Third Irish Workshop on Computer Graphics (2002)
Eurographics Irish Chapter

Distributed Multi-User Urban Simulation

Clodagh Rossi and Carol O'Sullivan
Trinity College Dublin

Abstract

The objective of the research described in this paper is the importation, representation and 3D navigation of a large-scale city model. We explore the issues to be considered when designing distributed multi-user Virtual Environments (VE), including the co-ordination of multiple users and dead reckoning techniques. Our aim is to reduce network traffic and provide quicker response times with faster rendering.

1. Introduction

Virtual Environments aim to provide realistic impressions of life-like situations, giving potential users who traverse them a feeling of realism. To achieve this goal, however, there are a number of factors that must be taken into account. For example, in 3D rendering, providing a high Level of Detail (LOD) is important but if the graphics is too detailed the response times will be affected. Consequently, the graphics may be excellent but users will be disturbed by the length of time they must wait to move around the VE. Therefore, lower levels of complexity in representations of objects may be used and will result in faster display time. A major challenge is to find the correct trade-off that is acceptable to the user. Introducing multi-user functionality to the VE will further complicate the timing issue, as more users navigating the VE at any one time results in less time being allocated to deal with the request of each individual user, hence degrading response times. The design of the multi-tasking environment that allows for the multi-user activity is important. Different techniques and concepts can be introduced to help improve the response times. For example including *Dead Reckoning* can, as highlighted in the next section, help reduce network traffic by using prediction methods to predict where users will move to next, so that instead of each user having to send their exact position after every move they only do so when a certain distance between their actual and their predicted position has been reached.

2. Previous Research

2.1. Dead Reckoning

There are three primary criteria that should be met when constructing networked VEs. Response times (latency) must be kept to a minimum, network bandwidth usage must be minimised and consistency must be maintained between all nodes [4]. To achieve consistency, entities send their current relevant details e.g. the position they are currently at along with their direction and speed, to other entities in the system. Every time an entity updates its criteria it must send the update to the other members of the system.

As a network becomes more complicated, maintaining more connections and managing more communication routes between computers, the chances increase that a message may get mis-routed, delayed or inadvertently consumed by the routing mechanism [5]. Also, when the number of messages on the network increases, the chances increase that two messages get sent at the same time and collide, garbling both.

Sometimes the network has an acceptable amount of communication traffic, but then a sudden surge of messages momentarily flooding the network could affect two major factors in this type of environment, consistency and communication costs [10]. This causes problems with network latency and bandwidth, as the network is unable to cope with the high stream of data being sent, and results in lost packets not reaching their destination. Therefore, inconsistencies in the VE will be displayed to the end-users. These problems will obviously affect the realistic nature of the VE and users' overall impressions. For example, a lost or delayed message containing positional information could lead to users seeing objects or people "jumping" across their screens instead of moving in a smoothly progressive and realistic manner.

To reduce the number of connections and the number of messages being sent, the dead reckoning technique may be employed. Dead Reckoning is a form of replicated computing in that everyone participating in a multi-user system

must simulate all the entities and users in the environment. A predefined set of algorithms is used by all entity nodes to extrapolate the behaviour of entities in the game, and an agreement on how far reality should be allowed to diverge from predicted behaviour before a correction is issued [2].

When an entity is created, the computer that owns the entity sends out information about itself. The packet of data contains information to describe its current state, *e.g.* its unique identifier, position, velocity, acceleration and orientation. Upon receipt of the first entity state packet, all other nodes on the network begin by using the pre-defined algorithms to predict the next movements of the entity based on its last known velocity - a process known as Dead Reckoning. As long as the entity continues to move in a predictable fashion, it appears in a consistent, synchronized way on all nodes on the net with no further network traffic required.

To cater for cases when the entity does not move as predicted, each entity must retain its last state update message [4] sent out to the net, as well as having control of its 'live' object. This *ghost* object's position, so-called since the object is actually controlled by another process [5], will be updated by being passed through a simulation loop which uses the dead reckoning algorithm employed by all entities in the system. This results in the entity having the values for its own predicted position used by all other entities as well as its actual current position. The two values are compared and if the difference is significant *i.e.* by an amount that exceeds the agreed-upon threshold, a new packet is sent out to the other nodes on the network. The new state packet is now used by the entity to calculate others' predictions of its movements.

The following algorithms provide an overview of both actual and proposed techniques, which could be used to provide dead reckoning in a VE:

2.1.1. Case 1

Dead reckoning is at the heart of popular simulation mechanisms, such as DIS (the Distributed Interactive Simulation protocol)[6], and is used in SIMNET [11] and NPSNET [8].

In both SIMNET and DIS, no central computer is used for event scheduling and each host independently maintains its own state. A broadcast communication model is used in conjunction with a homogeneous world database. Entities interact through a series of events. Entities are self-ruling and all events are broadcast and are available to all interested entities. An entity initiating an event does not calculate which other entities might be interested or how the receiving entities can be affected by it and each entity transmits the absolute truth about its state. This state is commonly referred to as *ground truth* information. The receiving entities are responsible for transforming the ground-truth information to model the real world. A host in the SIMNET and DIS model can only know what it is told. A constant update of position would consume a lot of network bandwidth so a dead reckoning algorithm is used.

The architecture adopted by NPSNET has evolved from SIMNET and DIS, and embodies the *players and ghost* paradigm. In this paradigm, each object is controlled on its own host workstation, by a software object called a player. On every other workstation in the network, a version of the player is dynamically modelled as a ghost object. The ghost objects on each workstation update their position through a simulation loop using a dead-reckoning algorithm. The player tracks both its actual position and the predicted position calculated with dead reckoning. The core simulator communicates to the network via a protocol converter interface that sends and receives network packets asynchronously using both a *send thread* and a *receive thread*. This allows the graphics display rate to be maintained while data is read or written in separate lightweight processes [9].

2.1.2. Case 2

Aronson (1997)[2] proposed a method where the entities position, velocity and acceleration are used to extrapolate the entity forward from its initial position at time $t0$. The first algorithm maintains an entity at the position specified in the entity's state from $t0$. The second algorithm extrapolates the entity forward from its known t0 position based on its velocity at $t0$. The third algorithm also extrapolates forward from the entity's last known position, but uses both velocity and acceleration in the extrapolation:

$$\begin{aligned} Position_{t1} &= Position_{t0} \\ Position_{t1} &= Position_{t0} + v^r(t1-t0) \\ Position_{t1} &= Position_{t0} + v^r(t1-t0) + 1/2a^r(t1-t0) \end{aligned}$$

2.1.3. Case 3

The same basic approach is used by Cai et al. (1999)[3]. However, as well as focusing on eliminating the transmission of irrelevant packets, they aim to reduce the number of state update packets sent by the dead reckoning mechanism. They note how most dead reckoning algorithms have fixed threshold levels regardless of the distance between the two entities and how this is not necessary. The larger the distance, the fewer updates are required *i.e.* the threshold is dynamically changed according to the relative distances between the simulation entities.

The extrapolation equations are shown in Table 1. One-step formulae use the last state update packet to extrapolate an entity's position, whereas multi-step formulae use the last two or more state update packets in the extrapolation.

The adaptive multi-level threshold uses relevance filtering to determine the levels of threshold. The Area of Interest (AOI) of an entity is defined as a circle with a constant radius around the entity, the length of which is defined according to the entity type. Also used is the Sensitive Region (SR): if one entity moves into another entity's SR, a collision is likely to happen. Figure 1 shows how both the AOI and SR are used to determine the level of threshold.

In all cases, extrapolation of A needs to be accurate,

	One-Step	Two-Step
1^{st} *Order*	$x_t = x_{t'} + v_{t'}T$	$x_t = x_{t'} + (x_{t'} - x_{t''}/(t' - t'')T$
1^{st} *Order*	$x_t = x_{t'} + v_{t'}T + 0.5a_{t'}T^2$	$x_t = x_{t'} + v_{t'}T + 0.5(v_{t'} - v_{t''}/(t' - t'')T^2$

$x_t = position, v_{t'} = velocity, a_{t'} = acceleration$
$T =$ Time elapsed from the last update, $t = t' + T$

Table 1: *Extrapolation equations*

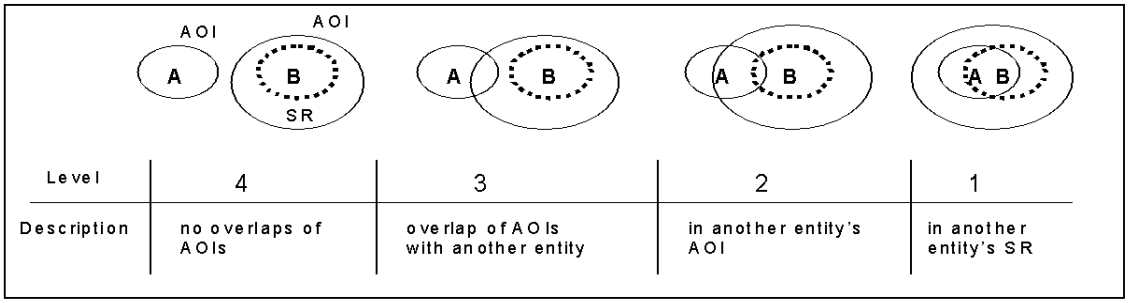

Figure 1: *The Area of Interest (AOI) and the Sensitive Region (SR) are shown for two entities*

though the degree of accuracy required is different. For example, in Level 1, A is in entity B's SR and to prevent entity B from making misjudgement on collision, entity A's update packet will be emitted most frequently, but its extrapolation will be the most accurate. Level 2 is where A is partly outside B's AOI, so to avoid missing the detection of an approaching entity, entity B needs to have a more accurate position of entity A. However there is no danger of a misjudgement on collision so a small threshold will be adequate in entity A's extrapolation. Level 3 is where A is outside B's AOI but inside its SR, so one entity may move to another's AOI in a short period of time. Extrapolation of A's position still needs to be accurate but the requirement is less rigid. As A is not in B's AOI, as in Level 4, its update packets do not need to be sent frequently, so a large threshold can be used in A's extrapolation.

2.2. Distributing Multi-User Virtual Environments

When designing distributed multi-user Virtual Environments (VE), certain key issues must be taken into consideration to achieve the end goal of providing a life-like simulation on screen. One of these involves providing a single-system image that hides any distribution from users. The user shouldn't know or care where resources are located. For example, when using a distributed database, the objects displayed may not be located on a particular user's machine, but obtaining this data from another location should not affect response times, with the user waiting extra periods while the new positional information is being sent across the network. Another issue would be enabling multiple users to share resources automatically instead of having a situation where they are queuing for a request to be processed while other users are being dealt with before them. The VE must also be consistent and reliable *i.e.* all users should observe the same outcome at relatively the same time. A good communication model, and organized well-adapted approaches to distributing the system will aid in achieving these goals [9].

2.2.1. Communication Models

The communication techniques adopted by multi-user VEs fall into three categories: point-to-point, broadcast, and multicast.

Point-to-Point One connection for each host is established and each host communicates with all other hosts in the system. This approach falls apart, however, when communication is scaled, as it requires n^2 messages, where n is the number of hosts in the network.

Broadcast This communication protocol allows each host on the network to send a single message that is received by all other hosts in the system, regardless whether they need the data or not.

Multicasting This involves hosts joining multicast groups. Data is sent to one or more multicast addresses and if a host is part of the multicast address it will receive the data.

2.2.2. Distribution Schemes

All distributed VEs share the problem of where to keep the VE database on the network and how to represent it. The

following list provides a number of options which could be adopted:

Replicated Database: Each host maintains a full of copy of the world database. Each host is responsible for the maintenance and upkeep of its internal database must reflect any changes made. The originator of any change to the database must send a message to the other hosts on the network informing them of the change.

Centralised Database A single server host is responsible for communicating with each of the clients to determine and report the current state of the system. The server maintains the database while the clients handle computation and rendering and present the users with a view of the VE.

Partitioned Database The database is partitioned among clients and a server is used to communicate the database between the relevant clients.

Shared Distributed Database Basically a shared memory system, where each new entity which is created or modified is distributed through the shared memory model.

2.3. Polygonal Meshes

In a VE the graphical detail displayed to the user is one of the critical features that will affect their overall impression of the environment. To provide a feeling of realism, realistic representations must be presented to the user.

Polygonal Meshes currently dominate the field of interactive three-dimensional computer graphics due to their mathematical simplicity that allows fast rendering of polygonal data sets. The following list outlines the main areas involved in gathering data and processing it into polygonal mesh format [7]:

Data Acquisition The first task is to retrieve the relevant properties of the objects, such as its geometry, surface texture, volumetric density information.

Mesh Generation This step takes the data retrieved in the first step and generates a single surface representation, *e.g.* a triangle mesh.

Mesh Decimation Reduces the complexity of a given input mesh by removal of detail information i.e. vertices and triangles

Although seen as one of the best methods for rendering objects in real-time, this process could still affect response times if a large number of objects are to be displayed on screen simultaneously or the level of detail is such that a high number of calculations are required to display them. Culling techniques can further reduce the computational cost of rendering polygonal objects, a good approach being: "Do not even attempt to render any geometry that the user will not ultimately see"[1].

3. Analysis and Design

3.1. Dead Reckoning Algorithm Chosen

We have developed a simple multi-user urban simulation that has a central server to which all entities connect. The entity upon connection to the server sends its state packet to the server, where the server assigns a unique id to the entity. The server then sends to the new entity the server's state packet and all state packets for the other entities. At the same time it sends the state packet for the new entity to all other nodes, which will recognise this as a new entity by its unique id.

Acceleration and speed are not taken into consideration in this prediction algorithm as entities move at a fixed rate, but the direction that the entity is moving must be sent by the entity so that other entities can predict the axis to move along *i.e.* if the entity is going forwards or backwards or to the right or left. The state packet is made up of the following data:

Entity Direction
Entity X Position
Entity Y Position
Entity Z Position

Once the state packet is sent and there is on-screen movement, the prediction algorithm comes into play. The entity after each actual move will also calculate its ghost position *i.e.* the position calculated from the last state packet sent which all other nodes will use to predict this entity's progress. The distance between these two points will then be found using the following equation, where c is the current value and p is the predicted value:

$$Dist = \sqrt{(X_c - X_p)^2 + (Y_c - Y_p)^2 + (Z_c - Z_p)^2}$$

If the distance is greater than the fixed threshold level, the entity must resend its state packet to the server for redistribution among the other nodes. A flag is set to indicate to the thread responsible for sending and receiving data from the server that a new state packet is to be sent and the new *last position sent* is recorded to calculate the ghost position.

The server has a client thread for each entity to communicate with the linked entity. Associated with each entity is a flag, which is set to true once a new state packet has been received for this entity. Once set, the packet is sent to all nodes and reset to false until another packet is received for the entity.

Two methods have been considered for the distribution of the new state packets. The first involves the server having two threads per client, one for receiving the state packet for this entity and one to send all other updated state packets in the system to the entity. This is due to the random sending of data across the network *i.e.* it can't be pre-determined when new state packets will be sent and will therefore not be sent in a pre-defined order. The other method would involve the server having one thread per client and then another thread, which distributes new state packets to all users when they are received.

The former method was chosen due to the unpredictable timing of state packets being sent by the entities.

3.2. Server and Client Design

The server is responsible for communicating all client activities to other clients in the system *i.e.* clients do not communicate with each other directly but through the server. To cater for these multiple tasks the server must be multi-threaded. The main thread deals with displaying graphics and user I/O. It also determines sector positions: To display all potentially visible objects on-screen results in very slow response times, so the environment is divided into blocks or sectors relative to a user's current position. As a user moves from one sector to another, different sectors are displayed around them. A listening thread must continuously wait at a predefined socket for new clients requesting to log on. When a new request is received, an individual client thread is created to deal with this client. Hence the more clients logged on, the more simultaneous threads are running.

The client also has multiple activities that need to be carried out simultaneously. These would include similar graphical responsibilities to the server, using the dead reckoning algorithms to determine if their position needs to be sent again and receiving other clients' positional information from the server. The functionality of both the server and the client is shown in Figures 2 and 3.

4. Evaluation and Future Work

The multi-user nature of this project means that all users must know and be able to see the exact location of all other users in the environment. A database provides polygonal mesh data and removes the need for replicated data. A multi-tasking environment allows for the required multi-user activities and dead reckoning reduces the network traffic required, contributing towards quicker response times and more efficient rendering.

The design of the server involves the server carrying out graphics rendering and catering for user input and output. Tests showed that this hindered the servers' capabilities of dealing with clients' requests and contributed towards delays on screens of up to 3 seconds. When the rendering was removed, this delay was negligible which strongly supports the theory that this activity should be removed from the server thus improving the overall multi-user interaction and communication.

The greater the number of objects displayed, the more time is required to draw each frame and therefore the slower the response times for all other activities. The solution to this problem is to divide the world into sectors and assign buildings to sectors depending on their position. The user's current camera position is used to identify the sectors that are directly around the user and that need to be drawn thus improving the response times. However, sectoring leads to *popping* affects on the screen when the user moves from one sector to another. This results in different sectors being drawn so that buildings seem to pop onto the screen out of nowhere. To reduce the popping affect so that it is less obvious means either drawing more sectors or increasing the size of sectors so that the objects popping up are too far away from the user's position to be noticed. This solution however again increases the rendering required and so is going to affect the response times.

A solution to achieving the same response times but displaying more objects on screen could be to draw more sectors but reduce the level of detail required on objects or buildings that are further away to the user. This would result in the buildings being visible but less rendering is required and so less time is needed to draw them. Only the objects in the direct vicinity of the user require the highest level of detail. Perceptual techniques can also be used to ameliorate these effects.

The dead reckoning algorithm must be upgraded to include a user's velocity and acceleration, along with the introduction of a steering wheel and pedals. This will help to eliminate slight jumping affects, which can occur on screen as the clients move around the city. The jumping affect is the result of being unable to predict if a user is slowing down or speeding up and so being unable to predict if they are going to stop moving altogether.

References

1. D. Aliaga, J. Cohen, A. Wilson, H.Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, E. Baker, R. Bastos, M. Whitton, F. Brooks and D. Manocha. "A Framework for the Real-Time Walkthrough of Massive Models", *Technical Report: TR-98-013, University of North Carolina at Chapel Hill* (1998). 4

2. J. Aronson. "Dead Reckoning: Latency Hiding for Networked Games", *Gamasutra Website: http://www.gamasutra.com/features/19970919/aronson_01.htm* (1997). 2

3. W. Cai, F.B.S. Lee and L. Chen. "An Auto-Adaptive Dead Reckoning Algorithm for Distributed Interactive Simulation", *Proceedings of the 13th Workshop on Parallel and Distributed Simulation* pp 82– 89 (1999). 2

4. D. Crowley. "DVRML: Extending VRML for Multi-User Virtual Reality", *M.Sc. Thesis, Trinity College Dublin* (1999). 1, 2

5. R. Gossweiler, R.J. Laferriere, M.L. Keller and R. Pausch. "An Introductory Tutorial for Developing Multi-User Virtual Environments",*Presence***3**(4) pp255–264 (1994). 1, 2

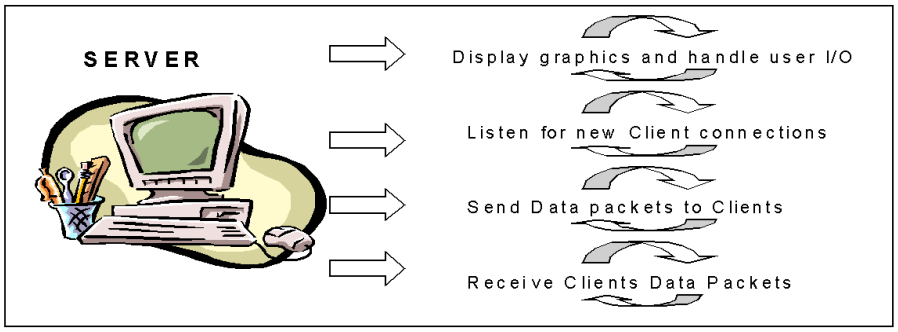

Figure 2: *The Server*

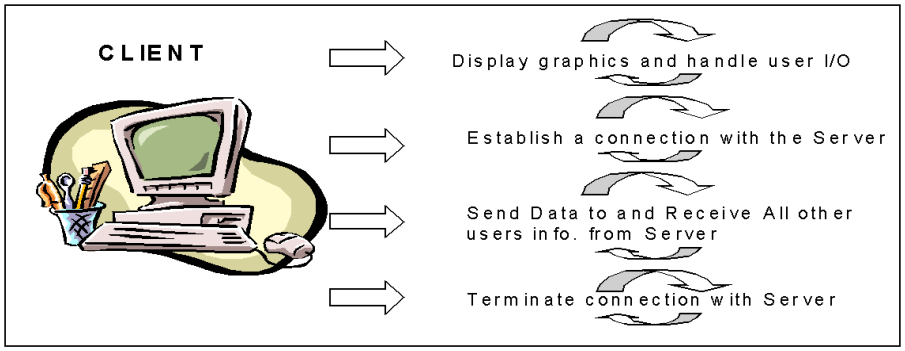

Figure 3: *The Client*

6. Institute of Electrical and Electronics Engineers, *International Standard, ANSI/IEEE Std* 1278 – 1993 (1993). 2

7. L. Kobbelt, S. Bischoff, K. Kähler, R. Schneider, M. Botsch, C. Rössl and J. Vorsatz. "Geometric Modeling Based on Polygonal Meshes", *Technical Report MPI-I-2000-4-0022000, Max-Planck-Institut für Informatik, Saarbrücken* (2000. 4

8. M.R. Macedonia, M.J. Zyda, D.R. Pratt, P.T. Barham and S. Zeswitz. "NPSNET: A Network Software Architecture for Large Scale Virtual Environments", *Presence***3**(4) pp 265–287 (1994). 2

9. K. O'Connell. "System Support for Distributed Multi-User Virtual Worlds", *Ph.D. Thesis, Trinity College Dublin* (1997). 2, 3

10. P.T.S. Tam. "Communication Cost Optimisation and Analysis in Distributed Virtual Environment", *Technical Report RM1026-TR98-0412, http://blackhole.cse.cuhk.edu.hk/paper/report/* (1998). 1

11. A. Pope. "The SIMNET Network and Protocols", *BBN Report No. 7102, BBN Systems and Technologies* (1989). 2

12. A.S. Tanenbaum. *Distributed Operating Systems* (1995).