the Number of Active Participants

Starting with the same configuration of six nodes, we have triggered active participants one by one, up to the extreme case when each node was active. Two of the participants were modifying

the position of the 3D virtual objects using Polaris position tracking sensors while the other four were manipulating the objects using GUIs. Each participant manipulates its own object.

6.2.1 Two, Three, Four, Five and Six Active Participants

Figure 35 illustrates the orientation drift value for a virtual object, as the number of active participants in the distributed environment increases, without any consistency management scheme. To be able to compare the results when active nodes join the VE, we have computed the orientation drift for one virtual object as displayed by node 2. This is then the drift between node 1 and node 2 experienced as a consequence of an increase in the number of active participants.

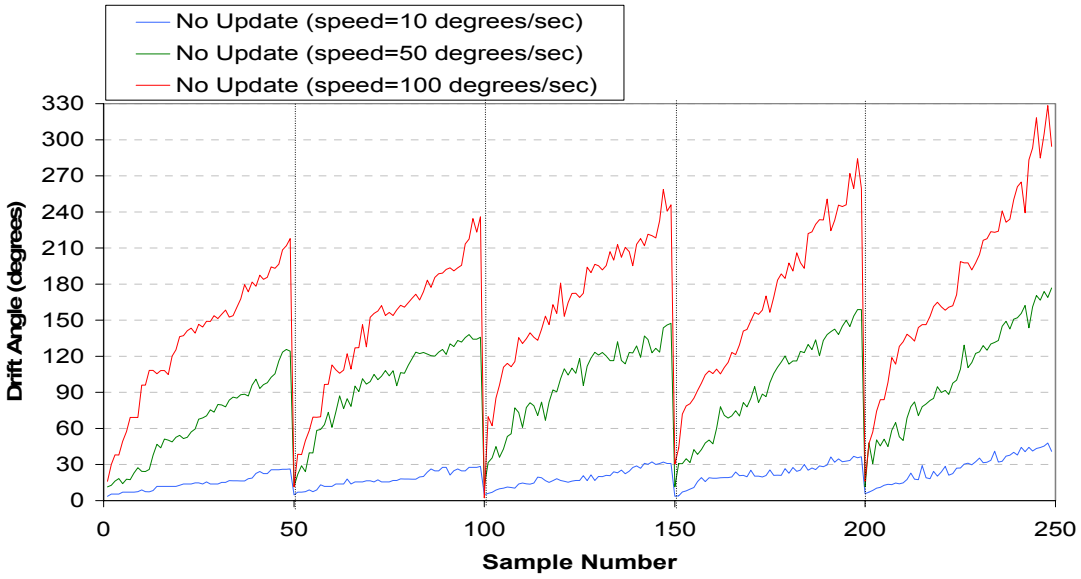


Figure 35. Orientation drifts between node 1 and node 2 without compensation when the number of active participants increases from 2 to 6.

The drift behavior on node 2, while the number of active participants increases, can be compared with the drift behavior on node 2 in the passive-participant setup (see Table 6. first column). As we can see the drift accumulates faster and reaches higher levels when the participants become active. Moreover, the drift variations are also higher compared with the passive-participant setup.

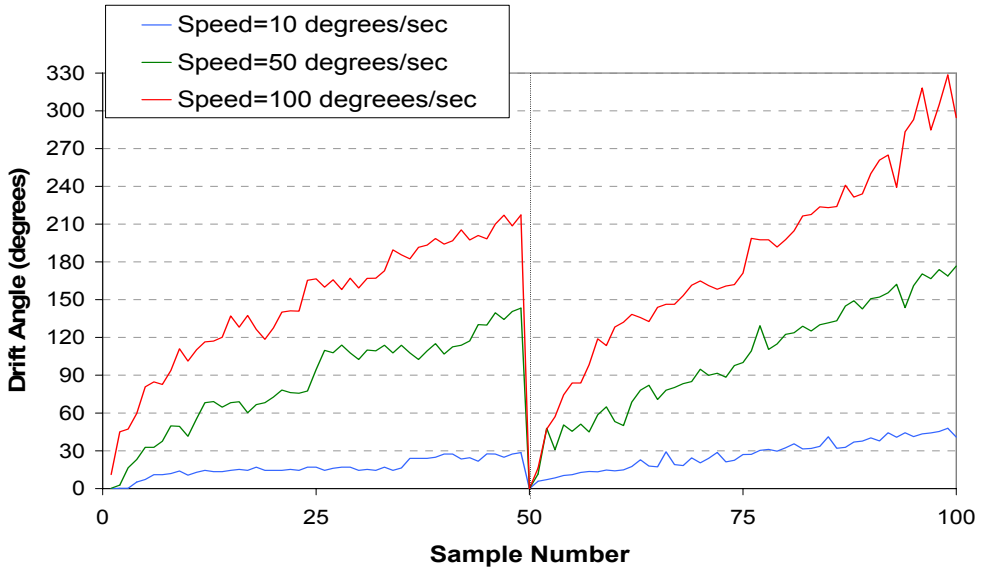


Figure 36 Drift behavior No Compensation: 6 active (left) vs. 6 passive participants (right)

Using the information from the CPO to update the orientation of the 3D object, the value of the drift is significantly reduced as illustrated in Figure 37.

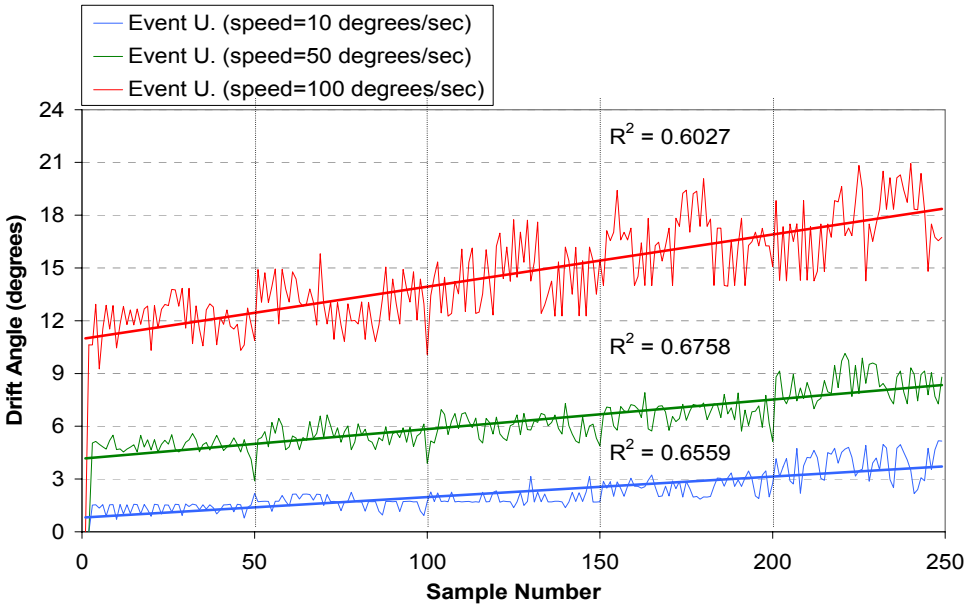


Figure 37. Event Update; 2, 3, 4, 5, and 6 active nodes

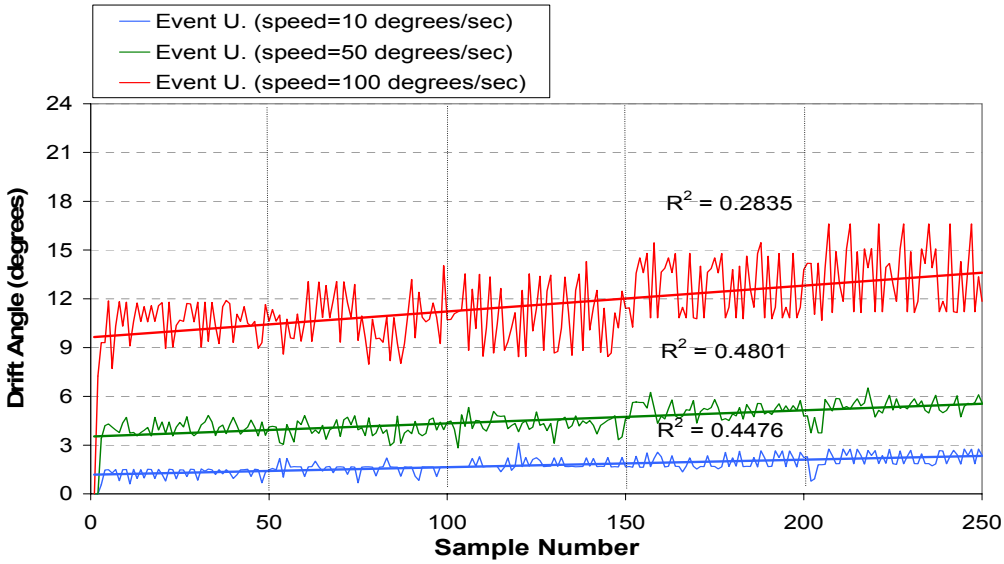


Figure 38. Event Update; 1 active + 1, 2, 3, 4, and 5 passive nodes

In comparison with the behavior of the algorithms on passive nodes setup (Figure 38), when the nodes become active (Figure 37) the amount of data exchanged in the network creates a certain amount of jitter. Since the Event Update method does not compensate for jitter, the drift variation increases.

Employing the ASA algorithm the drift is maintained at an almost constant average value as the number of active participants increases. The algorithm behavior, when the number of active participants increases, is similar to the behavior when the number of passive participants increases.

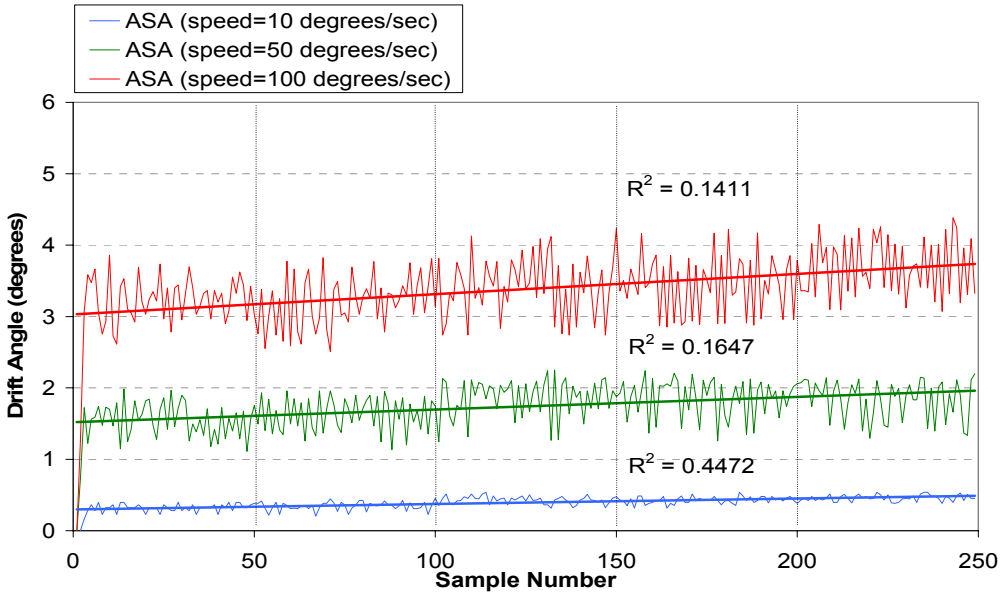


Figure 39. ASA with 2,3,4,5, and 6 active nodes

Figure 39 represents a plot of the drift values as the number of *active* participants increases from 2 to 6 while the ASA delay compensation algorithm is employed. For comparison, Figure 40 represents a plot of the drift values as the number of passive *participants* increases in the VE.

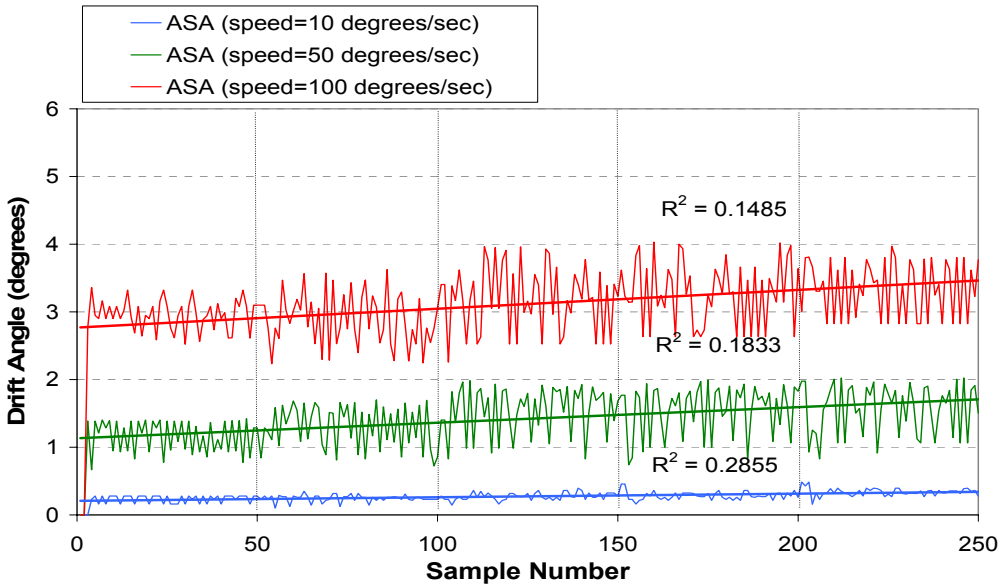


Figure 40 ASA with 1 active + 1,2,3,4 and 5 passive nodes

6.2.2 Scalability Regarding the Number of Active Participants

When the participants become active they produce and distribute data at the same time. In other words each active node is a server and other nodes interested (i.e. in the extreme case all) become its clients. In our experimental setup, the extreme case is made up of 6 active nodes that are interested in each other's data. Hence, each node is a server for the other nodes and a client

for them. Even if the experiment was deployed on a 100Mbps network, the amount of data exchanged in the VE creates some jitter. This is clearly visible in the drift behavior when no compensation is applied and when the Event Update method for compensation is used. Using the ASA algorithm, the effects of the jitter are compensated to a significant extent.

In the case of a two-participant setup, results show that the average drift, ψ_1, equals 3.2 degrees, while in the case of a six-participant setup the average drift, ψ_5, equals 3.7 degrees. In terms of scalability regarding active participants, using the metric from Section 6.1.3, in the six-participant setup, a low degree of scalability would translate to ψ_5 equaling $5*\psi_1$ or 16 degrees. However, the experimental results and trend lines in Figure 41 show that $\psi_5 \approx \psi_1$ when ASA is employed.

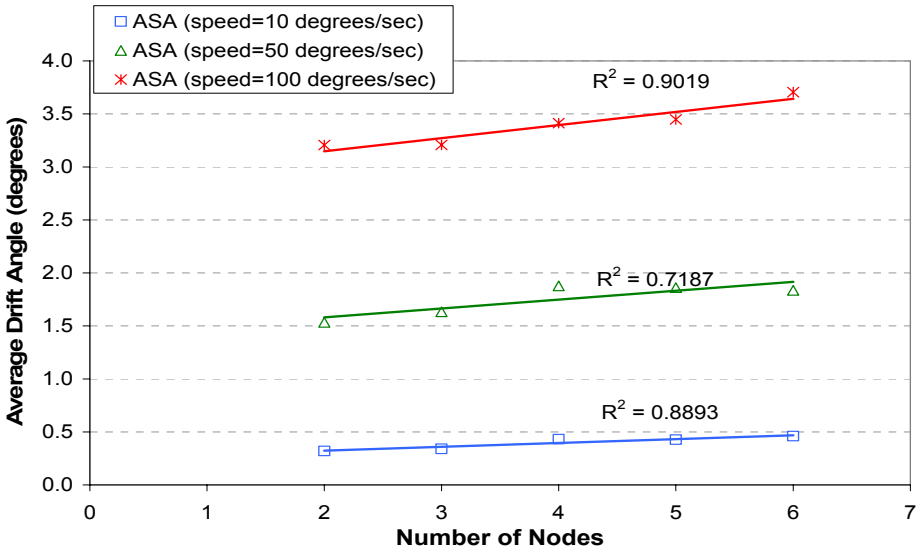


Figure 41. Orientation drift behavior as the number of active participants increases (ASA)

The slope of the regression line increases slightly with the action speed. At 100 degrees/sec the slope value is 0.123 and it decreases to 0.083 and further to 0.036 for action speeds of 50 and 10 degrees/sec, respectively. In the current scope of the experimental results, this shows that the ASA algorithm and the data distribution scheme are slightly sensitive to the active nodes, however, the sensitivity is less pronounced than in the Event Update method.

7. CONCLUSIONS AND OPEN PROBLEMS

7.1 Contributions and Implications of the Work

We have presented a survey of the current efforts in distributed systems architectures and in dynamic shared state maintenance for distributed interactive VEs. We have proposed a novel criterion for categorization of applications based on the participant interaction pattern with the virtual components of the shared scene. Such a categorization will help system designers to make the right choices in merging the domain requirements with the application and deployment infrastructure capabilities.

We have proposed a consistency maintenance algorithm, denoted ASA, that combines distributed monitoring with a distributed compensation method to maintain a consistent view of the shared scene in a distributed VE. The ASA addresses the impact of network latency on the shared state, in distributed interactive VE applications. By taking into account the measurement history of the end-to-end network delays among participants, the network jitter is taken into consideration. The decentralized computation approach for the drift values, carried out independently at each node, improves the system's scalability and its real-time behavior as compared to traditional delay compensation approaches.

Based on the observation that the trend of VEs is to employ an increasing number of sensors and that the sensor information must be delivered in near real-time to be useful, we have deployed the algorithm on a customized hybrid architecture.

7.2 Potential Applications - A Distributed AR Training Prototype

In an effort to improve airway management training and respiratory system prognostics we have developed the AR based simulator illustrated in Figure 42. Utilizing a human patient simulator (HPS) from Medical Education Technologies Inc. (METI) combined with 3D AR visualization of the airway anatomy and the (endotracheal tube) ETT, paramedics will be able to obtain a visual and tactile sense of proper endotracheal intubation (ETI) procedure.

(a) **(b)** **(c)**

Figure 42. Illustration of the AR tool for training paramedics on ETI

A trainee (a-concept, b-implementation) performing a medical procedure remotely supervised by the trainer(s) in (c).

The system consists of a HPS dressed with retroreflective material, an optical position tracking system, desktop computers, a lightweight HMD worn by the participants, and the intubation tools.

In medical training and simulation the learning potential of AR is significantly amplified by the capability of the system to present 3D medical models in real-time, at remote locations. An in-depth discussion of this prototype can be found in [107].

Such a system could allow paramedics, pre-hospital personnel and students to practice their skills without touching a real patient and will provide them with the visual feedback they could not obtain otherwise. Such a distributed AR training tool has the potential to:

- Allow an instructor to simultaneously train local and remotely located students.
- Allow students to actually "see" the internal anatomy and therefore better understand their actions on the HPS.

An extension of the above distributed prototype, that includes deformable 3D anatomical models [108] is under investigation.

7.3 Open Problems

The proposed algorithm is based on the assumption that the distributed system nodes are homogeneous. This assumption is particularly important when rendering complex graphics.

Since the system delays are directly dependent on the type of hardware involved at each site, in the case of a heterogeneous distributed system, additional measures must be taken.

The ASA is based on a distributed software monitoring scheme that is intrusive in its nature. We have taken additional steps in keeping the intrusiveness at low levels by avoiding I/O operations and keeping the measurements frequency at low.

Regarding the experiments, configurations of up to six participants were tested on the 100 Mbps network infrastructure. It is important to mention that even if the traffic increases significantly with each active node, the bandwidth limit has not been reached. For deployment on a wide area network, additional testing is required.

7.4 Research Horizons

The ASA can be combined with existing prediction (e.g. convergence) algorithms to improve the smoothness of the virtual object movement. Convergence algorithms are usually employed as a second phase of the dead-reckoning.

An immediate improvement would be the adaptation of the algorithm for heterogeneous distributed systems. We have presented a method to collect information from a node using the underlying operating system calls [109]. Such a system can be combined with the ASA to enhance the synchronization capabilities of a distributed interactive VE.

The ASA algorithm can be employed in new contexts. An interesting path would be the investigation of the ASA algorithm for haptic and audio cues in a distributed VE that includes haptic and 3D sound capabilities. Such an investigation could answer questions regarding how auditory and haptic cues enhance distributed collaboration and multimodal perception [110].

The combination of Distributed Systems with Virtual Environments poses many open questions, and consequently a great number of scientific challenges and opportunities.

APPENDIX A. QUATERNION BASICS, CORRECTION QUATERNION

Quaternions can be regarded as a 4-tuple consisting of a scalar component and a vector component denoted as $q = [w,(\vec{i}x,\vec{j}y,\vec{k}z)]$, where w is the scalar component of the quaternion and $(\vec{i}x,\vec{j}y,\vec{k}z)$ is the vector component.

A unit quaternion q specifies a 3D rotation (Figure 43) as an axis of rotation given by the vector component $(\vec{i}x,\vec{j}y,\vec{k}z)$ and an angle α about that axis indirectly determined by the scalar component as: $\alpha = 2\cos^{-1}(w)$

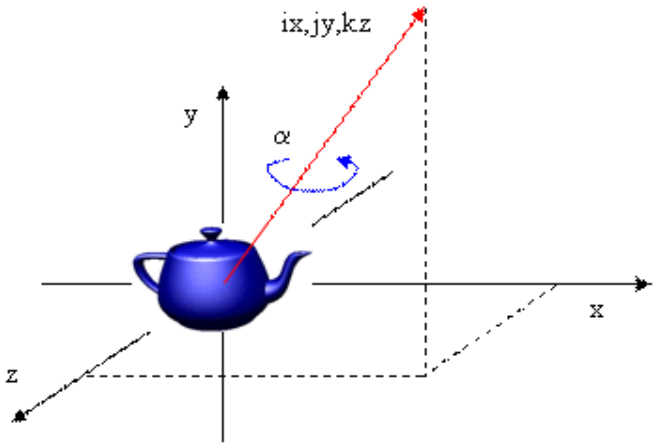


Figure 43. Rotation represented with quaternions

Inverse of a quaternion

If we designate the inverse of the quaternion q as q^{-1},

$$q^{-1} = [w,(\vec{i}x,\vec{j}y,\vec{k}z)]^{-1} = \frac{1}{|q|^2}[w,(-\vec{i}x,-\vec{j}y,-\vec{k}z)],$$

where $|q|$ is ***the norm*** of q sometimes called the length of q, and can be computed as

$$|q| = \sqrt{x^2 + y^2 + z^2 + w^2}$$

Quaternion correction

Quaternions were applied by NASA for satellite rotational maneuvers correction [111]. To express the error between the actual orientation of a 3D object and the desired orientation, the following equation can be used:

$$q_e = q_a^{-1} q_d$$

where q_e, q_a and q_d represent the error, actual, and desired rotation respectively.

$$q_e = [w_e, (i\, x_e, j\, y_e, k\, z_e)]$$

$$q_a = [w_a, (i\, x_a, j\, y_a, k\, z_a)]$$

$$q_d = [w_d, (i\, x_d, j\, y_d, k\, z_d)]$$

Furthermore knowing that $q_e = [\cos(\alpha_e/2), \sin(\alpha_e/2)(x_1, y_1, z_1)]$, the angular error can be inferred:

$$\alpha_e = 2\cos^{-1}(w_e) = 2\cos^{-1}[w_a w_d - (x_a x_d + y_a y_d + z_a z_d)]$$

APPENDIX B. APIS AND SDKS

OpenGL

Since its introduction in 1992, OpenGL® has become one of the most widely used, 2D/3D graphics API that allows development of graphical applications in hardware and software. The version used for this project was OpenGL® v.1.2 [112]. At the time of this writing OpenGL® v.1.5 is available. The relationship with other libraries and the VE application on a Linux based platform is described in Figure 44.

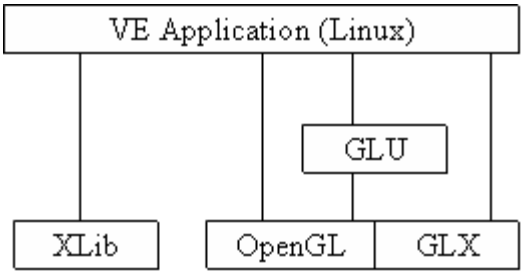


Figure 44. Relationship between OpenGL, GLU and Linux windowing APIs

In the development of the testbed application OpenGL was used indirectly through OpenGL Performer™ 2.5 toolkit.

Virtual Environment Software Sandbox

The Virtual Environment Software Sandbox (VESS) is a suite of libraries developed by the University of Central Florida's Institute for Simulation and Training [113]. VESS provides a high-level library allowing complex virtual entities (e.g. avatars), complete with geometry and

motion/articulation models, to be generated with a few simple lines of code. This is useful for dynamic networked VEs, which may involve many participants. VESS provides the developer with the ability to handle avatars at a high level and leave the details of movement, articulations, and behaviors to the system. VESS is also designed for easy portability. Its multi-layered architecture allows the developer to focus on the details of the application, without worrying about the specifics of the graphics API or hardware interfaces. Thus, applications built using the VESS libraries will be easily portable to any other platform. For this project VESS 2.0 was used. Currently, VESS 3.0 [114] is available.

OpenGL Performer

OpenGL Performer™ [115] is a powerful and comprehensive programming interface for developers creating real-time visual simulation and performance-oriented 3D graphics applications. The toolkit simplifies development of applications used for visual simulation, manufacturing, simulation-based design, virtual reality, scientific visualization, interactive entertainment, broadcast video, architectural walk-through, and computer aided design. For this work Open GL Performer™ 2.5 was employed. The latest major release, OpenGL Performer™ 3.1.1, is built atop the industry standard OpenGL® graphics library, interoperates with OpenGL Volumizer™, OpenGL Multipipe™ SDK, and OpenGL Vizserver™, includes both ANSI C and C++ bindings, and is available for the IRIX® operating system, Linux®, Windows® XP and Windows® 2000.

Distributed Augmented Reality Environment

DARE (Distributed Augmented/Artificial Reality Environment) is an object oriented library which uses AR paradigms to improve human-to-human interaction, enhancing the real scene that a person sees, with 3D computer generated objects. The initial targets of the library were AR applications. Later, the scope was expanded to the entire Virtuality Continuum spectrum. Currently a Beta version has been released for in-house use. The library is organized in several packages:

- *Networking***.** The Networking package provides a set of classes for fast development of scalable, real-time applications on a local area network. It facilitates development of real-time synchronization algorithms and load balancing schemes in a real-time distributed VE.
- *3D Deformation.* The Deformation package provides an interface to deform 3D models. This package is designed for anatomical models that deform based on user inputs and interaction in order to provide enhanced interactive simulations.
- *Calibration.* The Calibration package provides algorithms for determining the transformations necessary to accurately display virtual objects within a VE application. The algorithms include eye-point determination, camera calibration, and optical distortion calculation. The calibration algorithms can be adapted for use with any two-channel stereoscopic display.
- *Registration.* The Registration package provides algorithms for placing real and virtual objects into spatial coincidence (registration) and assessing the quality of registration. The

package aids in achieving superior registration by using robust optimization procedures for pose estimation.

- *Assessment.* The HMD prototype assessment battery includes visual perception tests aimed at assessing VE system parameters such as the resolution of the HMD. The human-in-the-loop test battery currently includes a modified Landolt C Visual Acuity test as well as a perceived contrast test (i.e. tests perceived luminance changes between a target and background). Both these tests assess the limits of the small spatial frequency channels of the human retina. Future tests will allow mapping of the contrast sensitivity function for the HMD display across all spatial frequency channels of the retina.

GIMP Tool Kit

GTK+ [116] is a multi-platform toolkit for creating graphical user interfaces. Offering a complete set of widgets, GTK+ is suitable for projects ranging from small one-off projects to complete application suites. It's called the GIMP toolkit because it was originally written for developing the GNU Image Manipulation Program (GIMP), but GTK has now been used in a large number of software projects, including the GNU Network Object Model Environment (GNOME) project. GTK is built on top of GDK (GIMP Drawing Kit) which is basically a wrapper around the low-level functions for accessing the underlying windowing functions (Xlib in the case of the X windows system), and gdk-pixbuf, a library for client-side image manipulation.

LIST OF REFERENCES

[1] D. C. Engelbart, "Augmenting Human Intelect: A Conceptual Framework," Stanford Research Institute SRI Project No. 3578, 1962.

[2] P. Milgram and F. Kishino, "A Taxonomy of Mixed Reality Visual Displays," *IECE Transactions on Information and Systems*, vol. E77-D, No.12, pp. 1321-1329, 1994.

[3] F. Biocca, A. Tang, and D. Lamas, "Evolution of the Mobile Infosphere: Iterative design of a high information-bandwidth, mobile augmented reality interface," presented at International Conference on Augmented, Virtual Environments and Three-Dimensional Imaging, ICAV3D'2001, Mykonos, Greece, 2001.

[4] R. E. Eberts and C. G. Eberts, "Four Approaches to Human Computer Interaction," in *Intelligent interfaces: theory, research, and design*, P. A. Hancock and M. H. Chignell, Eds.: North-Holland, 69-127, 1989.

[5] C. Swindells, J. C. Dill, and K. S. Booth, "System Lag Tests for Augmented and Virtual Environments," presented at 13st ACM Symposium on User Interface Software and Technology, San Diego, CA, 2000.

[6] MCI, "Latency statistics," vol. 2004: MCI, 2004.

[7] R. Held and N. Durlach, "Telepresence, time delay and adoption.," in *Pictorial communication in virtual and real environments*, S. Ellis, K. Kaiser, and A. C. Grunwald, Eds. London: Taylor & Francis, 232-245, 1991.

[8] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. J. Brooks, "Effect of Latency on Presence in Stressful Virtual Environments," presented at IEEE Virtual Reality, Los Angeles, CA, 2003.

[9] M. Lombard and T. Ditton, "At the Heart of it all: The concept of presence.," *Journal of Computer-Mediated Communication*, vol. 3, 1997.

[10] P. Verissimo and L. Rodrigues, *Distributed Systems for System Architects*: Kluwer Academic, 2001.

[11] G. Coulouris and J. Dollimore, *Distributed Systems Concepts and Design*, 3 ed: Addison Wesley, 2001.

[12] V. Hadzilacos and S. Toueg, "A Modular Approach to Fault-tolerant Broadcasts and Related Problems," 1994.

[13] A. S. Tanenbaum and v. M. Steen, *Distributed Systems, Principles and Paradigms*. New Jersey: Prentice Hall, 2002.

[14] G. Cabri and L. Leonardi, "Mobile-Agent Coordination Models for Internet Applications," *IEEE Computer*, vol. 33, pp. 82-89, 2000.

[15] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, pp. 80-112, 1985.

[16] N. Carriero and D. Gelernter, "A Computational Model of Everything," *Communications of the ACM*, vol. 44, pp. 77-81, 2001.

[17] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford, "TSpaces," *IBM Systems Journal*, vol. 37, pp. 454-474, 1998.

[18] Sun Microsystems, "JavaSpaces Specification," Sun Microsystems 1998.

[19] A. Kariv, "GigaSpaces Cluster White Paper," G. T. Ltd., Ed., 2002.

[20] P. Hutto and M. Ahamad, "Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories," presented at 10th Int'l Conference on Distributed Computing Systems, 1990.

[21] D. Terry, K. Petersen, M. Spreitzer, and M. Theimer, "The Case for Non-transparent Replication: Examples from Bayou.," *IEEE Data Engineering*, vol. 21, pp. 12-20, 1998.

[22] A. Goscinski and W. Zhou, "The Client-Server Model and Systems," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, vol. 3. New York: John Wiley & Sons, 431-451, 1999.

[23] R. L. Hyde and B. D. Fleisch, "A Case for Virtual Distributed Objects," *International Journal on Parallel and Distributed Computing*, vol. 1, 1998.

[24] R. Meier and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems," presented at 22nd International Conference on Distributed Computing Systems Workshops, Viena, Austria, 2002.

[25] A. Carzaniga, E. Di Nitto, D. S. Rosenblum, and A. L. Wolf, "Issues in Supporting Event-Based Architectural Styles," presented at 3 rd International Software Architecture Workshop, Orlando, Florida, 1998.

[26] D. Marinescu, L. Boloni, K. K. Jun, K. Palacz, and R. Sion, "The Bond Agent System and Applications," in *Agent Systems, Mobile Agents, and Applications*, vol. 1882, D. Kotz and F. Mattern, Eds.: Springer Verlag, 99-112, 2000.

[27] I. S. O. ISO, "Open systems interconnection (OSI)," 1987.

[28] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable content-addressable network," presented at ACM SIGCOMM 2001 Technical Conference, San Diego, CA, 2001.

[29] B. Yang and H. Garcia-Molina, "Comparing Hybrid Peer-to-Peer Systems," presented at Proceedings of the 27th International Conference on Very Large Data Bases, San Francisco, CA, 2001.

[30] K. Kant, R. Iyer, and V. Tewari, "On the Potential of Peer-to-Peer Computing: Classification and Evaluation," presented at CCGrid, Berlin, Germany, 2002.

[31] K. Kant, "An Analytic Model for Peer to Peer File Sharing Networks," presented at International Conference on Communications, Anchorage, Alaska, 2003.

[32] B. Leuf, *Peer to Peer Collaboration and Sharing over the Internet*: Addison Wesley, 2002.

[33] J. Lan, X. Liu, P. Shenoy, and K. Ramamritham, "Consistency maintenance in peer-to-peer file sharing networks," presented at Workshop on Internet Applications. WIAPP 2003, San Jose, CA, 2003.

[34] Dictionary, *The American Heritage® Dictionary of the English Language*, fourth ed, 2000.

[35] A. Lippman, "Movie-maps: An application of the optical videodisc to computer graphics," *ACM SIGGRAPH Computer Graphics*, vol. 14, pp. 32-42, 1980.

[36] S. Fisher, "Viewpoint Dependent Imaging: An Interactive Stereoscopic Display," presented at SPIE, 1982.

[37] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE," presented at SIGGRAPH, 1993.

[38] M. Czernuszenko, D. Pape, D. J. Sandin, T. DeFanti, G. Dawe, and M. Brown, "The ImmersaDesk and Infinity Wall Projection-Based Virtual Reality Displays," *Computer Graphics*, vol. 31, pp. 46-49, 1997.

[39] A. R. Smith and J. F. Blinn, "Blue screen matting," presented at International Conference on Computer Graphics and Interactive Techniques SIGGRAPH'96, New Orleans, LA, 1996.

[40] J. Rolland and H. Fuchs, "Optical Versus Video See-Through Head-Mounted Displays in Medical Visualization," *Presence: Teleoperators and Virtual Environments*, vol. 9, pp. 287-309, 2000.

[41] E. Wegman, "Affordable Environments for 3D Collaborative Data Visualization," *IEEE Computing in Science & Engineering*, vol. 2, pp. 68-72, 2000.

[42] F. J. Ferrin, "Survey of helmet tracking technologies," *SPIE - The International Society for Optical Engineering - Large-Screen Projection, Avionic, and Helmet-Mounted Displays*, vol. 1456, pp. 86-94, 1991.

[43] K. Meyer, H. L. Applewhite, and F. Biocca, "A survey of position trackers," *Presence: Teleoperators and Virtual Environments*, vol. 1, pp. 173-200, 1992.

[44] G. Burdea and P. Coiffet, *Virtual Reality Technology*: Wiley Interscience, 1994.

[45] J. Rolland, L. Davis, and Y. Baillot, "A Survey of Tracking Technology for Virtual Environments," in *Augmented Reality and Wearable Computers*, T. C. W. Barfield, Ed.: Lawrence Erlbaum Press, 2000.

[46] J.-F. Wang, V. Chi, and H. Fuchs, "A Real-Time Optical 3D Tracker for Head-Mounted Display Systems," presented at Symposium on Interactive 3D Graphics, Snowbird, UTAH, 1990.

[47] E. Foxlin and N. Durlach, "An Inertial Head-Orientation Tracker with Automatic Drift Compensation for use with HMDs," presented at VRST - Virtual Reality Software and Technology, 1994.

[48] J. A. Farrell and M. Barth, *The Global Positioning System & Inertial Navigation*: McGrawHd99 New York, 1998.

[49] A. State, G. Hirota, D. T. Chen, W. F. Garrett, and M. A. Livingston, "Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking," presented at SIGGRAPH, 1996.

[50] E. Foxlin, M. Harrington, and G. Pfeipffer, "Constellation: A Wide-Range Wireless Motion-Tracking System for Augmented Reality an Virtual Set Applications," presented at SIGGRAPH, 1998.

[51] U. Neumann, S. You, Y. Cho, J. Lee, and J. Park, "Augmented Reality Tracking in Natural Environments," presented at International Symposium on Mixed Reality, 1999.

[52] D. Miller and J. A. Thorpe, "SIMNET: The advent of simulator networking," *Proceedings of IEEE*, vol. 83, pp. 1114-1123, 1995.

[53] M. J. Zyda, D. R. Pratt, J. G. Monahan, and K. P. Wilson, "NPSNET: Constructing a 3D Virtual World," presented at ACM Symp. on Interactive 3D Graphics, 1992.

[54] M. Zyda and S. Singhal, "The origin of Networked Virtual Environments," in *Networked Virtual Environments: Design and Implementation*, S. Spencer, Ed. New York: Addison Wesley, 19-53, 1999.

[55] GameSpy, "Massively Multiplayer Online Games The Past, The Present, and The Future," in *GameSpy*, 2003.

[56] M. Capps, D. McGregor, D. Brutzman, and M. J. Zyda, "NPSNET-V: A New Begining for Dynamically Extensible Virtual Environments," *IEEE Computer Graphics & Applications*, vol. 20, pp. 12-15, 2000.

[57] C. M. Greenhalgh, J. Purbrick, and D. Snowdon, "Inside MASSIVE3: Flexible Support for Data Consistency and World Structuring," presented at Collaborative Virtual Environments 2000, San Francisco, CA, 2000.

[58] D. J. Roberts and P. M. Sharkey, "Maximizing Concurrency and Scalability in a Consistent, Causal, Distributed Virtual Reality System, whilst Minimizing the Effect of Network Delays," presented at 6th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE), Cambridge, Massachussetts, 1997.

[59] E. Frecon and M. Stenius, "DIVE: A scaleable network architecture for distributed virtual environments," *Distributed Systems Engineering Journal*, vol. 5, pp. 91-100, 1998.

[60] S. Pettifer, J. Cook, J. Marsh, and A. West, "DEVA3: Architecture for a Large-Scale Distributed Virtual Reality System," presented at ACM Virtual Reality Software and Technology, Seoul, Korea, 2000.

[61] J. W. Barrus, R. C. Waters, and D. B. Anderson, "Locales: Supporting Large Multiuser Virtual Environments," *IEEE Computer Graphics and Applications*, vol. 16, pp. 50-57, 1996.

[62] M. Fairen and A. Vinacua, "ATLAS, A Platform for Distributed Graphics Applications.," presented at Proc. VI Eurographics Workshop on Programming Paradigms in Graphics, 1997.

[63] D. Lee, M. Lim, and S. Han, "ATLAS - A Scalable Network Framework for Distributed Virtual Environments," presented at 4th International Conference on Collaborative Virtual Environments, Bonn, Germany, 2002.

[64] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation.," presented at ACM SIGCOMM'95, 1995.

[65] G. Singh, L. Serra, W. Png, A. Wong, and H. Ng, "BrickNet: Sharing Object Behaviors on the Net.," presented at Virtual Reality Annual International Symposium (VRAIS'95), Research Triangle Park, North Carolina, 1995.

[66] B. MacIntyre and S. Feiner, "A Distributed 3D Graphics Library," presented at ACM SIGGRAPH, 1998.

[67] H. Tramberend, "Avocado: A Distributed Virtual Reality Framework," presented at IEEE Virtual Reality, Houston, TX, 1999.

[68] D. Schmalstieg and G. Hesina, "Distributed Applications for Collaborative Augmented Reality," presented at IEEE Virtual Reality 2002, Orlando, Florida, 2002.

[69] M. Zyda and S. Singhal, *Networked Virtual Environments, Design and Implementation*. New York: Addison Wesley, 1999.

[70] S. Singhal and M. Zyda, "Exploiting position history for efficient remote rendering in networked virtual reality," *Presence: Teleoperators and Virtual Environments*, vol. 4, 1995.

[71] B. Callaghan, B. Pawlowski, and P. Staubach, "NFS version 3 protocol specification. Request for Comments (RFC) 1813," Information Sciences Institute, Marina del Rey, CA 1995.

[72] R. Campbell, "Managing AFS: The Andrew File System," Saddle River, NJ, 1998.

[73] N. Nakamura, K. Nemoto, and K. Shinohara, "Distributed virtual reality system for cooperative work.," NEC Research and Developement 35(4), 1994.

[74] V. Anupam and C. Bajaj, "Distributed and collaborative visualization," *IEEE Multimedia*, vol. 1, pp. 39-49, 1994.

[75] C. Carlsson and O. Hagsand, "DIVE- A platform for multi-user virtual environments.," *Computers and Graphics*, vol. 17, pp. 663-669, 1993.

[76] K. Birman and K. Marzullo, "The ISIS distributed programming toolkit and the META distributed operating system: A brief overview.," in *Mission Critical Operating Systems*: Amsterdam: IOS Press, 1992.

[77] G. Singh, L. Serra, W. Prg, and H. Ng, "BrickNet: A software toolkit for network-based virtual worlds.," *Presence: Teleoperators and Virtual Environments*, vol. 3, pp. 19-34, 1994.

[78] G. Hesina, D. Schmalstieg, A. Fuhrmann, and W. Purgathofer, "Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics," presented at ACM Virtual Reality Software and Technology, London, 1999.

[79] E. Berglund and D. Cheriton, "Amaze: A multiplayer computer game.," *IEEE Software*, vol. 2, pp. 30-39, 1985.

[80] J. A. Thrope, "The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting," presented at 9th Interservice/Industry Training System Conference, 1987.

[81] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and Z. Steven, "NPSNET: A Network Software Architecture For Large Scale Virtual Environments," *Presence: Teleoperators and Virtual Environments*, vol. 3, pp. 265-287, 1994.

[82] A. Katz and K. Graham, "Dead reckoning for airplanes in coordinated flight.," presented at Tenth Workshop on Standards for the Interoperability of Defense Simulations, Orlando, FL, 1994.

[83] D. R. Pratt, "A software architecture for the construction and management of real-time virtual worlds.," in *Computer Science*. Monterey, CA: Naval Postgraduate School, 1993.

[84] M. Friedman, T. Starner, and A. Pentland, "Device Synchronization using an optimal linear filter.," presented at Symposium on Interactive 3D graphics, Cambridge, MA, 1992.

[85] J. Escobar, C. Partridge, and D. Deuch, "Flow synchronization protocol.," *ACM/IEEE Transactions on Networking*, vol. 2, pp. 111-121, 1994.

[86] M. Zyda and S. Singhal, "Resource Management for Scalability and Performance," in *Networked Virtual Environments: Design and Implementation*, S. Spencer, Ed. New York: Addison Wesley, 181-249, 1999.

[87] S. K. Singhal and D. R. Cheriton, "Using projection aggregations to support scalability in distributed simulation.," presented at 16th International Conference on Distributed Computing Systems (ICDCS), Hong Kong, 1996.

[88] H. Abrams, K. Watsen, and M. J. Zyda, "Three tiered interest management for large-scale virtual environments.," presented at Virtual Reality Systems and Technology, Taipei, Taiwan, 1998.

[89] P. M. Sharkey, M. D. Ryan, and D. J. Roberts, "A local perception filter for distributed virtual environments.," presented at Virtual Reality Annual International Symposium (VRAIS), Atlanta, GA, 1998.

[90] S. R. Ellis and B. M. Menges, "Studies of the localization of virtual objects in the near visual field," in *Wearable Computers and Augmented Reality*, T. Caudell and W. Barfield, Eds.: Lawrence Erlbaum Associates, 263-293, 2001.

[91] S. Zhou, W. Cai, F. Lee, and S. Turner, "Consistency in Distributed Interactive Applications," presented at European Simulation Interoperability Workshop, England, UK, 2001.

[92] X. Qin, "Delayed Consistency Model for Distributed Interactive Systems with Real-time Continuous Media," Griffith University, Brisbane, Australia 2003.

[93] J. P. Rolland, F. Biocca, H. Hua, Y. Ha, C. Gao, and O. Harrisson, "Teleportal Augmented Reality System: Integrating virtual objects, remote collaborators, and physical reality for distributed networked manufacturing.," in *Virtual and Augmented Reality Applications in Manufacturing*, S. K. Ong and A. Y. Nee, Eds.: Springer-Verlag London Ltd, 400, 2004.

[94] D. J. Sturman and D. Zeltzer, "A survey of glove-based input," *IEEE Computer Graphics and Applications*, vol. 14, pp. 30-39, 1994.

[95] i. Essential Reality, "P5 Manual," 2002.

[96] A. J. Ballardie, P. F. Francis, and J. Crowcroft, "Core Based Trees," presented at ACM SIGCOMM, 1993.

[97] "NS-2 Network Simulator," K. Fall and K. Varadhan, Eds., 2004.

[98] "PARSEC - Parallel Simulation Environment for Complex Systems," 2004.

[99] "SSF - Scalable Simulation Framework," 2004.

[100] J. Rolland, F. Biocca, F. G. Hamza-Lup, and R. Martins, "Development of Head-Mounted Projection Displays for Distributed, Collaborative, Augmented Reality. Applications," *MIT Presence (in press)*, 2005.

[101] F. Biocca and J. Rolland, "Teleportal Face-to-Face system: Teleconferencing and tele-work augmented reality system." USA, 2000.

[102] F. G. Hamza-Lup, L. Davis, and J. Rolland, "The ARC Display: An Augmented Reality Visualization Center," presented at International Symposium on Mixed and Augmented Reality, Darmstadt, Germany, 2002.

[103] L. Davis, J. P. Rolland, F. G. Hamza-Lup, Y. Ha, J. Norfleet, B. Pettitt, and C. Imielinska, "Enabling a Continuum of Virtual Environment Experiences," *IEEE Computer Graphics & Applications*, vol. 23, pp. 10-12, 2003.

[104] N. D. I. Polaris, "Northern Digital Polaris tracking system," 2004.

[105] L. Davis, F. G. Hamza-Lup, and J. P. Rolland, "A Method for Designing Marker-Based Tracking Probes," presented at International Symposium on Mixed and Augmented Reality, Arlington, VA., 2004.

[106] D. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Trans. Communications*, vol. 39, pp. 1482-1493, 1991.

[107] J. P. Rolland, L. Davis, and F. G. Hamza-Lup, "Development of a training tool for endotracheal intubation: Distributed Augmented Reality," presented at Medicine Meets Virtual Reality (MMVR), 2003.

[108] A. Santhanam, C. Fidopiastis, F. G. Hamza-Lup, and J. P. Rolland, "Physically-based Deformation of High-Resolution 3D Models for Augmented Reality based Medical Visualization," presented at 7th International Conference on Medical Image Computing and Computer Assisted Intervention, Rennes Saint-Malo, France, 2004.

[109] F. Hamza-Lup, "A Less Intrusive System Monitoring Scheme," University of Central Florida, School of Computer Science, Orlando, Technical Report CS-TR-04-08, 2004.

[110] M. P. Hollier, A. N. Rimell, D. S. Hands, and R. M. Voelcker, "Multi-modal perception.," *BT Technological Journal*, vol. 17, pp. 35-46, 1999.

[111] J. B. Kuipers, *Quaternions and Rotation Sequences. A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Princeton: Princeton Univ. Press, 1998.

[112] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Reading, MA: Addison Wesley, 1999.

[113] J. Daly, B. Kline, and G. Martin, "VESS: Coordinating Graphics, Audio, and User Interaction in Virtual Reality Applications," presented at IEEE Virtual Reality, Orlando, FL, 2002.

[114] J. Daly, B. Kline, D. Cope, and G. Martin, "Virtual Environment Software Sandbox - User's Guide," Institute for Simulation and Training, Orlando IST-TR-03-06, 2003.

[115] SGI, "OpenGL Performer™ Programmer's Guide," SGI, Ed., 2004.

[116] T. Gale, "GTK+ 2.0 Tutorial," 2002.