Network Services for Multi-User Virtual Environments

Thomas A. Funkhouser
AT&T Bell Laboratories
Murray Hill, NJ

Abstract

This paper describes network services to support large multi-user virtual environments. A client-server design is proposed in which multiple servers coordinate execution, manage communication, offload processing, and provide persistent storage for their clients. Using this design, it is possible to support real-time features, such as collision detection, voice bridging, persistent updates, physical simulation, and autonomous agents, that would be difficult to implement for large virtual environments with a peer-to-peer design. The paper includes a description of services being implemented in RING, a client-server system for interaction between many users in large virtual environments.

1 Introduction

Multi-user virtual environment applications incorporate computer graphics, sound simulation, and networks to simulate the experience of real-time interaction between multiple users in a shared three-dimensional virtual world. Each user runs an interactive interface program on a "client" computer connected to a wide-area network. The interface program simulates the experience of immersion in a virtual environment by rendering images/sounds/etc. of the environment as perceived from the user's simulated viewpoint. Each user is represented in the shared virtual environment by an entity rendered on every other user's computer (see Figure 1). Multi-user interaction is supported by matching user actions to entity updates (e.g., motion/sound generation) in the shared virtual environment.

Applications for multi-user virtual environment technology include distributed training, simulation, education, home shopping, virtual meetings, and multiplayer games. For example, consider a virtual city in which multiple users interact in real-time. As each user moves through the city, a graphical representation of that user is displayed moving on each other user's screen. When any user talks into a microphone, his/her voice is played with appropriate stereo control so as to appear to come from the position of the entity representing the speaker. Social interactions and commerce might be more compelling with a 3D virtual city interface incorporating both graphics and sound than with textual or 2D multimedia interfaces such as those employed by current chat programs and browsers for the world wide web.

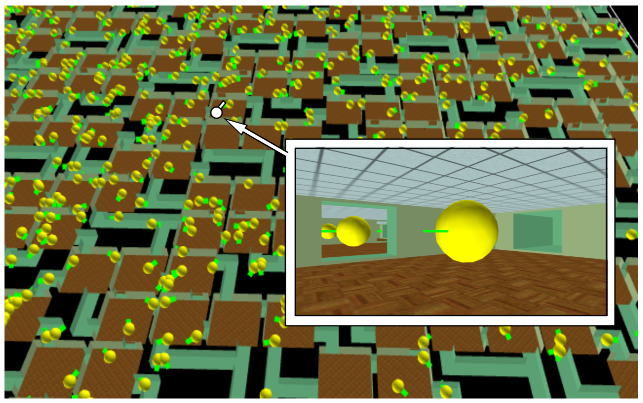
Figure 1: Multiple users (represented by spheres) interact in a virtual environment.

There are several examples of multi-user virtual environments available in research and commercial systems today. First, various research systems allow simple multi-user interactions in 3D virtual environments [2, 6]. Second, commercial multi-player games allow a small number of users to play in a shared 3D gaming experience [9]. Third, MUDs and chat groups allow hundreds of people to interact via text over networks in real-time [8]. Finally, military simulators allow up to a few hundred soldiers to train simultaneously on a shared virtual battlefield [4, 13]. Although there are many examples of interactive, real-time applications with a shared experience between multiple users, none of them provides the important combination of real-time graphics, real-time voice, and persistent data; and none scales to a very large number of users.

Interactive virtual environments with many simultaneous users distributed over a wide-area network provide an interesting research challenge due to their unique performance and distribution characteristics. The system must provide real-time, synchronized, and persistent access to data describing the virtual environment, even though the data is logically distributed across many clients and can be updated very frequently.

Historically, most multi-user virtual environment systems have been built using a peer-to-peer communication design. Clients store replicate copies of the virtual environment and maintain consistent state by passing messages via point-to-point [2, 15], multicast [5, 13], and/or broadcast networks [3, 4] (see Figure 2). The difficulty with peer-to-peer system designs is that they don't scale well. Generally, every client must store a representation for every entity, every client must process messages for every entity, and every client must simulate behavior for every entity in the virtual environment. Additionally, fully distributed algorithms must be used to synchronize access to shared data.

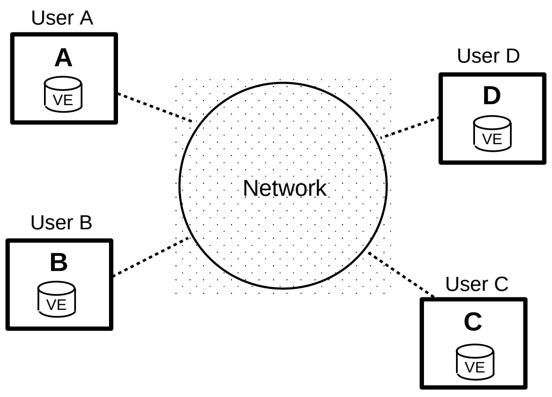

Figure 2: A multi-user virtual environment system.

The focus of this paper is to investigate services that can be provided to support multi-user virtual environments via a client-server design [10, 12, 16]. The idea is to include multiple server computers that perform processing and manage communication for their clients. In this case, the "dumb" network shown in Figure 2 is replaced by an "intelligent" network which provides coordination, computation, and storage services for the system (see Figure 3). We aim to overcome the difficulties of peer-to-peer system designs by implementing several components of the system in network-resident servers [1] which provide coordinating infrastructure and off-load processing, networking, and storage burden from the client computers. Our goal is to develop systems that are more realistic, more scalable, more persistent, more interactive, and/or more affordable than is possible with a peer-to-peer system design.

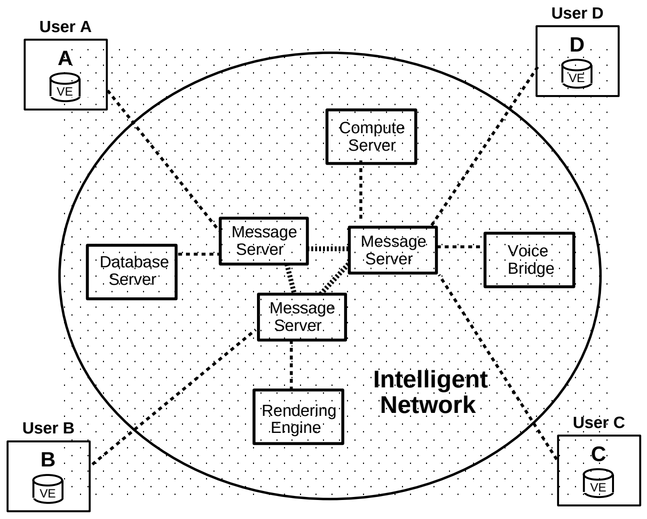

Figure 3: Servers manage communication and processing for a multi-user virtual environment.

The next section motivates the need for network services. Sections 3-6 describe specific services that may be supported in the network for multi-user virtual environment systems. Finally, a brief discussion and conclusion appears in Section 7.

[1]A service is considered to reside logically "in-the-network" if it does not execute on any client. Of course, it may actually execute on any computer connected to the network.

2 Network Services

The primary motivation for providing services in the network is that many functions of a multi-user virtual environment system are not logically controlled by any of its clients. For instance, which client should manage updates to the shared parts of the virtual environment (e.g. the roads of the virtual city)? Which client should control autonomous agents (e.g., tour guides)? How are updates to the shared environment stored persistently (e.g., when all users have turned off their computers)? Servers also provide a convenient location for coordination and synchronization of distributed algorithms (e.g., consider algorithms for generating unique entity IDs). Although it is possible to implement shared functions in a purely peer-to-peer system via complex election and agreement protocols, it is simpler and more efficient to manage and synchronize shared resources using a few server computers connected by fast networks.

Another motivation is that supporting the illusion of immersion and real-time interaction among many users in a virtual environment requires a lot of processing, networking, and storage capacity. The system must generate realistic-looking images on the screen of each client computer at frame rates suitable for interactive user control (i.e., more than ten frames per second). It must bridge streams of voice input from all users into a stream of voice output to be played on each client computer. Also, it should support collision detection, persistent updates, physical simulation, autonomous agents, and several other features requiring large amounts of computing resources. In conflict with these computational demands, practical economics dictate that affordable, large scale, multi-user systems must be built primarily using low-cost client computers. Although the capabilities of desktop computers is growing at an astounding rate, it will be quite a while before computers supporting all these features are widely available and inexpensive. Certainly, today's high-end PCs (with pentium processors, 8MB of memory, and 28.8 Kb/s modems) are insufficient for simulation of large multi-user virtual environments by themselves.

Server computers can assist the clients by executing some operations remotely. We aim to migrate operations whose computing requirements exceed what is generally available in clients, and whose latency requirements allow remote execution, onto server computers with faster processors, faster network connections, larger memories, and possibly special hardware. Since servers are shared between many clients, large scale systems can be constructed affordably using inexpensive clients (PCs) with low bandwidth network connections (modems), while more expensive high performance workstations (SGIs) and high bandwidth networks (ATM) are required only for a relatively few servers and their interconnections.

The challenge of building a client-server system is to identify which operations should be implemented in clients, and which should be implemented in servers. In general, the choice is dictated by function, expense, and performance. Operations requiring hardware too expensive to include in each client must be performed in servers (e.g., voice bridging). Other operations may be implemented in either clients or servers depending on their processing, memory, messaging, and latency requirements. In particular, operations that are extremely sensitive to latency (e.g., rendering for a head-mounted display) are not good candidates for execution in remote servers.

For the remainder of this paper, we describe and discuss services that are being included in servers for a multi-user virtual environment system called RING (**R**eal-time **I**nteractive **N**etworked **G**raphics) [10]. Although a mulitude of network services are possible, this paper focuses on the following ones: 1) message distribution, 2) interaction detection, 3) voice bridging, and 4) database services.

3 Message Distribution

In order to support very large numbers of users (> 1000) interacting simultaneously in a multi-user virtual environment it is necessary to develop a system design and communication protocol that does not require sending update messages to all participating clients for every entity state change. To accomplish this goal, we can off-load message distribution responsibilities from clients into *message servers.* If we include intelligent message servers in the network, clients do not need to send messages directly to other clients, but instead can send them to message servers which forward them to other clients and servers participating in the same multi-user virtual environment (see Figure 4).

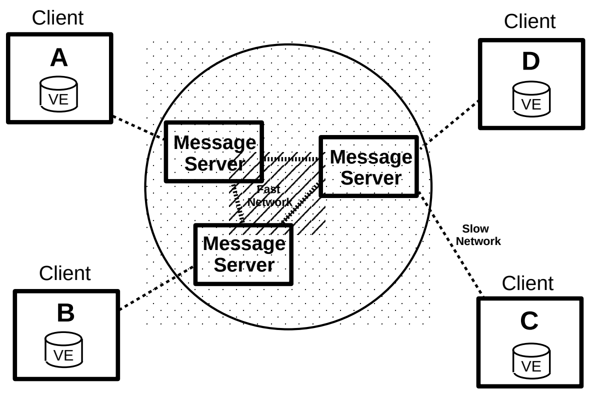

Figure 4: Message servers manage communication for their clients.

A key feature of this system design is that message servers can process updates before propagating them to clients, culling, augmenting, or altering them, and therefore off-load some message processing burden from their clients. For instance, a message server may determine that a particular update is relevant only to a small subset of clients and then propagate the message only to those clients or their message servers. In addition, a message server may send clients auxiliary messages that contain status information helpful for future client processing. Finally, a message server may replace some set of messages intended for a client with another (possibly simpler) set of messages better suited to the client's performance capabilities.

An important advantage of message servers is that they can dramatically reduce the message processing and network bandwidth burden of their clients. For instance, consider distribution of messages for updates to the positions of entities moving through the environment shown in Figure 1. If we aim to support only visual interactions, then each client must receive updates only for the subset of entities that are visible to its own entities (see Figure 5).

In RING, message servers use object-space visibility algorithms to compute the region of influence for each update and propagate the update only to the small subset of clients to which the update is relevant. Message filtering is implemented using precomputed line-of-sight visibility information stored in the message server for each region of the environment. As entities move through the environment, message servers keep track of which regions contain which entities by exchanging "periodic" update messages when entities cross region boundaries. Real-time update messages are propagated only to servers and clients containing entities inside some region visible to the one containing the updated entity.

With servers filtering messages based on entity visibility, each client must process only 2% of update messages in the virtual environment shown in Figure 1. For densely occluded environments, client message rates scale with the density of entities rather than the number of entities in the environment, and thus the RING system design scales to very large environments with very many simultaneous users while client message processing rates remain constant. It would be difficult to duplicate these results in a peer-to-peer system since the storage for precomputed visibility regions and the network bandwidth for periodic messages would exceed the capabilities of most client computers. See [10] for more details.

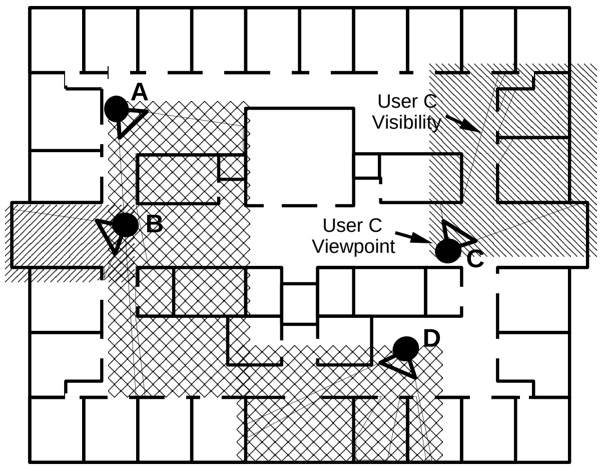

Figure 5: Update messages must only be propagated to clients managing entities that can perceive the updates.

Another advantage of message servers is that distribution of messages for high-level management of the virtual environment can be performed by servers without the involvement of every client. For instance, adding or removing an entity to or from the virtual environment requires notification of only one message server. That message server handles notification of other appropriate servers and clients. Since the message servers maintain the routing tables necessary for distribution of messages, clients are more modular, and may be added or removed from the system seemlessly. Moreover, the client-server design allows use of efficient networks and protocols available between server workstations, but not universally available to all clients. For instance, clients may connect to servers via low-bandwidth, connection-oriented networks (modems), while servers communicate with each other via high-bandwidth, packet networks (ATM).

The disadvantage of message servers is that latency is introduced as messages are routed through one or more server. In our experience, the extra latency is not significant. In extreme situations, long latencies can be mitigated with predictive behavioral models simulated in clients.

4 Interaction Detection

Detecting interactions (e.g., sight, sound, collisions, etc.) between entities in a multi-user virtual environment often requires complex geometric computations, and thus is a good candidate for assistance from server machines. Naive algorithms for detecting interactions between n entities execute in $O(n^2)$ and require up-to-date locations for all n entities. More efficient algorithms use spatial data structures [7, 14] and/or precomputed regional interactions [1, 17] to search for potential interactions. In any case, typical interaction detection algorithms for large virtual environments require more processing power, storage capacity, and/or network bandwidth than is available on affordable client computers.

Unfortunately, real-time interactions (e.g., collisions) require low latency response, and thus it is not practical to execute real-time interaction detection algorithms entirely in compute servers. However, servers can still assist clients with interaction detection by performing latency tolerant parts of the computation, while latency sensitive parts are executed locally in each client. In virtual environments with many entities, a large portion of an interaction detection computation is devoted to determining which entities can possibly interact. The $O(n^2)$ space of potential interactions is pruned via conservative approximation techniques (e.g., bounding boxes). Then, exact interactions are computed for the remaining subset using more precise algorithms. Based on this model of computation, we include *interaction detection servers* in the network that monitor updates to entities and conservatively predict interactions, notifying clients of potential interactions via messages (see Figure 6). With this server assistance, each client must compute exact interactions only for a small subset of the entities in the virtual environment.

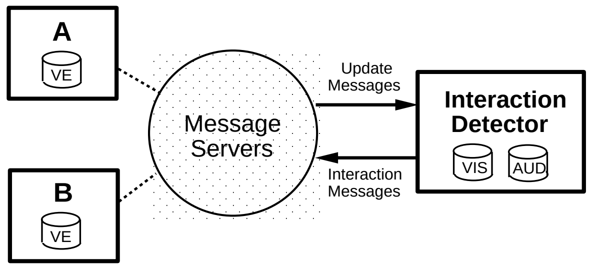

Figure 6: Interaction detection servers monitor entity updates and notify clients of potential entity interactions.

In RING, interaction detection servers compute potential visual interactions between entities and notify their clients via messages whenever an entity might become visible or invisible. Servers store a precomputed cell-to-cell visibility mask which contains over-estimates for visibility from any particular entity viewpoint (see Figure 7). Whenever an entity moves between cells, the server sends "Add Entity" messages to clients whose entities are in cells visible to the new cell, but not visible to the old cell. Likewise, "Remove Entity" messages are sent to clients whose entities are in cells visible to the old cell, but not visible to the new cell.

The server originated messages warn clients of potential visual interactions before they occur. Therefore, operation is tolerant of long message latencies. They allow clients to execute exact visibility algorithms considering fewer entities so that clients require less processing power, less memory capacity, and less network bandwidth. Since entities move between cells relatively infrequently, the extra messages sent by servers do not increase the network bandwidth to each client significantly. Similar techniques can be used for other types of interactions.

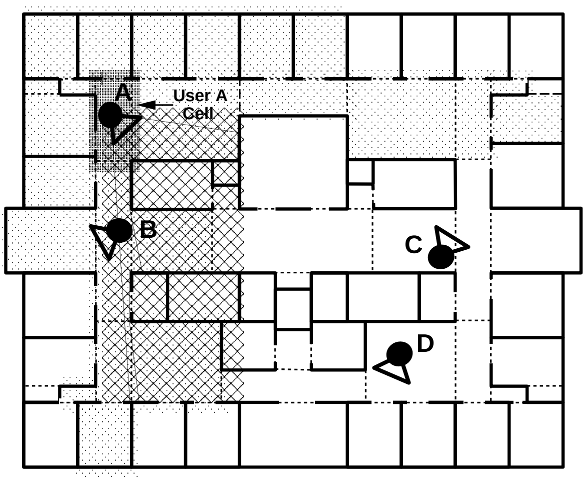

Figure 7: RING servers use precomputed cell-to-cell visibility (stipple) to predict potential visual interactions between entities.

5 Voice Bridging

In order to allow natural human interaction in a shared virtual environment, a system must support verbal communication between users. That is, users should be able to talk to each other (and hear each other) in a manner appropriate for the context of the shared 3D virtual environment. Assuming each client computer can capture an input voice stream with a microphone, and can play another (possibly stereo) output stream through headphones or speakers, the challenge for the system is to map the n input voice streams (one from each client) into a n output streams (one for each client). This mapping should simulate the properties of realistic sound propagation through the 3D environment.

The process of combining n input voice streams and producing a single output stream is called *voice bridging*. Currently available voice bridges generally use DSP processors to produce n output voice streams as weighted sums of n input voice streams [11]. This type of voice bridge can be represented by a matrix with element $a_{i,j}$ corresponding to the gain of $input_i$ for $output_j$. Conceptually, inputs come in on the left (rows) and are weighted to form outputs which come out on the bottom (columns).

We aim to approximate sound propagation through a virtual environment by adjusting the weights of a voice bridge dynamically so that voices are combined with gains appropriate for entity positions and orientations in the virtual environment. Of course, it is not possible to simulate complex sound propagation patterns exactly using this technology, but perhaps it is sufficient to provide spatial cues for speaker location and recognition. For instance, voices for users represented by entities further away in the virtual environment can be made to appear fainter ($a_{i,j} < 1$), and voices behind solid walls can be masked out entirely ($a_{i,j} = 0$). If stereo voice channels are available to each client, and delays or reverberations can be added in the voice bridge, we can provide further spatialization cues.

Clearly, supporting voice interaction in a purely peer-to-peer system is daunting, as it requires all n clients to receive and bridge $n-1$ voice streams transmitted by other clients. Practical clients have neither the network bandwidth nor the processing power to support such operations. However, it is feasible to support voice interaction in multi-user virtual environments by including shared *voice bridge servers* in the network.

The RING system design for voice bridging is shown in Figure 8. Clients use SVD modems to simultaneously send and receive voice streams along with update messages comprising entity positions and orientations. Each client's transmission is split into its voice and data parts, with the voice part going directly to a voice bridge, and the data part going to a voice bridge controller. The voice bridge controller updates the weights in the voice bridge dynamically based on the positions and orientations of entities in the virtual environment. A precomputed cell-to-cell audibility map (similar to the cell-to-cell visibility mask described in the previous section) is used to accelerate computations in the voice bridge controller.

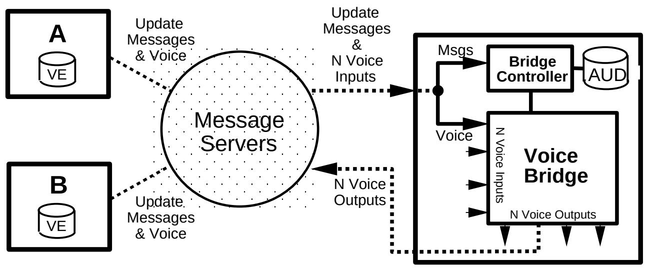

Figure 8: Voice bridges weight users' voices based on the position and orientation of entities in the virtual environment.

The implementation of real-time voice in virtual environments is still very experimental. Further research is required to determine which types of spatialization cues are valuable for meaningful communication among multiple participants.

6 Database Services

An important aspect of real-life virtual environment systems is support for database services, such as persistence, concurrency control, access control, and crash recovery. Many multi-user virtual environment applications require permanent updates to shared data. For instance, if a person makes a purchase in a virtual department store, the item should be removed from the store's inventory. Or, if an engineer modifies a part in a 3D design, the modification should be permanently available to other engineers sharing the design. It is important that only users with appropriate priviledges be granted access to shared data, and that no two users are able to modify the same shared data concurrently. If the system crashes, shared data should be recovered reliably.

For multi-user virtual environment systems, shared database operations are best implemented in database servers located logically in the network (see Figure 9). Although clients cache replicate copies of shared data locally, a "master" copy for each piece of persistent, shared data (i.e., not transient data – such as entity positions) is stored on a database server. Whenever a client makes an update to the shared data, a message is routed to the appropriate database server managing the data. The database server is responsible for protecting access to data, synchronizing updates, and updating client data caches after modifications.

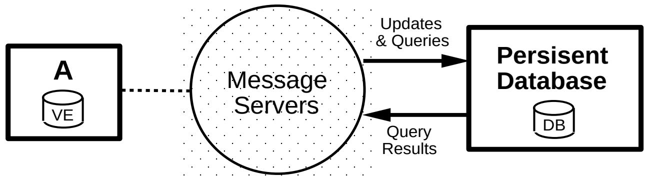

Figure 9: Database server provides persistent updates, concurrency control, access control, and crash recovery for a multi-user virtual environment.

Systems incorporating database servers simplify implementation of several services useful for multi-user virtual environments. For instance, database servers can provide access to on-line data (e.g., stock quotes, newspaper articles, banking transactions, etc.) so that appropriate up-to-the-minute and real-world data can be incorporated into virtual environments. Servers can also monitor and log updates in order to generate usage statistics, provide play-back for multi-player games, or support billing services. These features would be difficult to implement in a peer-to-peer system without complex agreement protocols and mutual trust between peers.

7 Conclusion

Multi-user virtual environment systems support interactions between many users in a shared virtual world. Many of the practical and technical challenges involved in implementing these systems can be addressed with a client-server design wherein multiple servers assist their clients with messaging, processing, and storage. Server computers: 1) coordinate the distributed system, 2) manage message distribution, 3) off-load processing, 3) support voice bridging, and 4) provide database services for the system. In a client-server system, the processing power, memory capacity, and network bandwidth requirements of clients is reduced significantly. Therefore, affordable multi-user systems can be constructed comprising mostly inexpensive client computers with low-bandwidth network connections, while a relatively small number of high performance server workstations connected by fast networks are needed to support these clients. As a result, client-server systems scale affordably to very large shared environments.

References

[1] Airey, John M., John H. Rohlf, and Frederick P. Brooks, Jr., Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24, 2 (1990), 41-50.

[2] Blanchard, C., S. Gurgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel, Reality Built for Two: A Virtual Reality Tool. *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, (Snowbird, Utah), 1990, 35-36.

[3] Blau, Brian, Charles E. Hughes, Michael J. Moshell, and Curtis Lisle, Networked Virtual Environments. *ACM SIGGRAPH Special Issue on 1992 Symposium on Interactive 3D Graphics*, (Cambridge, MA), 1992, 157-164.

[4] Calvin, James, Alan Dickens, Bob Gaines, Paul Metzger, Dale Miller, and Dan Owen, The SIMNET Virtual World Architecture. *Proceedings of the IEEE Virtual Reality Annual International Symposium*, September, 1993, 450-455.

[5] Carlsson, Christer, and Olof Hafsand, Dive: A Multi-User Virtual Reality System. *Proceedings of the IEEE Virtual Reality Annual International Symposium*, September, 1993, 394-401.

[6] Codella, C., R. Jalili, L. Koved, J.B. Lewis, D.T. Ling, J.S. Lipscomb, D.A. Rabenhorst, C.P. Wang, A. Norton, P. Sweeney, and G. Turk, Interactive Simulation in a Multi-Person Virtual World. *Proceedings of CHI'92*, May, 1992, 329-334.

[7] Cohen, J.D., M. C. Lin, D. Manocha, and M. K. Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. *ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3D Graphics*, (Monterey, CA), 1995, 189-196.

[8] Curtis, Pavel. Mudding: Social Phenomena in Text-Based Virtual Reality. 1992 conference on Directions and Implications of Advanced Computing, 1992.

[9] *Doom.* id Software, Mesquite, TX, 1993.

[10] Funkhouser, Thomas A. RING: A Client-Server System for Multi-User Virtual Environments. *ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3D Graphics*, (Monterey, CA), 1995, 85-92.

[11] Horn, David, and Atul Sharma. *A Versatile Audio Bridge for Multimedia Conferencing.* AT&T Bell Laboratories Technical Memorandum BL0113870-931210-05TM, December, 1993.

[12] Kazman, Rick, Making WAVES: On the Design of Architectures for Low-end Distributed Virtual Environments. *Proceedings of IEEE Virtual Reality Annual International Symposium*, September 1993, 443-449.

[13] Macedonia, Michael, R. Michael J. Zyda, David R. Pratt, and Paul T Barham, Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments. To appear in *Proceedings of IEEE Virtual Reality Annual International Symposium*, 1995.

[14] Naylor, Bruce F., John Amanatides, and William C. Thibault. Merging BSP Trees Yields Polyhedral Set Operations. *Computer Graphics* (SIGGRAPH '90). 24, 4, 115-124.

[15] Shaw, Chris, and Mark Green, The MR Toolkit Peers Package and Experiment. *Proceedings of IEEE Virtual Reality Annual International Symposium*, September 1993, 463-469.

[16] Singh, Gurminder, Luis Serra, Willie Png, Audrey Wong, and Hern Ng, BrickNet: Sharing Object Behaviors on the Net. *Proceedings of IEEE Virtual Reality Annual International Symposium*, March, 1995, 19-25.

[17] Teller, Seth J., and Carlo H. Séquin, Visibility Preprocessing for Interactive Walkthroughs. *Computer Graphics* (SIGGRAPH '91). 25, 4, 61-69.