`

An Implementation of a First-Person Game on a Hybrid Network

Anthony Steed*, Bingshu Zhu

Department of Computer Science, University College London

ABSTRACT

The majority of current networked virtual environments and networked games use a client-server model of networking. This makes synchronization of the environment simple, but it adds additional latency between any two clients. Peer-to-peer networking is a common alternative, but whilst these systems have potentially lower latency, they are more difficult to scale and lack a single point of synchronization.

In this paper we demonstrate an implementation of a form of hybrid networking, where clients communicate important state to a server, but communicate rapidly changing state peer-to-peer. By using the frontier set concept, we can make this scale since we ensure that only relevant position updates are sent peer-to-peer. The implementation is done as a modification to the game Quake 3 Arena. We show that the latency between clients is indeed reduced significantly for position events, and that this is achieved at a relatively small increase in network traffic.

CR Categories: C.2.4 [Computer Communication Networks]: Distributed Systems; I.3.2 [Computer Graphics]: Graphics Systems

Keywords: frontier sets, latency, synchronization, network scalability, networked virtual environments.

1 INTRODUCTION

Networked virtual environments (NVEs) pose a number of challenges to system designers [22]. The system designer must balance the requirement of ensuring consistent behavior of objects across a possibly widely distributed set of communicating clients, with the requirement of ensuring that any individual client has a continuous and usable experience. This is especially true in networked games such as first person games, where complete consistency between clients is unachievable across wide areas at a rate acceptable to the frenetic pace of the game. Thus in this type of games, the clients commonly run in a partially desynchronized manner. The main decision that a system designer has to make is to choose between a client-server system where synchronization is relatively easily determined by the server but where all messages must go via the server, and peer-to-peer systems where the synchronization problem is exacerbated, but the latency of any particular event is minimized.

In this paper we investigate a hybrid networking model, where communication is a mix of peer-to-peer and client-server, for the game Quake 3 Arena. Normally peer-to-peer communication in such a situation would be prohibitively expensive, however we use frontier sets to partition the world so each peer in the network can independently decide whether to send messages to other peers and thus each peer only communicates their position information with others peers that are likely to be visible. Bypassing the server in this way allows minimal latency between generation and rendering of positions of players in the game.

Reducing some of the sources of latency should provide a better experience for the players. This is because it can remove some discrepancies due to motion extrapolation and rollback that occur in such games where the server determines critical events and communicate these events to clients that are simply extrapolating a previous state. We show that hybrid networking is feasible in a real game scenario, and that, in packet count and throughput terms, it has a relatively small impact on the network load.

*A.Steed@cs.ucl.ac.uk

2 RELATED WORK

Today's NVEs have emerged from technologies developed for military simulators. Two dominant architectures have been pursued: peer-to-peer systems and client-server systems. SIMNET was an early example of a peer-to-peer system where each client simply broadcast information about its state on the network [18]. Such systems require a lot of bandwidth to cope with the volume of messages. There is also a general problem in ensuring consistency and security, problems acknowledged in the early MiMaze system [7].

The main alternative to peer-to-peer systems is client-server systems. In this type of system all clients connect to a server and that server is responsible for computing the state of the game and distributing it to all the clients. Consistency is easily managed since there is one canonical copy of the game state on the server. However a server introduces extra latency because any update to the game state needs to be relayed by the servers. The issue of latency is very important for real-time games, especially first-person shooter games which have become very popular in the last few years [2][10].

2.1 Partitioning

In order to have environments that scale to large numbers of users one common approach is to partition the world and only relay a subset of all events and state to each client [20].

One of the first systems to employ a partitioning scheme was the NPSNET system [16]. NPSNET divides the virtual world into fixed sized hexagonal cells. Each participant sends information (e.g. location updates) to their current local cell but can choose to receive information from potentially many cells that fall within their area of interest. The Spline system [26] divides a virtual world into arbitrary-shaped locales that are stitched together using portals. Each locale defines its own co-ordinate system and participants receive information from their current locale and its immediate neighbors. The ability to use variable-sized locales provides additional flexibility in coping with less predictable entities and is more appropriate for indoor environments.

Many games are built with environment models that have a lot of occlusion between models. One technique that is often used in such games is potentially visible sets (PVS) [27]. A PVS can be used to exclude a pair of entities from consideration for simulation

purposes because they are not mutually visible. However this visibility must be evaluated every time the entities concerned move. The RING system exploits a PVS data structure to increase the scalability of a client-server virtual environment [8]. In the RING system, a server culls messages if it knows that a client can't see the effect of the message.

Other systems propose to partition groups dynamically, depending on local awareness and neighborhood relationships [14][15][19]. These and similar schemes could provide for highly-scalable peer-to-peer networking, but have not as yet been used in practical implementations.

2.2 Latency

Latency in NVEs arises because of the physical transmission of the data and the cumulative processing latency of sender, receiver and router processes. Latency immediately induces inconsistency because an individual client "sees" the behavior of all other clients after a delay. Thus the player reacts to state when at the remote site that state may have already changed. A typical way this manifests in online FPS games is that the client sees and shoots at a target, but that target has moved. The role of the server is to be neutral and resolve such inconsistencies in as fair a way as possible. However, introducing a server causes slightly different problems, where players might think they have dodged a bullet, but because the server "sees" their behavior slightly delayed they subsequently get informed that they were shot in the past. This may require the client to roll-back to a previous state. A thorough survey of latency including a discussion of the inconsistencies that arise and strategies to combat them is given in [5][6].

2.3 Hybrid Networking

Hybrid networking is a term that is commonly used to refer to services that combine client-server and peer-to-peer communications. For example, Chen & Muntz propose a peer-clustering scheme that distributes some server responsibilities amongst peers [4]. In that scheme servers are necessary to support critical tasks, but peer clusters take over the handling of local interactions.

Our use of hybrid networking embodies a similar principle of delegating some responsibilities to the clients, but the goal is simpler: to reduce the latency with which certain events are seen by the clients.

2.4 Frontier Sets

The frontier set was introduced by Steed & Angus [24][25]. It is a concrete example of a more general class of algorithm called update-free regions [9][17]. These algorithms exploit the fact that if any two players know instantaneously where the other is, they might both be independently able to establish that they do not need to communicate until they leave an area. A simple example is two players on either side of a long wall. Until one or the other reaches the end of the wall, they can't possibly see their counterpart: for mutual visibility to be possible, one of them must round the wall.

A frontier set reifies this concept in the specific situation of a world where there is a visibility relationship such as a PVS [27]. If the environment is divided in to regions of space (or *cells*) then for any cell, it will be possible to identify openings (or *portals*) through which other cells can be seen. For any cell, it is possible to explicitly compute which other cells are visible from that cell, because if a cell is visible then there must be a line of sight through all the portals between them.

A single frontier is defined relative to pairs of cells in that subdivision. Given two cells A and B, a frontier comprises two sets of cells F_{AB} and F_{BA} such that no cell in F_{AB} is visible to a cell in F_{BA} and vice-versa. Figure 1 gives an example of a frontier. The complete set of frontiers for a whole environment will be referred to as a frontier set.

2.4.1 Example Usage

Consider two users moving around the environment depicted in Figure 1a. If Anne is in cell A, and Bob is in cell I at time t_0 then a frontier can be established, F_{AI} = {A,B,C,F}, F_{IA} = {G,H,I} as shown in Figure 1a. If Anne remains in the set of cells F_{AI} and Bob remains in the set of cells F_{IA} then they can never see each other. If this were a networked virtual environment this would mean that if Anne and Bob both exchanged location information at t_0 they would not have to send any further updates.

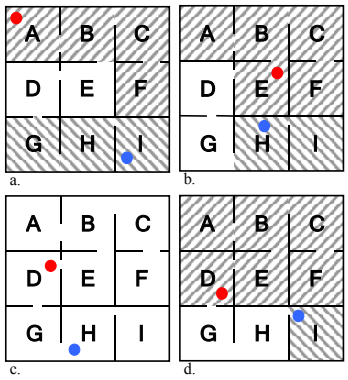

Figure 1. An example of frontiers in use. As users Anne and Bob move between cells, frontiers can sometimes be established. a) Anne and Bob are in cells A, I respectively. A frontier exists F_{AI} = {A,B,C,F}, F_{IA} = {G, H,I}. b) A frontier exists F_{EH} = {A,B,C,E,F}, F_{HE} = {H,I}. c) No frontier exists because cell D can see cell H. d) A frontier exists F_{DI} = {A,B,C,D,E,F} and F_{ID} = {I}.

The work of Steed & Angus was done on simulations using the Quake II code base [24][25]. The work presented in the remainder of this paper is live, playable system, implemented as a modification to the GPL version of the Quake 3 Arena code.

3 Architecture

3.1 Strategy

Even if peer-to-peer networking can be made to scale reliably, for many types of application, it will still be desirable to have a centralized server doing specific, application-critical state. This could be state that must be synchronized quickly so that clients do not diverge, or state that no client can be trusted to compute. In an FPS, examples might be the game score, and individual kills. However, much information is transient and updated frequently. In many systems, and particularly FPSs, this could be positions of players and other entities. In some engines such classes of information might even be separated on to different transport layers. The rationale for this is that if a position information packet is lost, it doesn't matter, and might even be detrimental to performance to retransmit the packet, as would be done with TCP.

The strategy with hybrid networking is thus to classify changing state into that which can be sent peer-to-peer, and that which needs to go client-server. There could be overlap between the two classes; that is information that is sent to both server and peer clients. Because sending directly to peers is low latency, this could be done where there are situation where the client believes that its information is correct and will not be altered by the server. A good example is position information in dynamic simulations. These systems commonly do dead-reckoning of position [18]. When a client observes that it needs to update the models of other

players, it would be best to notify all clients directly, as this will provide for lowest error when the dead-reckoning models adapt to the new information. Similarly in an FPS the information that I fired a weapon is useful to communicate as fast as possible to other players. The server in this FPS still needs to know the position and weapon firing immediately, as it needs to determine the overall effect of this.

A final strategic reason for distributing data both client-server and peer-to-peer is to provide redundancy for time-critical information, in case of loss of either peer-to-peer or client-server or server-client packets. Indeed, in certain situations, it might be necessary to fall-back to sending via a server anyway, because one of the peer-to-peer routes cannot be established due to firewall or other constraints.

3.1.1 Use of Frontier Sets

Although simply reducing the latency may be desirable in some situations, peer-to-peer traffic potentially incurs an overhead on network traffic. A *naïve peer-to-peer* algorithm generates $O(N^2)$ network traffic each frame, where N is the number of clients. A *perfect peer-to-peer* algorithm would send events only when they are necessary because they would be seen by the receiver. Obviously such an algorithm is un-implementable because the client would have to have prior knowledge of where the receiver was in order to know whether to send a packet. We thus propose to use frontier sets as an initial strategy to reduce traffic and make the system scale. It would be eminently feasible to have the server dynamically indicate to each client which other clients it would need to communicate to. The server could use a visibility data structure to determine this. However this introduces some latency in to the set up of communication.

Frontier sets work exclusively peer-to-peer, and in simulations for an FPS game they have been shown in simulation to achieve performance on the network very similar to the perfect peer-to-peer algorithm [25]. Further, in the FPS simulations, where the data sent peer-to-peer did not need to additionally be sent to the server, the frontier set algorithm was more efficient than client-server algorithm in certain situations. Frontier sets are relatively easy to implement, as their calculation does not need any negotiation with the peer or server. Using a peer-to-peer algorithm also reduces the latency of set-up of a particular peer-to-peer data flow: it doesn't require a server computation and thus data is sent from one peer to another as soon as the sending peer detects that it is necessary.

3.2 Impact on Latency

The key advantage of using peer-to-peer information is that it provides for lower latency: the information travels over only one link not two. However, there is another important advantage: even if the server is close to the "mid-point" of the link, that is the peer1-peer2 trip time is close to the sum of the trip time from peer1-server and server-peer2, the fact that we are cutting out a server process means that jitter in packet arrival time can be cut dramatically. Because the peers and server all update at a given rate, the server introduces a new source of process latency.

Once we introduce hybrid networking, each client sees their peers with minimum latency, thus reducing instantaneous inconsistency. However, furthermore, under the assumption that peer-to-peer and peer-server transit times are similar, the server is actually using the same state for calculations as the other peers are displaying at that time.

4 Implementation in Quake 3 Arena

Quake 3 Arena [11] is a first-person shooter that was originally released in 1999. It uses a client-server architecture. A machine can be set up as a dedicated server, or one participant can host a server. The server has an update rate of 20 Hz [1, p. 163]. Typical games have up to 32 participants. Like previous games in the series, Quake 3 Arena uses a cell partitioning of the world and there is PVS structure across this. The source code for Quake 3 Arena is available under the GPL, so it is easily extensible for experiments such as ours.

The first alteration that needs to be made is to set up data structures that allow frontier creation. As discussed in [24] in the context of Quake II, when a game world is loaded, the PVS data structure is converted in to an enhanced-PVS data structure. This takes 10-20 seconds depending on the map, and could be done as a pre-process.

The second alteration is to add a capability to notify each client of all its peers, so that they can make direct connections. There are additional messages to notify of clients leaving and departing the session.

The third alteration is the networking functions. Quake 3 Arena uses UDP distribution, with its own reliability mechanism. Snapshots of data are sent between client and server, and any resend is triggered only if necessary. Each snapshot packet contains a number of data structures, and we add a new data structure which is a peer-to-peer position data structure. The sending peer client simply sends a packet containing this data structure to the UDP port on the receiver client that the server normally connects to, and thus the receiver handles the packet as it would any packet from the server.

The main logic is thus on the sending side. Each client must know whether or not it currently has a frontier with another client. If it doesn't have one, based on the last position packet received, doesn't know, or it has left its half of the frontier then it sends a position packet.

To implement this each client calls a function, *CL_SendNetworkUpdate* each frame. The main job of this function is to establish if, since the last frame, this client or one of other clients have left the agreed frontier. If they have, they get rid of the current frontier. If there is no current frontier then it sends an update to the other client. Finally it tries to establish a new frontier with the current cells for this client and the other client. The client keeps two arrays frontierThis and frontierOther. Each element of these arrays is one of the sets of cells from a frontier. The space for each element much be a binary vector the length of the maximum number of cells in a map. This is not large compared to other static resources that are allocated in memory by the game engine. The process of building frontiers is almost identical to the process discussed in [25] and the implementation borrows heavily from the Quake II code. In essence though, the process requires a single iteration through the list of cells in the map, to construct the two cells lists for the frontier, if they exist. If the two cells are mutually visible, this function returns immediately.

One key aspect of the implementation is that peers send position updates at their frame rate. This can be much more than would be relayed by the server (i.e. 20Hz), but is a reasonable strategy for some situations as it results in much smoother movement. In our tests described in the next section we used fairly modern PCs, and Quake 3 Arena does not stress modern graphics cards. If some of the machines on the network were likely to be more stressed, then rate limiting the peer-to-peer sending would be necessary.

Final changes included logging capabilities to determine the latencies and data usage of the different networking routes. The server process collates all data logged by individual clients.

5 Preliminary Results

Preliminary analysis was done on a small number of two player games to establish throughput and latency characteristics. Latency

TEST	CLIENT	FRAME	SV PACKET	CL PACKET	SV BYTE	CL BYTE
1	A	7997	1795	0	35484	0
	B	7996	1795	0	50235	0
2	A	6949	1565	6957	51980	132541
	B	6957	1565	6949	57605	138111
3	A	9822	2209	582	68620	11659
	B	9817	2209	582	46049	11182

Table 1: Packets and bytes sent by each client, both to the server (SV) and to the other client (CL). The tests are distinguished by the amount of time which the two players can see each other.

is tricky to calculate due to the problems inherent in synchronizing clocks across the network. Figure 2 gives a reference diagram for the timings we have used.

t_{A1} to t_{A4} are client A's local time; t_{B1} to t_{B5} are B's local time and t_{SV1} to t_{SV2} are server's local time. In the game, when client A shoots towards client B at time t_{A1} (meanwhile, B is at time t_{B1}), A will send a packet to B indicating this event. This function is added for testing purpose. A will also send a packet to server and that is an original standard function of Quake 3 Arena. B receives this packet at time t_{B2} and the corresponding time at A is t_{A2}. B processes this event then sends back a response packet to A at time t_{B3}. A gets this response packet at time t_{A4}. The P2P one-way transmission latency Δt_1 is considered to be the average of $t_{A2} - t_{A1}$ and $t_{A4} - t_{A3}$. Since it is hard to synchronize the time on client A and B, the measurable time includes: t_{A1}, t_{B2}, t_{B3}, t_{A4}. Thus the one way P2P transmission is:

$$\Delta t_1 = \frac{(t_{A4} - t_{A1}) - (t_{B3} - t_{B2})}{2}.$$

The P2P overall latency (two-way transmission latency plus processing latency) is simply:

$$\Delta t_2 = t_{A4} - t_{A1}.$$

When A shoots B, another packet is sent to the server. The server gets it at time t_{SV1} then sends a packet to inform B at time t_{SV2}. $t_{SV2} - t_{SV1}$ is the processing latency at the server. B gets this packet from server at time t_{B5}. In the testing plan, overall latency of the message through server is measured and denoted as:

$$\Delta t_3 = t_{B5} - t_{B2} + \Delta t_1.$$

The three latencies were measured by observing 20 packets sent both ways through the network, as shown in Table 2. We can see here that there is a very marked difference between the one-way and client-server communication.

Data rates were also calculated. Rates vary drastically depending on whether the two players can see each other. In Table 1, Test 1 is a situation when clients are in valid frontier sets so they do not need to communicate. In Test 2 valid frontier sets have never been able to be set up. The figures show that clients are sending packets peer-to-peer at the frame rate. In this extreme situation communication between client and server is much less than that between clients. This is because the original Quake 3 Arena server does not send packets to client at the frame rate, but at a fixed lower rate. In addition, Quake 3 Arena does delta-compression on the data packets so packet size is normally small. From the table it could be calculated that the average packet size of client-server Quake 3 Arena is 27.8 bytes and that of peer-to-peer Quake 3 Arena is 19.5 bytes. The new Quake 3 Arena being tested only sends player position information peer-to-peer so 19.5

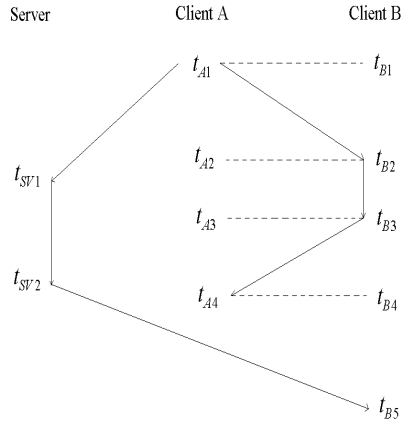

Figure 2. Time stamps used in the calculation of difference in latency for client-server and peer-to-peer communication.

Timing	Δt_1	Δt_2	Δt_3
Mean	6.375	12.95	28.25
Std.dev	2.804	5.633	11.77

Table 2 Timings of peer-to-peer single way, peer-to-peer two way, and client-server communication

is quite a high number. In the future the modified Quake 3 Arena could also perform compression on the packets. Test 3 is close to the real situation of playing. In this situation the traffic between the peers is roughly 25% of that to the servers.

6 CONCLUSIONS

We have demonstrated a practical implementation of hybrid networking as a modification to the Quake 3 Arena game. We showed that we could reduce instantaneous inconsistency by reducing client-client position update latency and that this could be done with reasonable increase in network traffic.

Currently this is a proof-of-concept demonstration, though there is no theoretical reason why this should not scale to support larger numbers of players. Although we only presented preliminary tests with two players, the game does function correctly with more players and large scale tests are being planned. The previous simulations on Quake II, [24][25] should indicate that the total packet overhead for 16 or 32 player games would not be onerous .

Future work will look at several issues: delegating more responsibilities to the players, stress testing with more players, subjective and qualitative review of the impact of latency reduction on player experience. We would highlight some expected problems that are common to any peer-to-peer scheme, such as maintaining consistency when there are dense clusters of players. This is could be a problem because at least with a client-server system the peak load on the network, servers and clients, is well known, whereas in peer-to-peer systems peak load is harder to calculate.

The modified Quake 3 Arena code is available on *http://www.cs.ucl.ac.uk/staff/A.Steed/*

REFERENCES

[1] ARMITAGE, G., CLAYPOOL, M., BRANCH, P. 2006 Networking and Online Games: Understanding and Engineering Multiplayer Internet Games, Wiley. k

[2] BEIGBEDER, T., COUGHLAN, R., LUSHER, C., PLUNKETT, J., AGU, E., AND CLAYPOOL, M. 2004. The effects of loss and latency on user performance in unreal tournament 2003®. In Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support For Games (Portland, Oregon, USA, August 30 - 30, 2004). NetGames '04. ACM Press, New York, NY, 144-151.

[3] CECIN, F. R., DE OLIVEIRA JANNONE, R., GEYER, C. F., MARTINS, M. G., BARBOSA, J. L. 2004. FreeMMG: a hybrid peer-to-peer and client-server model for massively multiplayer games. In Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support For Games (Portland, Oregon, USA, August 30 - 30, 2004). NetGames '04. ACM Press, New York, NY, 172-172.

[4] CHEN, A. AND MUNTZ, R. R. 2006. Peer clustering: a hybrid approach to distributed virtual environments. In Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support For Games (Singapore, October 30 - 31, 2006). NetGames '06. ACM, New York, NY, 11.

[5] DELANEY, D , WARD, T., S.MCLOONE 2006. On consistency and network latency in distributed interactive applications: a survey--part I, Presence: Teleoperators and Virtual Environments, v.15 n.2, p.218-234, April 2006.

[6] DELANEY, D , WARD, T., S.MCLOONE 2006. On consistency and network latency in distributed interactive applications: a survey--part II, Presence: Teleoperators and Virtual Environments, v.15 n.4, p.465-482, April 2006.

[7] DIOT, C. GAUTIER, L. 1999. A Distributed Architecture for MultiParticipant Interactive Applications on the Internet. In *IEEE Network*, 13(4), 6-15.

[8] FUNKHOUSER, T. A. 1995. RING: A Client-Server System for Multi-User Virtual Environments. In *1995 Symposium on Interactive 3D Graphics*. 85-92, April 1995.

[9] GOLDIN, A., GOTSMAN, C. 2004. Geometric message-filtering protocols for distributed multiagent environments. Presence: Teleoperators and Virtual Environments, 13(3), 279-295.

[10] HENDERSON, T. 2001. Latency and User Behaviour on a Multiparticipant Game Server. *Networked Group Communication 2001*, Third International COST264 Workshop, London, UK, November 7-9, 2001 1-13

[11] IDSOFTWARE. 1999. Quake 3. http://www.idsoftware.com/games/quake/quake3/

[12] IEEE. 1993. ANSI/IEEE Standard 1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation, March 1993.

[13] KELLER, J., SIMON. G. (2003) Solipsis: A massively multi-participant virtual world. In Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03).

[14] KAWAHARA, Y., MORIKAWA, H., AND AOYAMA, T. 2002. A peer-to-peer message exchange scheme for large scale networked virtual environments. In Proceedings of the the 8th international Conference on Communication Systems - Volume 02 (November 25 - 28, 2002). ICCS. IEEE Computer Society, Washington, DC, 957-961.

[15] KNUTSSON, B. LU, H., XU, W., HOPKINS. B. (2004) Peer-to-peer support for massively multiplayer games. In Proceedings of the 23rd Conference of the IEEE Communications Society (Infocom 2004), Washington, D.C., 2004. IEEE Computer Society.

[16] MACEDONIA, M. R., ZYDA, M. J., PRATT, D. R., BARHAM, P. T., ZESWITZ, S. 1994. NPSNET: A Network Software Architecture for Large Scale Virtual Environments. *Presence: Teleoperators and Virtual Environments*, 3(4): 265-287, MIT Press.

[17] MAKBILI, Y., GOTSMAN, C., BAR-YEHUDA, R. (1999) Geometric Algorithms for Message Filtering in Decentralized Virtual Environments. Proceedings of the ACM Symposium on Interactive 3D Graphics, 39-46.

[18] MILLER, D., AND THORPE, J. 1995. SIMNET: the advent of simulator networking. *Proceedings of IEEE*, 83(8): 1114-1123.

[19] MORILLO, P., MONCHO, W., ORDUÑA, J.M., DUATO, J. 1996. Providing Full Awareness to Distributed Virtual Environments Based on Peer-to-peer Architectures, in Computer Graphics International (CGI'06), volume 4035 of Springer LNCS, pp. 336-347

[20] MORSE, K. L., BIC, L. AND DILLENCOURT, M. 2000. Interest management in large-scale virtual environments. *Presence: Teleoperators and Virtual Environments*, 9(1):52--68, MIT Press.

[21] QuakeWorld, http://en.wikipedia.org/wiki/QuakeWorld

[22] SINGHAL, S. ZYDA, M. 1999. Networked Virtual Environments: Design and Implementation. Addison-Wesley.

[23] SMED, J. KAUKORANTA, T. AND HAKONEN, H. 2001. Aspects of Networking in Multiparticipant Computer Games. In Loo Wai Sing, Wan Hak Man, and Wong Wai (eds.), *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*. Hong Kong SAR, China, Nov. 2001, 74-81.

[24] STEED, A., ANGUS, C. 2005, Supporting Scalable Peer-to-peer Virtual Environments Using Frontier Sets. In Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality (March 12 - 16, 2005). VR. IEEE Computer Society, Washington, DC, 27-34.

[25] STEED, A., ANGUS, C. 2006, Enabling scalability by partitioning virtual environments using frontier sets. Presence: Teleoperators and Virtual Environments, 15 (1). pp. 77-92.

[26] STERNS, I.B., YERAZUNIS, W.S. 1997. Diamond Park and Spline: Social Virtual Reality with 3D Animation, Spoken Interaction and Runtime Extendability. *Presence: Teleoperators and Virtual Environments*, 6(4), 461-481, MIT Press

[27] TELLER, S.J. SEQUIN, C.H. 1991. Visibility Preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91),* 25(4):61-90.