R96-6

# ELIMINATING NETWORK TRAFFIC CAUSED BY DETERMINISTIC OBJECTS IN MULTI-USER VIRTUAL WORLDS

JDS Pugh
*Networking Research Group*
*jds_pugh@postoffice.utas.edu.au*

June 1996

## ABSTRACT

Virtual worlds with large numbers of users have been realised using unreliable multicasts and dead reckoning techniques. Deterministic objects have had to be modelled as non-deterministic objects using these techniques in such systems, but there is scope for reducing the amount of network traffic associated with deterministic objects by using algorithms based on the Time Warp method for distributed simulation. This paper introduces a software architecture called *Replicated Interactive Time Warp* that eliminates network traffic caused by deterministic objects, while maintaining consistency among virtual worlds. This occurs at the expense of extra local computation and brief anomalies in the user interface due to network latency and the optimistic approach taken for concurrency control.

**Keywords and phrases**

Distributed Virtual Reality, Multi-User Virtual Worlds

## INTRODUCTION

Ever since the concept of virtual worlds has existed one of the main aims has been to enable multi-user participation. Remotely located users should be able to meet in a virtual world as though they were actually there.

For the purposes of this discussion a *virtual world* is defined to be a real-time simulation with a graphical user interface. Users can interact with the virtual world via this interface. Each user will typically interact with a virtual world running on a dedicated computer that holds a copy of the representation of the virtual world required for rendering. Each computer communicates via the network to inform other participants of updates to the virtual world and in order to receive updates from other participants.

A number of multi-user systems have been developed, for example SPLINE (Anderson *et al*. 1995), DIVE (Carlsson & Hagsand 1993), MASSIVE, GreenSpace (Mandeville *et al*. 1995). Their 'size and richness' has been limited by their network capabilities, and many systems are only able to handle 'a handful of users' (Macedonia & Zyda 1996). Systems based on multicasting and dead reckoning have been the most successful and can support hundreds of simultaneous users, for example NPSNET (Macedonia *et al*. 1994).

It is difficult to increase the number of simultaneous users in a virtual world due to the non-deterministic nature of their actions and consequent network traffic. *Dead reckoning* takes advantage of physical constraints such as inertia to approximate future states of the objects in the virtual world. In its simplest form, it reduces the frequency that updates need to be sent to other participants by sending them only after the object has deviated from its predicted path by a predetermined error (see Pratt 1993). This method also handles missing network messages from unreliable delivery satisfactorily due to the frequency of updates. Depending on the characteristics of object concerned, these are sent typically three times per second or at least every five seconds (Macedonia 1994). If one message is missed it will not be too long before the next one is received, setting the object back onto its correct course. The number of users in the system can be increased by using better dead reckoning algorithms to reduce the network traffic, but there is a fundamental limit for non-deterministic objects because their future states cannot, by definition, be precisely predicted.

A virtual world also includes *deterministic objects*. Deterministic objects have completely predictable and repeatable behaviour. There is scope for reducing network traffic caused by them, or even eliminating it. Even seemingly random behaviours can be introduced in a deterministic manner using pseudo random number generation. No communication need occur between virtual worlds if all the objects are deterministic and the virtual worlds are exact replicas of each other.

When non-deterministic events affect a deterministic virtual world it is possible to maintain consistency among replicas if each event is introduced consistently into each virtual world. This means introducing the event at the same time at each replica. Time is important because each virtual world will be dynamically evolving, so an event will have different effects depending upon when it is introduced.

Figure 1. illustrates this time dependency by considering a virtual world consisting of a deterministic 'puck' object bouncing around in a box. The two virtual worlds in the figure are replicas of each other and are connected via a network. At a certain point in time a non-deterministic event is inserted into the virtual world that causes the box to be partitioned into two halves. In the virtual world where the non-deterministic event originated, the puck is trapped in the left hand side of the box. Due to the speed of light limitation, the remotely located box will receive the event later, by which time the puck could have entered the right hand partition, causing the virtual worlds to evolve inconsistently.

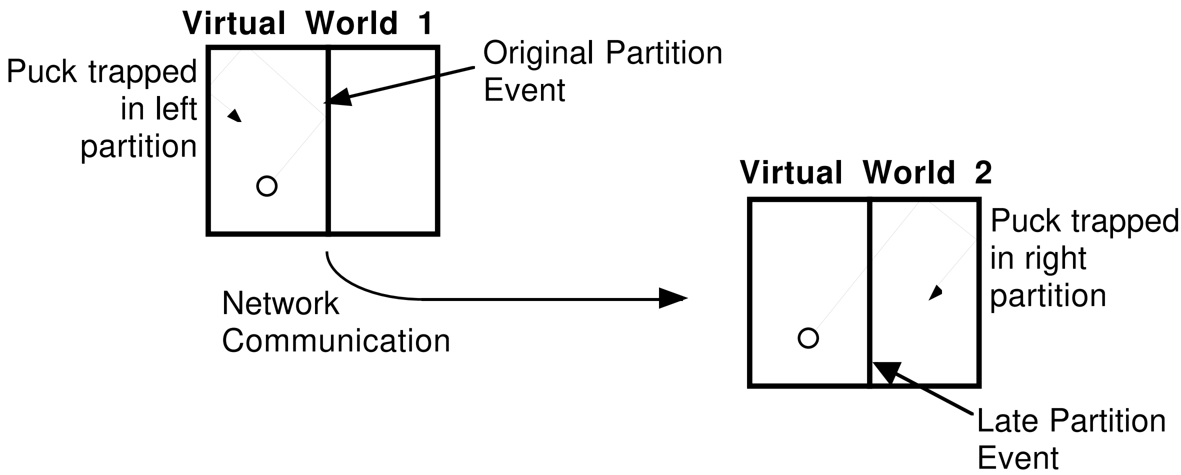


*Figure 1. Inserting a Non-Deterministic Event into a Deterministic Simulation: Time Matters*

To ensure consistency the simulation must jump back, or rollback, in time. The late event must then be inserted, and a roll forward in time needs to be performed to ensure that the user perceives time to be steadily increasing. A system that can insert events in this consistent manner will be able to eliminate network traffic caused by deterministic objects and will only have to send information about non-deterministic events.

Even though deterministic objects can be accurately dead reckoned, dead reckoning does not support rolling backwards and forwards in time, and so cannot guarantee consistency among virtual worlds without regular updates for all objects.

This report introduces a new system called *Replicated Interactive Time Warp* that takes advantage of the predictability of deterministic objects and attempts to perform rolling backwards and forwards in time efficiently. This allows network traffic to be eliminated for deterministic objects at the expense of the extra local computation required to perform the rolling through time.

## REPLICATED INTERACTIVE TIME WARP

The Replicated Interactive Time Warp (RITW) architecture is based on the Time Warp architecture for distributed simulation (Steinman 1995). The Time Warp architecture can roll through time efficiently and is also suited to the construction and parallel execution of virtual worlds. In this case replicas of the Time Warp simulation are run on separate computers, typically one computer for each user. Each individual Time Warp simulation is synchronised with the local real-time clock. If a computer cannot process events fast enough to keep up with the local real-time clock then the simulation cannot be allowed to communicate with other simulations because the events sent by the slow machine will cause excessive rollbacks on the machines that are keeping up with their local real-time clocks.

Simulations are synchronised with each other by synchronising local real-time clocks. Inaccuracy in synchronisation of the clocks can adversely affect the performance of the system by causing excessive rollbacks in the computers farthest in the future. Several methods are available for synchronising local real-time clocks over a network. The basic premise is to calculate the approximate network latency by sending a message and receiving an immediate reply. The latency is approximately half the round trip delay. The sent synchronisation time can thus compensate for network latency. Systems such as the Network Time Protocol can get typical errors of less than 30 milliseconds over WANs (Coulouris *et al.* 1994). Since non-deterministic objects are not likely to be updated much more than every 33 milliseconds (once per frame for 30 frames per second), an inaccuracy of 30 milliseconds means that clock inaccuracy will not account for much more than one rollback for each non-deterministic event inserted into the simulation: an acceptable statistic.

When each Time Warp simulation replica is self contained with objects that have deterministic behaviours only, the simulations remain synchronised without any further communication. It is when non-deterministic events are inserted into the simulation that communication needs to occur. Each event will contain the following information:

*Target Object ID*. An ID that uniquely identifies the target object for the event (eg. puck 1).

*Event Type*. The type of the event the object is to execute (eg. move).

*Event Data*. Data for the event (eg. position{5,8}).

*Scheduled Time*. The time at which the target object is required to process the event (eg. 103465 milliseconds).

If these events are communicated reliably, all simulations can remain consistent with the help of a rolling mechanism.

## ROLLBACK

To support rollbacks in time when late events are received, the following information is stored for each RITW object (see also figure 2.):

*Simulation Time*. The scheduled time of the object's last processed event.

*Input Event Queue*. A priority queue consisting of incoming events prioritised by their scheduled times.

*Event Store*. The events that have been processed by the object from the Input Event Queue.

*State Store*. The states of the object at various points in time.

*Output Event Store*. The events generated by the object after processing events from the Input Event Queue.

An event is defined to be *late* if its scheduled time is earlier than its target object's simulation time. Each time an event that is not late arrives for an object, it is placed on its input event queue. When an event needs to be processed, it is drawn from the top of the input queue (that is, the event with the earliest scheduled time). Before execution this event is placed on the event store, the state of the object is placed on the state store and the object's simulation time is set to the event's scheduled time. After execution, any events generated from the processing of the input event are scheduled to their appropriate objects and recorded on the output event queue.

If an event arrives late the object will rollback in time so that the late event can be inserted correctly (figure 2.). The state of the object is restored to the latest state before the late event's scheduled time (states later than the restored one can now be removed as they will no longer be needed). The object's simulation time is set to the time the state was stored at. The late event is then placed on the input event queue, along with the saved events from the event store scheduled to be executed after the late event. Any events that have been scheduled by the object in the period rolled back (between the object's new simulation time and the object's old simulation time) could have been scheduled incorrectly because of the absence of the late event's effects. These can be cancelled by immediately sending out corresponding anti-events (aggressive cancellation). Alternatively they can be cancelled by sending anti-events only when the output events are verified, during the rollforward, to have been sent incorrectly (lazy cancellation, see Steinman 1995). Other objects are thus rolled back only as a secondary affect.

Note that anti-events assist in the rollback process of individual Time Warp simulations. They are not sent between Time Warp simulation replicas.

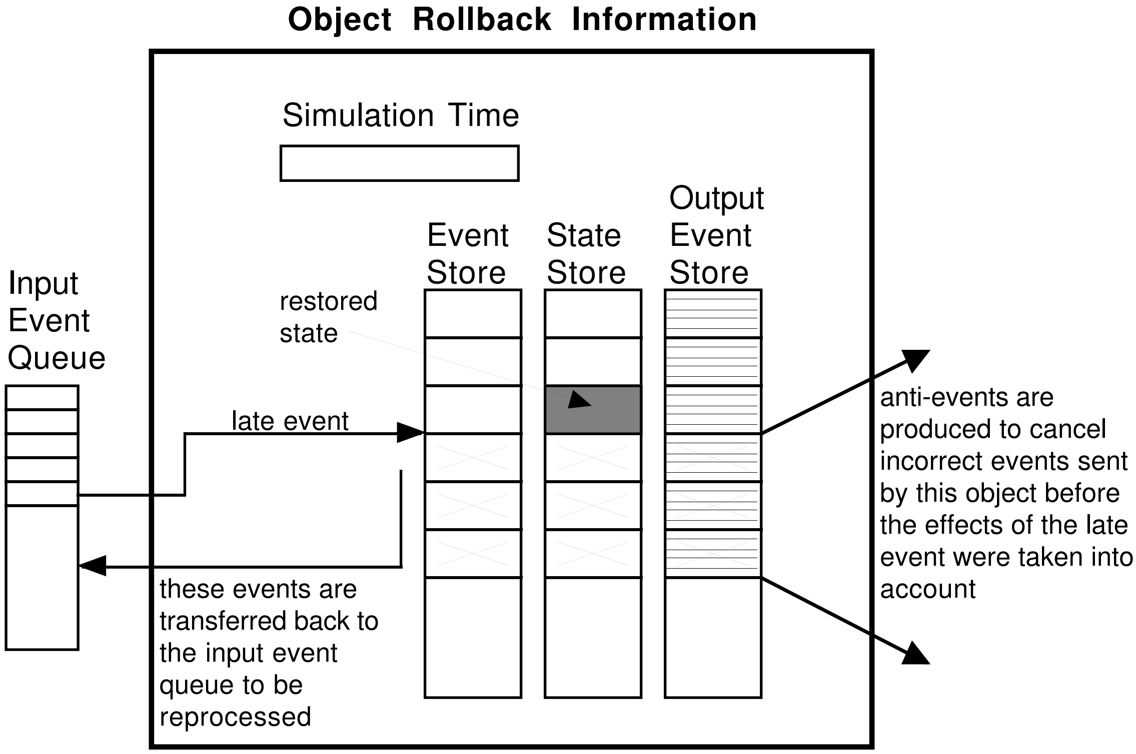


*Figure 2. Rollback of a Deterministic Object*

In order to avoid confusing the user when a rollback occurs, time should continue to appear to remain in sync with the local real-time clock. To do this, a rollforward of any rolled back objects to the current local real-time needs to occur as soon as possible. This involves the object executing its input events until its simulation time has caught up with the local real-time clock.

The effects of the insertion of the late event and the rolling through time will cause anomalies visible to the user if the network latency is significant. Take for instance the example in figure 1. where the box is partitioned. If the remote user receives the partition event late, they may perceive the puck enter the right hand partition. When the late event is received and the rolling has been completed, the puck will have jumped from the right hand partition to its correct position in the left hand partition. The severity of the anomalies produced will be proportional to the network latency. This is the price paid for performance benefits of an optimistic approach to concurrency control and cannot be avoided.

Because other objects are rolled back only as a secondary effect and the rollback process is relatively simple, rollbacks can occur very quickly. Lazy cancellation can aid the process further by reducing the number of secondary rollbacks.

The prototype implementation (on a 33Mhz 68040 Macintosh) has shown that the computation of simple object behaviours such as the bouncing puck's takes only a small amount of time. This means that it is possible to perform rollforwards quickly without consuming a large amount computational power. In the prototype implementation pucks in the order of thousands can be moved per second (hundreds of pucks moving ten times per second). Further code optimisation can increase this figure. When complex behaviours are introduced (eg. a flight simulation model), rollforward becomes more time consuming.

To enable rollback, the states of the RITW objects are required to be stored regularly. In traditional Time Warp simulation Global Virtual Time (GVT) is calculated at regular intervals. GVT is defined as the earliest scheduled time of an unprocessed event (or event in transit) in a distributed simulation (Steinman 1995). Rollbacks will thus not occur past GVT (events can't be scheduled into the past). This means that any states earlier than GVT can be removed to make room for future states. In the RITW system this value can be calculated at each replicated Time Warp simulation. Since there are multiple Time Warp simulations running that can interact with each other, there needs a notion of a global GVT: the earliest GVT amongst the Time Warp simulation replicas. This is expensive to calculate because it needs to receive a GVT value from each Time Warp simulation replica and needs to take into account events that are in transit on the communications medium. It involves a lot of network traffic, which defeats the purpose of this software architecture.

For the RITW system a scheme that is less reliable, but does not decrease performance can be used. States can be stored for a time long enough to almost guarantee that arriving events won't be scheduled before the earliest stored state. Alternatively they can be stored until available memory is used; old states can even be swapped to disk. When an event does arrive before the earliest stored state the simulation cannot perform the rollback and must fail or resynchronise from its peers. Techniques are available for reducing the memory requirements for state storage in Time Warp simulations and can be applied to the RITW simulations. One such technique is incremental state storage where the changes in state are stored, rather than the states themselves (Steinman 1995).

Traditionally in Time Warp simulation it is accepted that interactive input (non-deterministic events in the RITW system) cannot be rolled back. They are scheduled to be performed no later than GVT so that they guaranteed not to be rolled back. This can cripple the interactive performance of the simulation, especially when network latency is large. Because the RITW system in intended to be used even when network latencies are large, these events are simply scheduled to occur immediately and will take part in the rollback process. These events are tagged so that they do not produce corresponding anti-events. This means that once they are in the simulation they remain there, even through rollbacks. The consequence of this method is that the user may act on an erroneous virtual world state that exists before some late event is inserted. This may cause a problem because non-deterministic events will be able to be produced in an illegal order; for instance and object could be deleted and then moved. In this case some application dependent resolution method must be applied. An example of this is given later in the example virtual world.

An issue that must be looked at when considering the performance and scalability (in terms of the number of users that can be supported) of the RITW system is the performance and scalability of the underlying communications technology. The largest network latencies will be observed when reliable multicast events are lost and need to be retransmitted. New reliable multicast schemes that use negative acknowledgements should meet the requirements of the RITW system, for example Holbrook *et al.* (1995), Whetten *et al.* (1995) and Koifman & Zabele (1996). It is possible that the negative acknowledgement scheme could be integrated into the RITW mechanism for increased efficiency. The reason being that to enable negative acknowledgements it is necessary to keep track of past events that have been transmitted. These are also to be stored as part of the RITW mechanism.

The majority of the network traffic will typically be caused by non-deterministic objects (updating their positions and orientations) and so can be handled by unreliable multicasts. Reliable multicasting is only needed when non-deterministic objects interact with deterministic objects, injecting non-deterministic events into the RITW simulation.

## AN EXAMPLE VIRTUAL WORLD

An example virtual world has been created using a prototype implementation of the RITW system. The virtual world consists of non-deterministic user controlled ‘space ships’ and deterministic ‘pucks’ that bounce around the screen (the user interface is two dimensional for simplicity). The non-deterministic ships use Dead Reckoning and multicasting to reduce their network traffic, and there is one ship per user.

Pucks are modelled using the RITW system and can collide with ships and will bounce off them. Pucks move by executing locally scheduled ‘move’ events. Each time a puck executes a move event, a new one is scheduled slightly into the future. The number of move events that are rolled back when a late ‘collision’ event arrives targeted at the puck is approximately equal to the network latency divided by the interval between puck ‘move’ events. If the pucks are updated at a reasonable rate of ten times each second, this means that there will typically be only one or two rollbacks caused by late collision events when the network latency is around one to two hundred milliseconds — a reasonable figure for a WAN.

To enable collision between pucks and ships, each time a move event is executed the puck sends its new position to the local user controlled ship. Anti-events are not produced for this event because they are not being sent to a Time Warp object. If the local ship detects a collision, it schedules a collision event to occur immediately (scheduled for the current local real-time), targeted at the puck involved. As this is a non-deterministic event being injected into the RITW system, it is multicast to remote RITW simulations so that consistency can be maintained. The collision event specifies a new position and velocity for the puck.

Figure 3. illustrates what happens when two non-deterministic collision events directed at the same puck occur before either knows of the other’s existence (that is, concurrent access to the puck ). When each late event is received by its respective virtual world, it is simply inserted into the simulation at the correct time. The result is that both virtual worlds process the same events at the same times in the same orders, and so remain consistent.

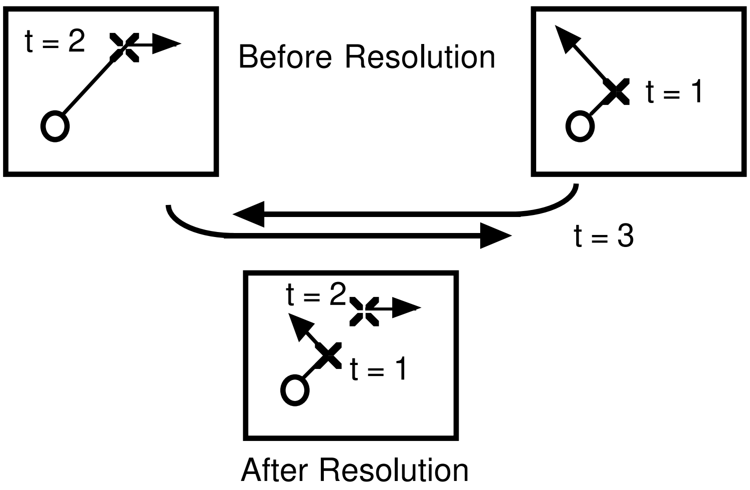


*Figure 3. Resolution Example 1*

Figure 4. shows what happens when an illegal ordering of non-deterministic events is obtained through concurrent access to an object. At time t = 1 the puck is deleted and at time t = 2 the puck is moved. To resolve the situation after each

virtual world receives the late event (after time t = 3), the move event is simply ignored. The net effect is that the user that deleted the puck observes no ill effects, whereas the user that moved the puck will have their virtual world adjusted to conform with the common view given after resolution: the object is just deleted at time t = 1.

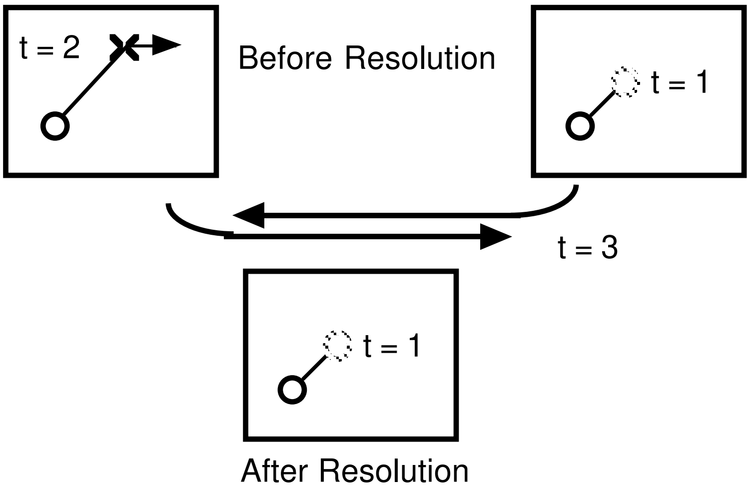


*Figure 4. Resolution Example 2*

## CONCLUSION

The RITW system comes into its own when large numbers of simple deterministic objects are present in a virtual world and the virtual worlds are linked via low bandwidth network connections. This makes it ideal for use on the Internet where bandwidth is scarce, but relatively powerful personal computers are connected that can handle the demands of the RITW system. The system will allow developers to create richer and larger multi-user virtual worlds than could have been possible before on low bandwidth networks by reducing network traffic at the expense of extra local computation.

## REFERENCES

Anderson, D., Barrus, J., Howard, J., Rich, C., Shen, C. & Waters, R. 1995. 'Building Multi-User Interactive Multimedia Environments at MERL', *IEEE MultiMedia*, Winter, 1995.

Barrus, J. W., Waters, R., C. & Anderson, D. B. 1995. 'Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments', Mitsubishi Electric Research Laboratories, Cambridge Research Center, Cambridge, Massachusetts. http://www.merl.com/TR/TR95-16/Welcome.html.

Carlsson, C., & Hagsand, O. 1993. 'DIVE — A Multi User Virtual Reality System', *IEEE Annual Virtual Reality International Symposium*, September 1993, IEEE Service Center, ISBN 0-7803-1364-X.

Coulouris, G., Dollimore J. & Kindberg T. 1994. *Distributed Systems Concepts and Design*, Queen Mary and Westfield College, University of London, 2nd Edition, Addison-Wesley, ISBN 0-201-62433-8.

Holbrook, H., Singhal, S. K. & Cheriton, D. R. 1995. 'Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation', to be presented at SIGCOMM 95.

Koifman, A. & Zabele, S. 1996. 'RAMP: A Reliable Adaptive Multicast Protocol', *Fifteenth Annual Joint Conference of the IEEE Computer and Communication Societies*, San Francisco, CA, March 26-28, 1996.

Macedonia, M. R., Zyda, M. J. 1995, 'A Taxonomy for Networked Virtual Environments', *Proceedings of the 1995 Workshop on Networked Realities*, Boston, MA, October 26-28, 1995. http://www-npsnet.cs.nps.navy.mil/npsnet/publications.html.

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', Computer Science Department, Naval Postgraduate School, Monterey, California. Presence, Vol. 3, No. 4. http://www-npsnet.cs.nps.navy.mil/npsnet/publications.html.

Macedonia, M. R., Zyda, M. J., Pratt, D. R., Barham, P. T. & Zeswitz, S. 1995. 'Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments', Computer Science Department, Naval Postgraduate School, Monterey, California. *IEEE Annual Virtual Reality International Symposium*, 11-15 March, 1995, RTP, North Carolina. http://www-npsnet.cs.nps.navy.mil/npsnet/publications.html.

Mandeville, J., Furness III, T., Kawahata, M., Campbell, D., Danset, P., Dahl, A., Dauner, J., Davidson, J., Howell, J., Kandie, K. & Schwartz, P. 1995. 'GreenSpace: Creating a distributed Virtual Environment for Global Applications', *Proceedings of the IEEE Networked Reality Workshop*, October 26-28, 1995, Boston, MA.

Pratt, D. R. 1993. 'A Software Architecture for the Construction and Management of Real-Time Virtual Worlds', Department of Computer Science, Naval Postgraduate School, Monterey, California, PhD Thesis, pp. 141-147.

Steinman, J. S. 1995. 'Scalable Parallel and distributed Military Simulations Using the SPEEDES Framework', California Institute of Technology, Jet Propulsion Laboratory, Pasadena, California.
ftp://pebbles.jpl.nasa.gov/pub/SPEEDES_Papers.

Whetten, B., Montgomery, T. & Kaplan, S. 1995. 'A High Performance Totally Ordered Multicast Protocol'. *Proceeding of the conference on Theory and Practice in Distributed Systems*, Springer Verlag, LCNS 938.
ftp://research.ivv.nasa.gov/pub/doc/RMP/RMP_dagstuhl.ps.