Design and Implementation of Distributed Interactive Virtual Environment

CHAN Ming-fei

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science & Engineering

Supervised by:
Prof. John C.S.Lui

June 1999

Design and Implementation of Distributed Interactive Virtual Environment

submitted by

CHAN Ming-fei

for the degree of Master of Philosophy
at the Chinese University of Hong Kong in June 1999

Abstract

Large scale distributed virtual environment (DVE) systems model the activities of thousands of entities interacting in a virtual world simulated over wide area networks. These systems are growing to include more clients for applications such as multiplayer video games, military and industrial training, and collaborative engineering. In these applications, each host receives updates (such as position and orientation) from remote clients, models and renders the scene, and performs other tasks such as collision detection. The number of clients places a heavy burden on both the networking resources and computational resources available to the application. Today, how to meet the growing requirements of bandwidth and computational resources is one of the major challenges facing the design and implementation of large scale DVE system. A scalable DVE system usually employs many servers to handle clients' requests. The problem is how to allocate workloads among servers such that the computational load on each server is roughly the same, while communication cost is minimized.

In this thesis, we discuss the DVE scalability problem, briefly overview the major bandwidth reduction techniques and partitioning techniques currently being investigated and implemented in contemporary DVE systems, and propose an effective partitioning algorithm to solve the scalability problem.

The main idea of solving the scalability problem is to divide the virtual world into partitions and then assign these partitions to different servers. The server in the logical partition is responsible for maintaining all clients within its partition. We use

Hybrid approaches to partition the virtual world. The major approach is based on linear optimization technique.

Experiments are carried out to illustrate the effectiveness of the proposed partitioning algorithm under various settings of virtual world. The insight gained from our work and the challenges still facing the design of large scale DVE system are also discussed.

Acknowledgments

I have been incredibly lucky to have Professor John C.S. Lui as my supervisor. His warmth, enthusiasm, and clever insights have been inspirational. He has a unique intuition for what will and will not work, and his on-the-spot suggestions often saved me lots of fruitless works.

My work benefited greatly from cooperation and discussions with members of DVE research group, particularly Mr. Peter Tam and Mr. Oldfield So. They offer valuable suggestions to me. This thesis would not exist without the help of them.

It has been very pleasant to be a graduate student of Department of Computer Science and Engineering at the Chinese University of Hong Kong. The department provides a friendly research environment. Many people were always there when I needed help wading through any difficulty.

Finally, I would like to thank my family for having faith in me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Advances in multimedia systems, parallel/distributed database systems and high speed networking technologies enable system designers to build a distributed system that allows many users to virtually explore and interact under the same 3D virtual world. In general, a 3D virtual world is composed of many high-resolution 3D graphics sceneries that represent a real-life world. For example, we can have a 3D virtual world that represents a lecture hall with hundreds of students and scientists listening to a seminar given by Professor Daniel C. Tsui[1], or we can have a large 3D virtual world that represents the latest COMDEX show with thousand of attendants reviewing the latest softwares and electronic gadgets. This type of shared, computer-resident worlds are called the *distributed virtual environments* (DVEs)[22] and like other ground-breaking computer technologies, DVEs can change the way we learn, work, and interact with other people.

To illustrate how DVEs can change our lifestyles and the way we handle our business operation, we can consider the following situation. Let say that we have an architect from New York, a civil and structural engineer from Paris, a financial planner from Hong Kong and an interior designer from Tokyo, who are having a business meeting concerning about the design and financial planning issues of a new high-rise office complex to be built in London. Under the DVE setting, these people can convene a meeting while still reside at their respective homes/offices. Their meeting can be carried out through the DVE system, they can interact with each other in a virtual

[1] A 1998 Noble Prize winner in Physics for the discovery of a new form of quantum fluid with fractionally charged excitations.

world of the new high-rise office complex that they are proposing to build. Each participant in this business meeting can virtually *walk through* the proposed high-rise office building, interact and carry out the discussion without leaving their own offices. For example, in this virtual high-rise office complex, each user in the meeting is represented by a 3D object, which is known as an *avatar*, and each individual can walk around in this virtual office building, and in the process, rearrange any 3D objects in the environment (e.g., paintings, furniture, selecting different kinds of wall papers, ... etc). Any change to a 3D object in this virtual world will be visible to all participants. Also, during the meeting in this virtual world, participants will be able to interact with each other in real-time, as well as inquiring any relevant information about the virtual world that they are exploring. For example, querying about the credit information of a manufacturer who is responsible to produce the office furniture.

1.1 Challenging Issues

There are many challenging issues to design a cost-effective, scalable and high performance DVE system. In what follows, we list some of the important research issues (although not exhaustive) in the design of such kind of DVE system.

- *Backend Database:* Designing a spatial database engine so that users can virtually explore a huge 3D environment and at the same time, able to query and retrieve relevant information about the environment being explored. The type of queries that being supported are of relational, spatial, and possible fuzzy types. This research issue has been addressed in the VINCENT project[3], which is the earlier version of our current DVE system.

- *Object Consistency:* Since DVE clients can manipulate any object in the 3D virtual world (e.g., a user may want to pick up a book), therefore, it is important to keep all these 3D objects in a consistent state such that once the object is being accessed, other users may not be able to access it anymore. In general, there are several approaches to solve this concurrent access problem, either by exclusive locking of the object, or by defining various operations (similar to defining the read/write operations in database) that can be performed on the object so as

to allow many users to concurrently access the object. Although concurrency control has been well studied in the database research community[15], concurrent data access under a DVE environment is more complicated because each object is rich in semantics and therefore, high degree of concurrency control algorithm can be defined.

- *View Consistency:* Since users can move around in the virtual world and any user can access any object in the environment, therefore, all users under the same virtual world have to be notified of the activities so as to keep their local views consistent. For example, if a user moves a chair from one location to another location, another user in the virtual world has to have this activity visually displayed on his/her local screen. The propagation of this information to all clients demands a tremendous communication bandwidth. Recent research work on multicasting techniques[9, 16, 17] can help to reduce the aggregate bandwidth demand on the network. In [4], the authors derive the optimal synchronization interval so that every client in the same virtual world can view all objects with a high degree of consistency.

- *Balancing Workload and Reducing Communication Cost[1]:* A virtual world may represent a large real-life environment and this require a large amount of computational power so as to render the realistic-looking 3D models at an interactive frame rate, detect and resolve object collisions, monitor user input devices and process their inputs. Each client/server also needs to process many updates sent by others so as to keep the states of every object in the virtual world consistent. The complexity of these tasks is increasing rapidly as DVE systems become more complex.

  As the number of clients in DVE system increases, the network traffic generated by these clients increases to an enormous level. The communication networks (WANs and LANs) supporting this exercise can easily become overwhelmed by the resulting traffic load and a portion of the transmitted packets can be lost. Furthermore, the increased computation overhead incurred at each client/server for processing the incoming information makes the scale up effort more difficult. It is clear that the DVE systems will have to employ multiple servers to handle clients' requests. Mapping the entities located in the same region to hosts

located in the same LAN and let the gateway host be the DVE server of that LAN can help to reduce update messages on networks. We can integrate other filtering techniques with this approach to further reduce both communication and computational demands. DVE systems based on a good mapping will have better intrinsic scalability. Therefore, an interesting and important problem in designing a DVE system is how one can *partition/map* the workload among different servers in the DVE system and at the same time, maintain a manageable level of communication overhead. This is the main focus and contribution of this thesis.

1.2 Previous Work

Let us briefly describe some previous work on DVE system. In [3], the authors illustrated how to design and implement a virtual walk-through system such that a user can query and retrieve information about the virtual world. One major limitation of the work is that it only allows a single user to explore the virtual world[2] and therefore, there is no communication and interaction between users. In [18, 19], the authors described how to build a storage system that can support applications like the video-on-demand and 3D walk-through of a virtual world. The result is particularly interesting in the sense that the storage server can guarantee the timely delivery of data to different multimedia applications, which may have vastly different quality of service. In [20], the authors demonstrated how to build a distributed virtual environment for military purpose and showed that it is possible to support hundreds of users. In [11], authors designed a prototype system for a large scale DVE that operates based on the Internet IP Protocol[12]. In this work, the authors illustrated that it is impossible for a single system to handle all the required workload and therefore, partitioning approach was mentioned. However, there is no detail description on how to maintain synchronization among partitions and how to carry out the partitioning operation with the consideration of communication cost. In [21], the authors provided a software toolkit known as DIVE so as to build a DVE system. In the DIVE toolkit, users can define

[2]Of course, the system allows many users to explore the same virtual world but under different sessions.

their own objects and behavior but the DIVE system assumes a single server, multiple clients architectures that each DIVE world is maintained by a dedicated server only. In [4], the authors derive the optimal synchronization interval so that every client in the same virtual world can view all objects with a high degree of consistency.

1.3 Organization of the Thesis

The thesis is organized as follows. Chapter 2 contains a description of the DVE system architectures, the avatar objects and their characteristics.

Chapter 3 surveys how existing DVE systems have solved the scalability problem, and the limitations of those approaches. Special emphasis is paid to research on bandwidth and computation reduction techniques.

Chapter 4 contains the formulation of the workload partitioning problem. We propose a partitioning algorithm to solve the scalability problem. The partitioning algorithm is based on the linear optimization technique and is shown to be computationally efficient and can effectively partition the workload evenly among the servers and at the same time, reduce the communication overhead.

Chapter 5 contains the result and analysis of various experiments with various sizes of virtual world and different avatar's location distributions to illustrate the effectiveness of our proposed partitioning algorithm.

Chapter 6 describes some considerations of implementing DVE systems and the insight gained from our work.

Lastly, conclusion and summary is given in Chapter 7.

Chapter 2

Distributed Virtual Environment

In this chapter, we describe various elements in a distributed virtual environment (DVE), namely, 1) possible architectures, 2) representations of clients as avatars, their area-of-interest (AOI) and, 3) the dynamic join and leave property of avatars in the DVE system.

2.1 Possible Architectures

To realize the multimedia service like the DVE application, one first has to consider the possible architecture. In general, there are two possible architectures for implementing a DVE system. The choice of which architecture to use depends on the size of the virtual world (or the 3D environment) that we want to model as well as the number of concurrent clients. These two types of architectures are 1) the single server distributed virtual environment architecture (SSDVE) and 2) the multiple servers distributed virtual environment architecture (MSDVE).

In the SSDVE architecture, all clients are connected to the dedicated server. To make sure that all clients have the same consistent view of the virtual world, any action or activity generated by a client has to be transmitted to other clients[1] in real time. This form of communication is accomplished as follows. The initiating client will send a message to the DVE server, the DVE server will first transform the message to the

[1] In general, the information is sent to all other clients or only a subset of the clients in the system.

corresponding database operations (e.g., locking an object in the virtual world) and then the server will broadcast (or multicast) the new information to other clients in the DVE system so that every client can update their local view of the virtual world. It is important to point out that, experience has shown that the SSDVE approach does not scale well. If the number of clients is large, then the demand on the processing power, system buffer and communication bandwidth will be a serious problem. Therefore, SSDVE is only suitable for small scale DVE system, for example, the virtual world with small number of objects and the number of clients is small.

To support a large number of concurrent clients in a DVE system, one can adopt the MSDVE architecture. In the MSDVE architecture, multiple servers will be used and each server is responsible to handle a subset of the virtual world (e.g., some number of clients and some number of objects in the virtual environment), the communication of its attached clients as well as the communication between servers. It is important to point out that in order to keep the view consistency among the clients, it is unavoidable to have server-to-server communication and that the communication cost between servers is more *expensive* than the communication cost between the server and its attached clients. This is due to the fact that the server and its attached clients may reside in the same local area network while different servers may reside at two extremes edges in the Internet. Figure 2.1 illustrates that we use three servers to divide up the virtual world. Clients are *attached* to a specific server whenever the client is in the administrative region of that server.

2.2 Representations of Clients as Avatars

In a distributed virtual environment, we usually use an avatar, which is a 3D active object, to represent a DVE client in the virtual world. In order to provide the interactive capability of a client, the avatar can move or travel in the virtual world. The client can also use his/her avatar to communicate with other avatars (or other users in the virtual world), or use his/her avatar to access any 3D objects, such as books, chairs, glass, ... etc, in the virtual environment. Since an avatar can move around and can interact with any static or dynamic 3D objects within the virtual world, any action that is performed by an avatar may require the DVE system to relate this information

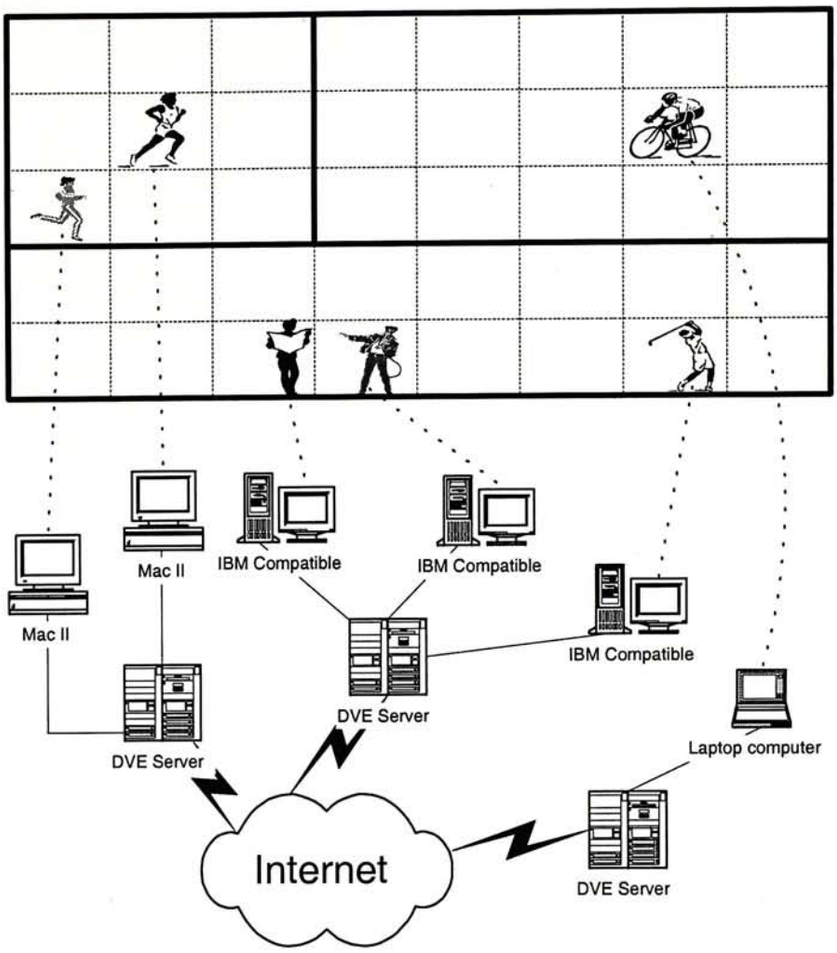


Figure 2.1: MSDVE architecture for our DVE system

to other avatars so as to keep the information of the virtual world consistent.

One simple way to maintain the consistency of the virtual world is to broadcast any action taken by any avatar to all other avatars in the system. However, this will incur a significant communication overhead. In general, each avatar only needs to know the activities that happened near his/her vicinity, for example, any activity that is within 10 meters of his/her position in the virtual world. Therefore, one way to significantly reduce the total communication overhead is to allow every avatar to define his/her own *area of interest* (AOI). In general, AOI is the region of the virtual world that if there is any activity happened in this region, the avatar needs to know so as to update it's own state and to make his/her view consistent. Figure 2.2 illustrates the concept[2] of AOI. In this figure, we have three avatars, A1, A2 and A3. Since the AOI of A1 and

[2]Without the loss of generality, we use a circle to represent the AOI of each avatar.

A2 intersects, therefore, any activity that happened in the intersection region can be seen by both A1 and A2. On the other hand, the AOI of A3 does not cover any other avatars. Therefore, the server does not have to inform A3 about any activity that was generated by avatars A1 or A2.

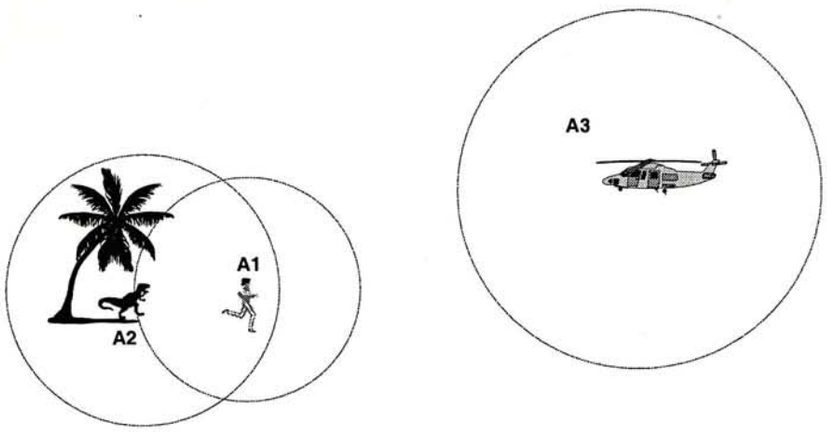


Figure 2.2: Avatars and their area of interest

2.3 Dynamic Membership

It is important to point out the dynamic characteristic of avatar in joining and leaving the virtual world. The dynamic characteristic of avatar in joining and leaving the virtual world refer to the fact the number of clients who are exploring a virtual world *may* be dynamically changing in time. For example, for the virtual world of business meeting we described in Chapter 1, the number of clients is *fixed* through the virtual world application (e.g. people are conducting a business meeting over the DVE system). On the other hand, the virtual world of the COMDEX show we described in Chapter 1, the number of clients can vary in time since a user may want to logon to the DVE system and explore the COMDEX virtual world anytime, or a user may decide to leave the COMDEX virtual world when he/she found the electronic gadget in mind.

This dynamic characteristic of avatar in joining and leaving the virtual world increase the necessity of an efficient partitioning algorithm. Moreover, Since an avatar can move from one location to another, it is very possible that an avatar can move out of the region that is managed by a server, say S_i, and move into another region that is managed by another server S_j where $i \neq j$. It is easy to observe that if we do not ad-

just the avatar-to-server assignment, eventually, the workload among servers will vary significantly and the amount of traffic[3] between servers may reach an unacceptable level. Therefore, it is important to find an efficient algorithm that can partition the workload in the virtual world *evenly* so that every server will carry the same amount of workload and at the same time, minimize the server-to-server communication. The partitioning problem will be explored in Chapter 3.

[3]The traffic between servers is to keep the view of every avatar in the system consistent.

Chapter 3

Bandwidth and Computation Reduction Techniques

Distributed Virtual Environment systems have adopted widely disparate approaches for disseminating information about entity motion and modeling those entities at remote hosts. The broad range of techniques reflects the relative lack of experience in developing such systems. In this chapter, we discuss techniques used to reduce the bandwidth and computational demands of large scale DVE systems.

Given the limitation of hardware, we must seek to reduce the bandwidth and computational demands of DVE systems without introducing additional latency for information dissemination. Three basic approaches are available:

- Transmit less information about each entity and/or transmit entity updates less frequently.
- Limit the number of entities that are of interest to each host.
- Have each packet provide information about multiple entities.

In following sections, we describe the major techniques based on the above approaches.

3.1 Network Communication

Three distribution methods [23] are shown in Figure 3.1. Multicast services allow arbitrarily-sized groups to communicate on a network via a single transmission by the source. Multicast allows a host to send data simultaneously to a set of (but not necessarily all) locations. With broadcast, data is sent to all hosts while unicast establishes communication between two hosts.

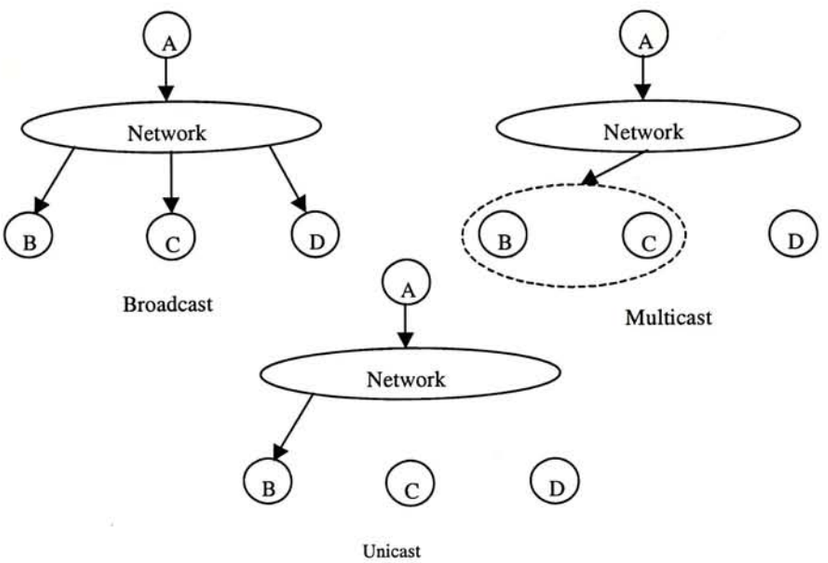


Figure 3.1: Examples of broadcast, multicast and unicast

Most DVE systems have employed some form of broadcast or unicast. For example, broadcast is used in the earliest military SIMulator NETwork (SIMNET)[25] and unicast is used in the Distributed Interactive Virtual Environment (DIVE)[26] system. NPSNET-IV [27] is the first DVE application to use the IP multicast protocol.

Broadcast is not appropriate for DVE systems because the network becomes flooded with unwanted traffic and it is difficult to avoid routing loops. Moreover, IP broadcast requires that all hosts examine a packet even if the information is not intended for that host, incurring a major performance penalty for that host because it must interrupt operations in order to perform this task at the operating system level. As the number of simulator increases, the traffic generated by these simulators increases to an enormous level. The resulting traffic load can easily overwhelm the communication networks and a portion of the transmitted packets can be lost.

Unicast requires the establishment of a connection or path from each node to every other node in the network for a total of $N * (N - 1)$ virtual connections in a group. It is also not appropriate for DVE.

A general problem of using multicasting is that the IP Multicast protocol is neither reliable nor order preserving. Thus packets might get lost, be duplicated or arrive in different orders. Communications reliability often forces a compromise between bandwidth and latency. Reliable multicast protocols are currently not practical for large groups because in order to guarantee that a packet is properly received at every host in the group, an acknowledgment and retransmission scheme is required. With a large distributed simulation, reliability, e.g., as provided in TCP, would penalize real-time performance merely by having to maintain timers for each host's acknowledgment and by holding up flow when a packet is lost for retransmission. Flow control introduces delay to the network to reduce congestion. Therefore, it is also not appropriate for DVE which can recover from a lost packet more gracefully than from late arrivals - it is impossible for real-time simulations to go backward in time.

3.2 Dead Reckoning

Dead reckoning [10] is used to reduce the number of state update messages that need to be transmitted by each simulator for the purpose of maintaining accurate state representation. The basic idea is to predict the trajectory of a simulation entity based on its speed and orientation. In a dead reckoning system, each simulator has a high fidelity model which maintains accurate information (position, velocity, orientation, etc.) of the state of its own entity. Each simulator also maintains a less accurate model, called the dead-reckoning model, for each entity participating in the simulation. The high fidelity model of an entity provides the exact position/orientation of that entity. The corresponding low fidelity model provides the dead-reckoned (inaccurate) position/orientation of the entity. When the state of an entity changes, the simulator of this entity updates its high fidelity model and compares it with the corresponding dead-reckoning model. If the entity's dead-reckoned position/orientation has deviated from the exact position/orientation by more than a threshold value, the simulator of that entity issues a new message to communicate to other hosts the actual information

of its entity. When any simulator receives a update message for one of the dead-reckoned entities, it corrects the dead-reckoning model for that entity and begins a new extrapolation based on the new information of the received message.

The earliest implementation of dead reckoning protocol in DVE is the Amaze multi-player game. In this game, every host transmits position (x_0) and velocity (v) updates about local player once per second. The dead reckoned position at time t is given by:

$$x(t) = x_0 + v * t$$

The SIMulation NETworking (SIMNET) system moves away from the fixed-rate update approach used by Amaze. The host transmits an update packet either when the true and dead reckoned models differ by some error threshold or when no update has been otherwise within a five second timeout period. The Distributed Interactive Simulation (DIS) protocol, IEEE standard 1278, is similar to the SIMNET protocol in most respect, though its update packets also include acceleration, orientation, and angular velocity information. The dead reckoned position at time t is given by:

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

Some studies concluded that second-order dead-reckoning is the recommended mechanism since it gives a good balance between accuracy and complexity.

The current state-of-the-art in dead reckoning algorithm raises several limitations [13]. First, all existing protocols are tightly coupled to their underlying network environment. Most systems have been designed for use over a LAN providing high reliability and predictable latency characteristics. Even the DIS protocol design, targeted for WAN, does not directly address the variable performance of long-haul communication networks. Second, existing simulation protocols do not accommodate the variable modeling fidelity needs at each remote simulation host but instead associate a single dead reckoning error threshold with the source transmissions. As simulations contain increasing numbers of entities, hosts cannot afford to model all entities in full detail. Instead, the simulation needs to support a continuum from low-fidelity modeling to high-fidelity modeling so that individual host can select the appropriate level-of-detail

based on local requirements. Finally, analyses of dead reckoning protocol behavior have concentrated almost exclusively on single entity types. These analyses do not offer a general-purpose technique for assessing the protocol's behavior over more general entity motion.

3.3 Message Aggregation

Message aggregation [24] attempts to merge a group of entity updates into an update packet, thereby reducing packet header overhead in the network and reducing packet-processing overhead at receivers. This approach is a viable way to reducing bandwidth and computational load, yet it must be designed with care. Message aggregation means that the earlier packets will have to wait for the arrival of additional packets to combine into larger packets. If this waiting period becomes too large, the earlier packets in the bundle would become too old and transmitting them would be no better than discarding them. The size of the bundle must therefore be limited by taking into consideration the maximum end-to-end delay permitted in the DVE system as well as by the maximum packet size allowed by the network protocol. A key challenge in message aggregation is determining which entities to group together. Three approaches have been used in previous distributed simulation systems: network-based, organization-based, and grid-based.

3.3.1 Network-Based Aggregation

Network-Based Aggregations group simulation entities by their physical location in the network. This approach is best suited for environment in which the wide-area network or network tail-circuits represent the primary bandwidth bottleneck. It is most beneficial only when there is some correspondence between the entity locations in the virtual world and their physical locations. However, entities on a LAN need not share any relationship to one another, either in terms of entity type or entity location within the virtual world. A receiver who subscribes to the aggregation would typically receive a considerable volume of information from entities that are of no local interest.

3.3.2 Organization-Based Aggregations

Organization-Based Aggregations groups simulation entities by their organizational hierarchy. Although it is easy to construct and maintain, this approach offers limited value because each organization's member entities may travel within different regions of the virtual world. For common operations such as collision detection and scene rendering, each host wants data about all entities located within a nearby region of the virtual world. If only organization-based aggregations are available, the host must subscribe to information from all organizations represented within that region, even though most of the organizations' member entities may actually be far from the viewer. Consequently, organization-based aggregations are most beneficial only when there is some correspondence between the static entity organization and the dynamic entity location within the virtual world.

3.3.3 Grid-Based Aggregations

Grid-Based Aggregations [28] group simulation entities by their location within the virtual world. The virtual world is divided into rectilinear or hexagonal grids whose associated aggregation transmits packets bundling information about entities in that region. Most existing implementation of grid aggregations dispense with a designated aggregation entity and instead simply associate a multicast address to each grid. Each entity transmits updates to the multicast group associated with its current virtual world location, so although the data is not bundled into the same packet, the multicast group allows remote hosts to select the virtual world region of interest. Grid-based aggregations pose several disadvantages. They mask the organizational relationships between the various entities. Moreover, establishing an optimal grid size for use by all simulation hosts is difficult because the ideal grid size depends on the amount of inter-host interaction in the simulation scenario and on the number of entities running on each host.

3.4 Relevance Filtering

Large-scale distributed simulations model the activities of thousands of entities interacting in a virtual environment simulated over wide area networks. Originally these systems used protocols which dictated that all entities broadcast messages about all activities, including remaining immobile or inactive, to all other entities, resulting in explosion of incoming messages for all entities, most of which are of no interest.

Relevance filtering works by entirely eliminating the transmission of irrelevant packets. Specifically, relevance filtering refers to the process of analyzing the semantic contents of packets and selecting only the ones that meet certain criterion. Relevance filtering can be effectively used to overcome the restricted bandwidth of WANs. Filtering can also be used when the total traffic is large enough to overwhelm the small bandwidth of a local site or when the slow nodes in this site cannot handle the fast rate of message arrival.

3.4.1 Entity-Based Filtering

The filtering scheme uses a one-dimensional vector of distances for each simulated avatar. The vector is stored in the gateway of the LAN where the simulator of this avatar resides. Assuming that avatars in the simulated environment are numbered 1 through M, the vector for the ith avatar will be stored in the form

$$D_i = (d_{i1}, d_{i2}, ..., d_{ii} = 0, ..., d_{iM})$$

Where d_{ij} is the distance between avatar i and avatar j. We use *"area of interest"* to specific a neighborhood region such that the avatars located within that region are important to avatar i (e.g., they are visible to avatar i or can be affected by it). State update messages from avatars outside this reachability region need not be delivered to avatar i. Relevance filtering is based on the concept of distance computations. Filtering is performed by network gateways at the transmission and reception of a message. Filtering at transmission is the main process that could eliminate the majority of irrelevant messages. Filtering at reception performs a final check and eliminates any irrelevant messages that have not been detected during the transmission phase.

Algorithm Filter-at-Reception

/* This algorithm is executed in the gateway of site s, $1 \leq s \leq N$ */
/* Nodes in site s are numbered from $M_{s-1}+1$ to M_s */
loop
    wait for a new external packet
    denote this packet by E_k and let node k be its sender
    /* note that either $k < M_{s-1}+1$ or $k > M_s$ */
    $L = \emptyset$ /* L is a list of local nodes that should receiveE_k */
    /* update the position of object O_k */
    for $i = M_{s-1}+1$ to M_s do /* check all local nodes */
        update d_{ik} based on the contents of E_k
        if $d_{ik} \leq R_i$ then $L = L \cup \{i\}$ endif;
    endfor;
    if $L = \emptyset$ then discard E_k
        else send E_k locally to members of L endif;
endloop;

Algorithm Filter-at-Transmission

Loop
    Wait for a new local packet
    Denote this packet by E_i and let node i be its sender
    $G = \emptyset$ /* G is a list of gateways that should receive E_i */
    For $g \in \{1, 2, ..., N\} - \{s\}$ do /*check relevance of E_i for all other sites */
        Relevance = false;
        For $k = M_{g-1}+1$ to M_g do
            Recompute d_{ik} based on the contents of E_i
            If $d_{ik} \leq R_k$ then Relevance = true; exit inner loop; endif;
        Endfor;
        If Relevance = true then $G = G \cup \{g\}$endif
    Endfor;
    If $G = \emptyset$ then discard E_i
        Else send E_i to each gateway in G endif;
Endloop;

Figure 3.2 shows an example of filtering. Consider a state update message generated by a node in the first site(the sender node is denoted by S). Other local nodes in this site as well as gateway G_1 receive this message without much delay. Gateway G_1 then executes the filter-at-transmission algorithm. In the scenario shown in Figure 3.2, gateway G_1 determines that no node in the third site needs to receive the message. Gateway G_1 therefore sends the message to gateways G_2 and G_4 (but not to G_3). Each of gateways G_2 and G_4 executes the filter-at-reception algorithm and sends the message only to those local nodes that need to receive it(receiver nodes are denoted by R).

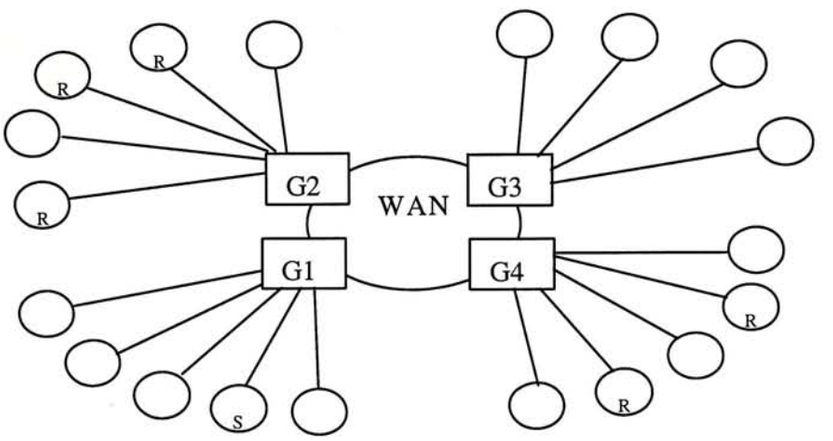


Figure 3.2: Filtering with four sites

Because of filtering, gateways may be deprived of receiving some critical packets from external simulators. This makes the information maintained by each gateway less accurate and can render their subsequent filtering decisions incorrect. We may use Gateway Dead Reckoning or Periodic Broadcast to reduce or eliminate filtering errors. We should notice that some entities simulated on the same site or LAN may far apart in the virtual environment while entities simulated far apart on the WAN may be very close in the virtual environment. This filtering approach filtered messages before they being sent from gateways, irrelevant messages on the WAN are reduced dramatically.

3.4.2 Grid-Based Filtering

Under grid-based filtering approach, the virtual world is decomposed into regions, each of which has a state server to maintain the states of the simulation entities and

environment objects in that region. The set of nodes in a region form a multicast group. Whenever a simulation entity changes its state, the update is sent to the multicast group associated with the region which the entity is currently in. When a simulation entity crosses region boundaries, it will join the multicast groups corresponding to the regions that are now within its sensing range, and leave those multicast groups that are associated with regions which become outside of its sensing range. When a node joins a multicast group, the corresponding region's environment state server will transport the current state of those entities in the region to the node to bring it up to date.

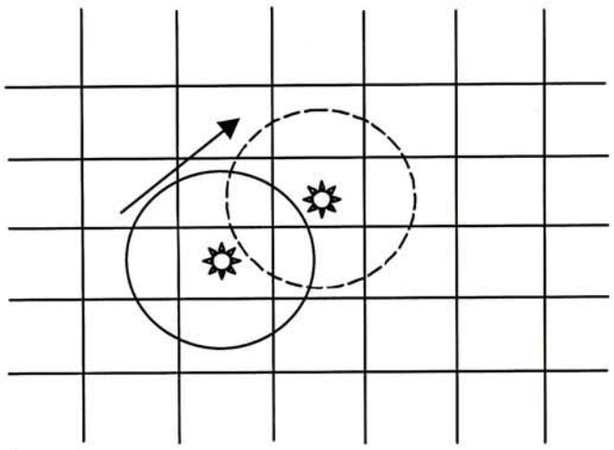


Figure 3.3: The virtual world is decomposed into rectangular cells

The entity-based filtering is generally more computationally expensive than its grid-based counterpart but is also a more efficient bandwidth reduction method. The grid system has the potential of delivering irrelevant data to entities located outside the circular region of interest but inside the grids covering this circular region. There is a tradeoff in the choice of the grid size. Small size grids result in better filtering efficiency at the cost of more overheads due to the management of a large number of multicast groups. In [33], the use of hexagonal grids (rather than square grids) is advocated on the ground that hexagons have uniform orientation and uniform adjacency and would better approximate a circular area of interest than squares.

3.5 Quiescent Entities

If an entity becomes totally stationary (zero velocity and acceleration) and does not move any articulated part, the dead-reckoning model implies that no state updates

would be emitted from this entity. Many DVE systems require such a quiescent entity to regularly emit its state at a low rate (e.g. once every 5 seconds). These redundant state updates, also called "keep-alive" messages or "time-out" packets, can comprise 70% of traffic in large scale DVE systems [33, 34]. One benefit of the keep-alive messages is that hosts joining an exercise in progress can correctly build their database and confirm the existence of all other entities participating in the exercise. To reduce the amount of extra traffic generated by stationary entities, we can consider two variable timeout schemes: a tiered-timeout approach and a exponential-backoff approach.

The tiered-timeout approach defines a small number of specific timeout levels that are selected explicitly by the entity based on its behavior. For example, when the entity becomes idle, it transmits an *timeout-length-announce* message, announcing that it will use a longer transmission timeout with its(less frequent) update packets. When the entity's behavior becomes more dynamic, a corresponding *timeout-length-announce* message restores the transmission timeout to the original(short) duration.

Using the exponential-backoff approach, the entity's timeout value is determined implicitly based on its recent update behavior. Under this approach, the entity applies exponential backoff to increase the transmission timeout if it has not transmitted any update packets within the previous timeout period. The timeout is bounded by some maximum value, and any position update transmission immediately restores the timeout to the original(short duration) value.

3.6 Spatial Partitioning

It's clearly not practical to download an entire virtual universe every time you enter it, nor could you store it all in memory on a normal computer system. Even if you could, the sheer complexity of the virtual world would bring the 3-D rendering subsystem to its knees. What we need to do is somehow partition the virtual world into smaller pieces. Only the pieces that we can actually see would be downloaded, stored in memory, and rendered.

3.6.1 Necessity of Spatial Partitioning

One of the main reasons VRML [5] browsers appear to be so slow, especially for indoor scenes such as building interiors, is that they must render everything in the entire world. If the user is in a small room next to a huge concert hall, the browser must still draw every single chair, every fold of curtain, every door and stairwell and banister in the concert hall - even though the user can not see any of those objects. This must be done for every single frame the computer renders. Therefore, the number of frames per second that can be generated is quite small. Low frame rates not only make the world less fun to explore, they can also make it nearly impossible to navigate though the environment.

The use of Level of Detail(LOD) nodes can help a little, but they operate strictly on the basis of distance and are therefore of limited use in this type of situation. More importantly, they are of little use in indoor scenes where there is a high level of depth complexity. Ideally, we never want to draw anything that the user cannot see. We can move closer to this goal by using a technique known as spatial partitioning.

Spatial partitioning works by taking a volume of space and subdividing it into smaller regions, then computing region-to-region visibility information. At any given time, only a subset of the overall environment needs to be visible to the user; this dramatically reduces the amount of rendering that must be done and, therefore, increases the frame rate.

There are a number of different approaches to partition a virtual world. The most important ones are bounding boxes, quad trees, octtrees, and BSP trees. Binary Space Partitioning (BSP)[32] trees are the approach most commonly used in computer games, such as Quake. Regardless of which approach is used, the basic technique is the same. On each frame, the user's viewpoint location is compared to a set of data structures in order to determine which region the user is in. Once this region is found, the set of visible regions is identified, and those regions that are not visible are flagged as being hidden. In VRML, this hiding is accomplished using a *Switch* node whose *whichChoice* evenIn is set to -1 for a hidden region.

3.6.2 Binary Space Partitioning Tree

A Binary Space Partitioning (BSP) tree is a data structure that represents a recursive, hierarchical subdivision of n-dimensional space into convex subspaces. BSP tree construction is a process which takes a subspace and partitions it by any hyperplane that intersects the interior of that subspace. The result is two new subspaces that can be further partitioned by recursive application of the method.

A "hyperplane" in n-dimensional space is an n-1 dimensional object which can be used to divide the space into two half-spaces. For example, in three-dimensional space, the "hyperplane" is a plane. In two-dimensional space, a line is used.

3.6.3 BSP Tree Construction

The BSP tree is constructed via a recursive algorithm. At each level, the process takes a subspace and partitions it by a selected hyperplane which intersects the interior of that subspace. This partitioning results in two new convex subspaces. The process can then be recursively called to further partition these two subspaces.

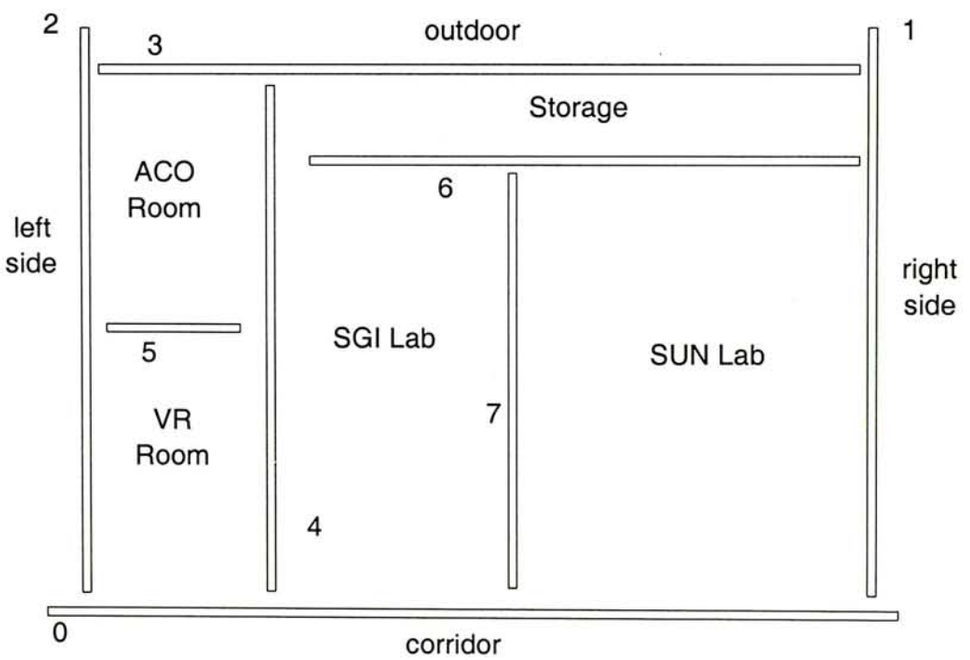


Figure 3.4: The floor plan with partitioning planes added

The algorithm can be summarized as follows:

1. Select a partition hyperplane.

2. Partition the set of objects with the hyperplane.

3. Recurse with each of the two new sets.

For example, let us look at a floor plan in Figure 3.4 . We add the planes that partition it into regions, and assign a number to each of those planes. The binary tree corresponding to this partitioning is shown in Figure 3.5.

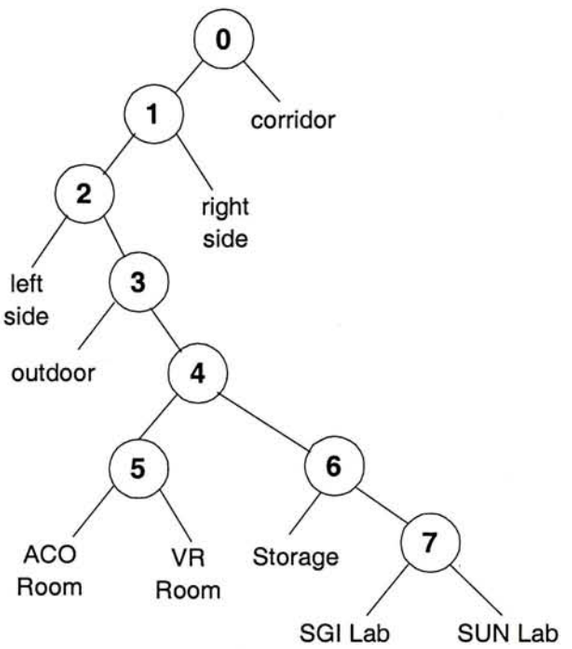


Figure 3.5: The BSP tree for the lab

By traversing the tree and always rendering the far side of each partition first, the correct drawing order is obtained. This avoids the need for Z-buffer testing, which is an expensive per-pixel operation. In VRML, we might use Switch nodes to hide regions that are not visible from the region that the user is in, so at least we do not waste time to render things the user can not see.

The use of spatial partitioning can not only lead to dramatic improvements in rendering performance, it can also be the basis for filtering of updates in a multi-user environment. For example, if a user is standing in the ACO room, they don't need to receive updates from someone standing in the SUN lab.

Chapter 4

Partitioning Algorithm

In this chapter, we present the partitioning problem of the DVE system. We first formulate the partitioning problem and illustrate that it is an NP-complete problem in general. We then present the iterative partitioning algorithm. The effectiveness of this algorithm will be illustrated in Chapter 5.

4.1 Problem Formulation

Let us define the following notation.

P	=	Number of partitions or servers in the DVE system.
n	=	Number of avatars in the DVE system.
a_i	=	Avatar i where $i = 1, 2, \ldots, n$.
$w(\cdot)$	=	A function that maps the processing of messages from an avatar to the computational cost of a server. For example, $w(a_i)$ represents the computational workload for avatar a_i.
$l(a, b)$	=	A function that maps the information exchange between avatars a and b to the network communication cost.
W_1	=	A non-negative relative weight to represent the computational workload cost on a server.
W_2	=	A non-negative relative weight to represent the importance of the server-to-server communication cost. Note that $W_1 + W_2 = 1.0$.
$C_{\mathcal{P}}^{W}$	=	Computation workload cost for a given partition configuration $\mathcal{P}$.
$C_{\mathcal{P}}^{L}$	=	Communication cost for a given partition configuration $\mathcal{P}$.

$C_{\mathcal{P}}$ = Total cost for a given partition configuration $\mathcal{P}$.

We can use a graph to represent the DVE system. Given a graph $G = (V, E)$ where V represents the set of avatars in the DVE system and E represents a set of edges so that $e_{ij} \in E$ represents that avatar a_i and a_j should communicate with each other. Let $\mathcal{P}$ be a partition that divides V into P (number of servers) disjoint subset $V_1, V_2, \ldots, V_P$ such that $V_i \cap V_j = \emptyset$ for $i \neq j$ and $\cup_{i=1}^{P} V_i = V$. In other words, all the avatars in the subset V_i will be assigned to the i^{th} server in the system.

Given a partition $\mathcal{P}$, we can define the workload cost $C_{\mathcal{P}}^{W}$ of this partition strategy as:

$$C_{\mathcal{P}}^{W} = \sum_{j=1}^{P} \left(\left| \sum_{a_i \in V_j} w(a_i) - w^* \right| \right) \tag{4.1}$$

where $w^* = \sum_{i=1}^{n} w(a_i)/P$ is the computational workload per server under the perfectly balanced workload partition strategy. Therefore, $C_{\mathcal{P}}^{W}$ measures the *deviation* from the ideal load balancing partitioning strategy.

For the communication cost between servers under a partition strategy $\mathcal{P}$, we have to consider the AOI of each avatar. Specifically, if avatar a_i is within the AOI of avatar a_j, then any action taken by the avatar a_i needs to be sent to a_j. We let $l(a_i, a_j)$ be a function that assigns the communication cost between a_i and a_j. We define the following indicator function between an avatar u and a partition V_l:

$$ADJ(u, V_l) = \begin{cases} 1 & \text{if } \exists\, v \in V_l \text{ such that avatars } u \text{ and } v \text{ are in } V \text{ and } l(u, v) > 0; \\ 0 & \text{otherwise.} \end{cases}$$

Then, given a partition strategy $\mathcal{P}$, let C_{ij} be the communication cost between partition V_i and V_j. The communication cost C_{ij} can be expressed as:

$$C_{ij} = \sum_{u \in V_i} \max_{v \in V_j} \{l(u, v) * ADJ(u, V_j)\} + \sum_{u \in V_j} \max_{v \in V_i} \{l(u, v) * ADJ(u, V_i)\} \tag{4.2}$$

Let $C_{\mathcal{P}}^{L}$ be the communication cost for the partition strategy $\mathcal{P}$, we have:

$$C_{\mathcal{P}}^{L} = \sum_{i=1}^{P}\sum_{j>i}^{P} C_{ij} \tag{4.3}$$

Therefore, $C_{\mathcal{P}}^{L}$ represents the total server-to-server communication cost given the partition $\mathcal{P}$. The overall cost[1] for the partition strategy $\mathcal{P}$, denotes by $C_{\mathcal{P}}$, can be expressed as:

$$C_{\mathcal{P}} = W_1 C_{\mathcal{P}}^{W} + W_2 C_{\mathcal{P}}^{L} \tag{4.4}$$

where W_1 and W_2 represent the relative importance of the workload cost and the communication cost respectively. For example, to implement a DVE system across the Internet, we can assign more weighting to W_2 so as to reduce the communication cost. Lastly, the DVE partitioning problem is to find an optimal partition $\mathcal{P}^*$ such that

$$C_{\mathcal{P}}^{*} = \min_{\mathcal{P}} \{C_{\mathcal{P}}\} \tag{4.5}$$

Before we discuss the proposed partitioning algorithm, we need to show the following important result[1, 2].

Theorem 1 The workload partitioning problem given in Equation (4.5) is NP-complete.

Proof: Let us consider the simplified version of the workload partition problem where $W_2 = 0$ (which corresponds to the case that the network has an infinite communication bandwidth and therefore the server-to-server communication cost is negligible). Given a set of nodes in V, we partition them into P disjoint subsets $V_1, \ldots, V_P$ such that $\cup_{i=1}^{P} V_i = V$ and the partitioning cost is:

$$C_{\mathcal{P}} = \sum_{i=1}^{P} \left| \sum_{a \in V_i} w(a) - w^* \right|$$

[1] In this paper, we assume the communication cost between avatars which are assigned to the same server to be negligible as compare to the communication cost of avatars which are in different servers. This assumption can be easily relaxed and be included in the total cost $C_{\mathcal{P}}$.

The main idea is to transform the partitioning problem to the *subset sum problem*[14], which is known to be NP-complete.

The subset sum problem can be described as follows. Given a set of real numbers $\mathcal{N} = \{n_1, n_2, \ldots, N_N\}$ and a real value k, the subset sum problem is to determine whether there exists a partitioning of the set $\mathcal{N}$ into disjoint partitions $\mathcal{N}_1, \ldots, \mathcal{N}_l$ such that the sum of the elements in each $\mathcal{N}_i$ is equal to k.

Given an instance of the subset sum problem, the transformation works as follows. We create an avatar for each element $n_i \in \mathcal{N}$, and the value of n_i is equal to $w(a_i)$, the computational workload of the avatar a_i. The number of partitions P for the workload partitioning problem is equal to $\frac{1}{k}\sum_{n_i \in \mathcal{N}} w(a_i)$. If an input instance of the subset sum problem should return a yes, then it implies that the workload partitioning problem can evenly divide up the workload among P servers. If the answer is no, this implies that the workload partitioning problem will have a load imbalance cost which is greater than zero. Since we can transform the subset sum problem in polynomial time and then use the workload partitioning problem to solve the subset sum problem, therefore, the workload partitioning problem is also NP-complete. ∎

4.2 Exhaustive Partition (EP) Algorithm

One way to partition the avatars among different servers is by the exhaustive approach, that is, given n avatars in the DVE system and P servers, then each avatar can have at most P choices, therefore, the total number of partition policies is

$$|\mathcal{P}| = (P)(P)\cdots(P) = P^n \tag{4.6}$$

Note that although the exhaustive algorithm can find the optimal partition (e.g., partition with the minimum cost $C^*_{\mathcal{P}}$), however, this algorithm can only be applied to a small DVE system. For example, for $n = 16, P = 2$, the system needs to evaluate 65536 different partition policies. For a moderate sized DVE system with $n = 16, P = 4$, the exhaustive algorithm requires approximately 4.3×10^9 evaluations to find the optimal partition.

Lemma 1 The complexity of Exhaustive Partition Algorithm is $O(P^{n+2}n^2)$.

Proof: Let the number of avatars and the number of servers be n and P, respectively. The complexity for calculating the cost between two servers is $O(n^2)$. For one evaluation of the cost between P servers, the complexity is $O(P^2n^2)$. We need P^n evaluations to get an optimal solution because there are P^n partition policies. Thus, the overall complexity of the EP algorithm is $O(P^{n+2}n^2)$. ∎

4.3 Partitioning Algorithm

Due to the NP-completeness nature of the problem in Equation (4.5), we propose the following partitioning algorithm. In general, the algorithm has the following steps:

Partitioning Algorithm:

1. **begin**
2. Use the *recursive bisection partitioning(RBP)* algorithm to find the initial partition $\mathcal{P}_1$;
3. current_cost = $C_{\mathcal{P}_1}$;
4. difference = ∞;
5. **while** (difference > d^*){
6. Use the *layering partitioning(LP)* algorithm to find a new partition $\mathcal{P}_2$;
7. Given $\mathcal{P}_2$, use the *communication refinement partitioning(CRP)* algorithm
8. to find a new partition $\mathcal{P}_2'$;
9. difference = $|C_{\mathcal{P}_2'} - \text{current\_cost}|$;
10. current_cost = $C_{\mathcal{P}_2'}$;
11. }
12. final partition is $\mathcal{P}_2'$;
13. **end**

As illustrated, the partitioning algorithm has three components, namely, 1) the *recursive bisection algorithm*, 2) the *layering partitioning algorithm* and, 3) the *communication refinement partitioning algorithm*. The recursive bisection algorithm is a divide-and-conquer approach in finding the initial partition $\mathcal{P}_1$ that reduces the

workload deviation and inter-server communication cost. The layering partitioning algorithm and the refinement partitioning algorithm are based on linear optimization technique[6, 7, 8] to minimize the the workload deviation and the inter-server communication respectively. The algorithm will iterate the layering partitioning algorithm and the communication refinement partitioning algorithm until a difference of the total partitioning cost is less than some pre-defined threshold d^*. In Chapter 5, We will show that the proposed partitioning algorithm can efficiently find a partition strategy that can reduce the total cost.

4.3.1 Recursive Bisection Partition (RBP) Algorithm

The main idea about the recursive bisection partitioning algorithm is to divide up the avatars in the virtual world into groups and then based on divide-and-conquered to find a partitioning strategy $\mathcal{P}$. In the recursive bisection algorithm, we first assume that the AOI of avatar is in the form of a circle with an average diameter of $\mathcal{D}$. We then divide up the virtual world into N disjoint squared cells such that the area of a cell is equal to $\mathcal{D}^2$. The rationale of dividing the virtual world into cells is that with high probability, most of the communication between avatars is between neighboring cells. Figure 4.1 illustrates that the virtual world is divided into 32 disjoint cells.

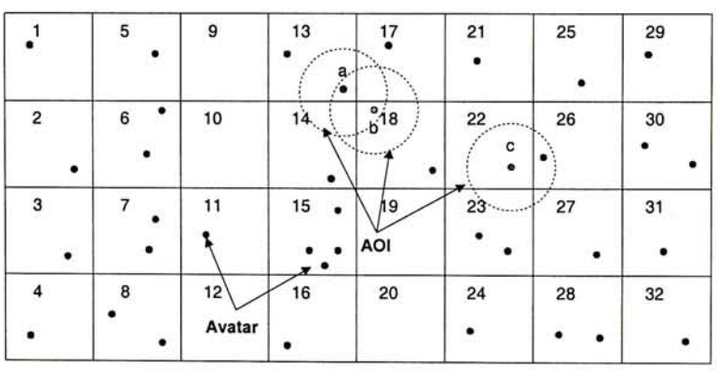


Figure 4.1: A virtual world represented by 32 disjoint cells

Given the state of the DVE system, we can construct a graph $G_{RBP} = (V_{RBP}, E_{RBP})$ as follows:

1. For each cell c_i, $1 \leq i \leq N$, create a node v_i in V_{RBP}.

2. Compute the workload for cell c_i, which is equal to $\sum_{a_j \in c_i} w_{a_j}$.

3. For any two adjacent cell c_i and c_j, there is an edge E_{ij} between v_i and v_j such that $C(E_{ij})$, the cost of E_{ij}, is:

$$C(E_{ij}) = \sum_{u \in c_i} \max_{v \in c_j} \{l(u,v) * ADJ(u,c_j)\} + \sum_{u \in c_j} \max_{v \in c_i} \{l(u,v) * ADJ(u,c_i)\}$$

The recursive bisection partitioning algorithm is based on the concept of divide-and-conquered. Without the loss of generality, let us first present the RBP algorithm for N cells system and the number of servers (P) is equal to two. Let V_k^n be the partition for the k^{th} server (where $k = 1, 2, \ldots, P$) with $n \leq N$ cells, and initially, we set:

$$V_1^N = V_{RBP} = \{v_1, v_2, \ldots, v_N\} \quad ; \quad V_2^0 = \emptyset \tag{4.7}$$

Let $\mathcal{P}_i$ be the i^{th} partition configuration and let $C_{\mathcal{P}_i}$, the cost based on Equation (4.4), be the cost of partition configuration $\mathcal{P}_i$. Based on the initial partition, we have $\mathcal{P}_0 = (V_1^N, V_2^0)$ and the corresponding $C_{\mathcal{P}_0}$. We can then find $\mathcal{P}_1$ by moving one cell from V_1^N to V_2^0 and compute the cost $C_{\mathcal{P}_1}$. Note that the cell can be chosen in such a way that the total cost $C_{\mathcal{P}_1}$ is minimized, which can be achieved by considering each cell in V_1^N and this process takes a linear time with respect to the total number of cells in the system. Formally, we have:

$$\mathcal{P}_i = (V_1^{N-i}, V_2^i) \qquad i = 0, 1, \ldots, N \tag{4.8}$$

where $\mathcal{P}_{(i+1)}$ can be derived by:

$$\begin{aligned} \mathcal{P}_{(i+1)} &= (V_1^{(N-(i+1))}, V_2^{i+1}) \\ &= (V_1^{(N-i)} - \{v_j\}, V_2^i \cup \{v_j\}) \text{ for } v_j \in V_1^{(N-i)} \text{ and } C_{\mathcal{P}_{(i+1)}} \text{ is minimized} \end{aligned} \tag{4.9}$$

Note that $C_{\mathcal{P}_1}$ and $C_{\mathcal{P}_N}$ represent the two extremes of the highest load imbalanced cost (i.e., all cells are assigned to one server and there is no server-to-server communication). Therefore, the RBP algorithm is to choose a configuration that:

$$\mathcal{P}_{i^*} = \{\mathcal{P}_i \mid C_{\mathcal{P}_i} = \min_{0 \leq j \leq N} \{C_{\mathcal{P}_j}\}\} \tag{4.10}$$

The above bisection algorithm applies for $P = 2$. For a larger number of P, we can first use the bisection partitioning algorithm presented above, then choose a partition that has the largest cost and then apply the bisection partitioning algorithm again. At the end of the RPB algorithm, we obtain a partition strategy $\mathcal{P}_{RBP}$ that partition the graph G_{RPB} into P disjoint regions (or $V_{RPB} = \{V_1 \cup \cdots \cup V_P\}$) such that all the nodes in V_i will be assigned to the i^{th} server.

Lemma 2 The complexity of Recursive Bisection Partition Algorithm is $O(N^3(P-1))$.

Proof: Let the number of cells and the number of servers be N and P, respectively. The number of evaluation of the partitioning configurations is $N(P-1)$. For each evaluation, we need to calculate the cost between two servers, the complexity is $O(N^2)$. Therefore, the overall complexity of RBP Algorithm is $O(N^3(P-1))$. ∎

4.3.2 Layering Partitioning (LP) Algorithm

Although the RBP algorithm can produce a partition strategy $\mathcal{P}_{RBP}$, there are several shortcomings in the approach. For example, the computational complexity is high (as illustrated in Chapter 5) and at the same time, the overall cost $C_{\mathcal{P}_{RBP}}$ for $\mathcal{P}_{RBP}$ can still be reduced further. The main idea about the layering partitioning algorithm is to label each avatar using a server number. The label (or server number) serves as a possibility of moving an avatar to that partition. The decision of whether to move the avatar can be formulated as a linear programming optimization which we illustrate in this chapter.

Since we have obtained a partition $\mathcal{P}_{RBP}$ from the bisection partitioning algorithm, we can *relax* the assumption that the DVE world is divided up into cells. We first have to construct a graph $G_{LP} = (V_{LP}, E_{LP})$ such that each node in the graph represents an avatar. An edge $e_{ij} \in E$ represents that avatar a_i is within the AOI of avatar a_j and the cost of this edge e_{ij} is $l(a_i, a_j)$. In general, construction of the graph $G_{LP} = (V_{LP}, E_{LP})$ is:

Graph Construction Algorithm:

1. **begin**
2. **for** each avatar a_i, create a node v_i in G_{LP};
3. **for** each $v_i \in G_{LP}$, **do** { /* initiaze */
4. initialize variables connected[v_i]=false;
5. initialize variables server_number[v_i] = k where $v_i \in V_k$ and $1 \leq k \leq P$;
6. /* note that the server index k for V_k can be obtained from the output of the RBP algorithm */
7. }
8. **for** $v_i \in V_{LP}$ **do** {
9. /* create edges and mark those nodes along the partition boarder as connected */
10. **for** $v_j \in V_{LP}$ where $i \neq j$, **do** {
11. **if** v_j is within the AOI of v_i **then** {
12. create an edge e_{ji} in E_{LP}; /* e_{ji} is an edge between v_j and v_i where $v_i, v_j \in V_{LP}$ */
13. set the weight of $e_{ji} = l(v_j, v_i)$;
14. **if** (server_num[v_i] $\neq$ server_num[v_j])**then** {
15. connected[v_i] = connected[v_j] = true; }
16. }
17. }
18. }
19. **for** all $v_i \in G_{LP}$ **do**{ /* connect the remaining nodes */
20. **if** (connected[v_i] = false) **then** {
21. **if** ((there exists a node v_j which is a neighbor of v_i) /* v_j is a neighbor of v_i if e_{ij} exists*/
22. **and** (connected[v_j] = true)) **then**
23. connected[v_i] = true;
24. **if** (connected[v_i]=false) **do** {
25. find a nearest node v_k such that connected[v_k]=true **and** server_num[v_i]=server_num[v_k];
26. create an edge $e_{ik} \in E_{LP}$;
27. set weight of $e_{ik} = l(v_i, v_k)$ or $\epsilon > 0$;
28. connected[v_i]=true;
29. }
30. **if** (connected[v_i]=false) **do** {
31. find a nearest node $v_k \in G_{LP}$ such that connected[v_k]=true;
32. create an edge $e_{ik} \in E_{LP}$;
33. set weight of $e_{ik} = l(v_i, v_k)$ or $\epsilon > 0$;
34. connected[v_i]=true;
35. }
36. }
37. }
38. **end**

The purpose of the constructing graph G_{LP} is to produce a *connected* graph so that we can perform the layering step[2], which can be described as follows.

First, a node v_i is considered as a *boarder node* when there exists a node v_j such that 1) there exists an edge $e_{ij} \in E_{LP}$ and, 2) server_num$[v_i] \neq$ server_num$[v_j]$. In other words, node v_i is along a partition boarder. Let $\mathcal{S}_{bn}$ be the set of all boarder nodes in the graph G_{LP}. For each node $v_i \in \mathcal{S}_{bn}$, we find a partition j^* such that the sum of weight from node v_i to partition j^* is

$$\max(\sum_{v_k \in V_{j^*}} l(e_{ik})) \qquad \text{for } 1 \leq j^* \leq P$$

Then we set the layer number of the node v_i, denotes by layer_num$[v_i]$, as j^*. At this point, we let $\mathcal{S}_l$ to denote the set of nodes that has an assigned layer number. Note that $\mathcal{S}_l \subset V_{LP}$. The remaining step is to consider all those nodes in G_{LP} which have no layer number yet. To accomplish this, let us consider a node v_i which has no layer number. For this node v_i, we find a label j^* such that the sum of weight from node v_i to nodes with label j^* is

$$\max(\sum_{v_k \in \mathcal{S}_l} l(e_{ik})) \qquad \text{where } layer\_num[v_k] = j^*$$

then we set layer_num$[v_i]$, the layer number of node v_i, as j^*. Now for all those nodes that have the newly assigned layer number, we add them to the set $\mathcal{S}_l$. We repeat the layer number assignment process for all those nodes in G_{LP} that have no layer number.

Figure 4.2 illustrates a graph G_{LP} with the corresponding edge weight and all the boarder nodes are high lighted. Note that graph G_{LP} is divided into three partitions, namely, V_i, V_j and V_k. Figure 4.3 shows the assignment of layer number for the boarder nodes and Figure 4.4 illustrates the assignment of layer number for the remaining nodes.

After we finished layering all nodes in the graph G_{LP}, we can consider moving some of the nodes with layer number i to server i, where $1 \leq i \leq P$, so as to reduce the workload deviation (e.g., reduce the workload cost according to Equation (4.1)).

[2]The purpose of the layering is to identify which node can be assigned to a different servers.

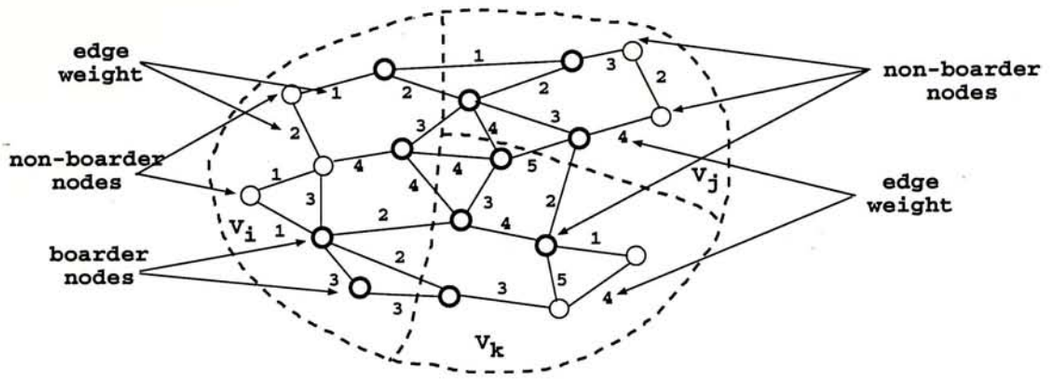


Figure 4.2: Graph G_{LP} with boarder nodes (nodes in bold circle), edge weight and three partitions V_i, V_j and V_k

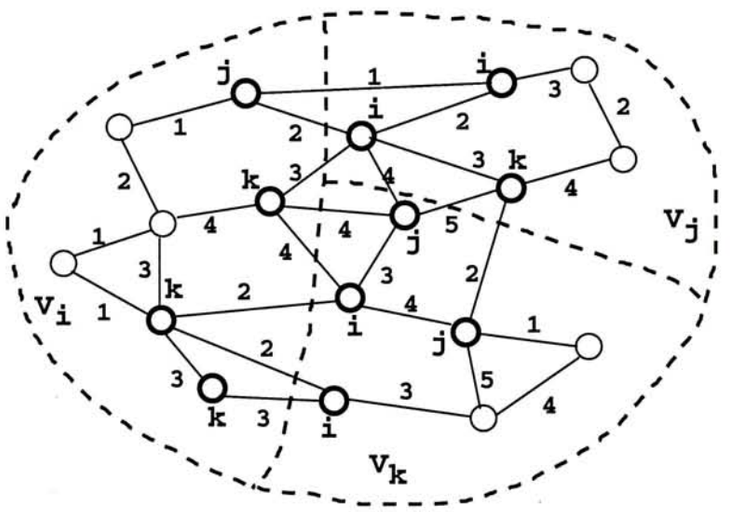


Figure 4.3: Assigning layer numbers to boarder nodes in G_{LP}

The number of nodes that can be moved can be formulated as an linear optimization problem. Let α_{ij} represents the number of nodes in partition V_i that can be moved to partition V_j (e.g., these are the nodes that are in partition V_i and with layer number equal to j). For example, in Figure 4.3(b), $\alpha_{ij} = 2$. Let $|V_i|$ represents the total number of nodes in partition V_i (e.g. in Figure 4.3(b), $|V_i| = 7$). Let x_{ij} to be the decision variable of the number of nodes that we eventually move from partition V_i to V_j so as to reduce the workload cost of the DVE system. We would like to minimize the total number of movement, or minimize $\sum_{1 \leq i \neq j \leq P} x_{ij}$, so as to achieve workload balancing property.

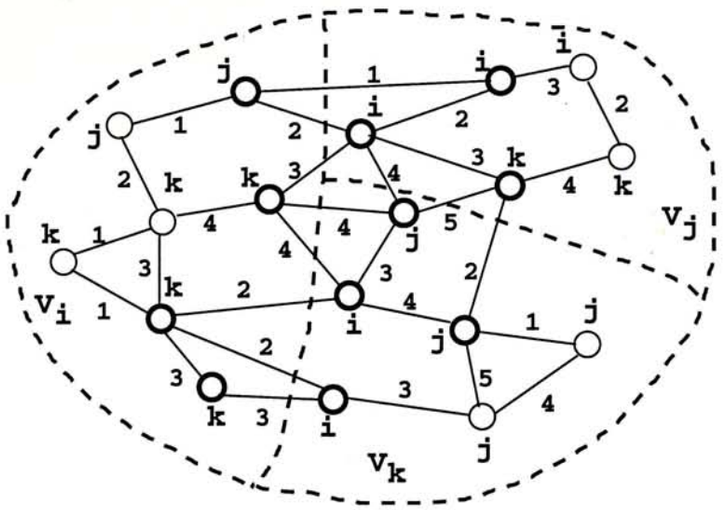


Figure 4.4: Assigning layer numbers to other nodes in G_{LP}

The formulation of the linear optimization is:

$$\text{Minimize} \sum_{1 \le i \ne j \le P} x_{ij} \tag{4.11}$$

subject to

$$0 \le x_{ij} \le \alpha_{ij} \le |V_i| \tag{4.12}$$

$$\sum_{1 \le i \le P} (x_{ij} - x_{ji}) = |V_j| - \frac{1}{P} \sum_{i=1}^{N} w(a_i) \quad \text{for } 1 \le j \le P \tag{4.13}$$

The constraint in Equation (4.12) is to ensure that the number of nodes that we move from partition V_i to partition V_j is less than or equal to the feasible number of candidate nodes. The constraint in Equation (4.13) is to ensure that the difference of the total number of nodes that move into server i and the total number of nodes that move out of server i is equal to the workload deviation of server i under the ideal load balanced situation. In other words, we try to make sure the workload in server i is as close to the ideal workload balanced situation as possible.

Lemma 3 The complexity of Layering Partitioning Algorithm is $O(P^6)$.

Proof: Let the number of partitions be P. We will have $P(P-1)$ variables and $P(P-1)+P = P^2$ constraints. The number of iterations required for linear programming is problem dependent. A better estimate is $2(P^2 + P(P-1))$. The time required

for one iteration of the linear programming is $O(P(P-1) * P^2)$. Thus, the overall complexity for the linear programming is $O(P^6)$. ∎

To illustrate this linear optimization problem, let us consider the graph in Figure 4.4. The formulation is given as follows.

$$\begin{aligned}
&\textbf{Minimize} && x_{ij} + x_{ik} + x_{ji} + x_{jk} + x_{ki} + x_{kj} \\
&\textbf{subject to:} && x_{ij} \leq 2; x_{ik} \leq 5; x_{ji} \leq 3; x_{jk} \leq 2; x_{ki} \leq 2; x_{kj} \leq 4 \\
& && x_{ij} + x_{ik} - x_{ji} - x_{ki} = 7 - 6 = 1 \\
& && x_{ji} + x_{jk} - x_{ij} - x_{kj} = 5 - 6 = -1 \\
& && x_{ki} + x_{kj} - x_{ik} - x_{jk} = 6 - 6 = 0
\end{aligned}$$

The solution to the above optimization problem is $x_{ij} = 1$, $x_{ik} = x_{ji} = x_{jk} = x_{ki} = x_{kj} = 0$. Therefore, we choose the node that has a layer number equal to j in partition V_i and move it to partition V_j. In selecting which node to move, we start from the boarder nodes, then to the nodes in the inner layers. Note that during the node movement, only those nodes which will not increase the communication cost will be moved. After we moved a node v_l from partition V_i to V_j, we re-assign the server number of node v_l as server_num(v_l)= j. The process stops when the number of nodes moved is equal to the solution. The new partitioning policy $\mathcal{P}_{LP}$ for the graph G_{LP} in Figure 4.4 is given in Figure 4.5.

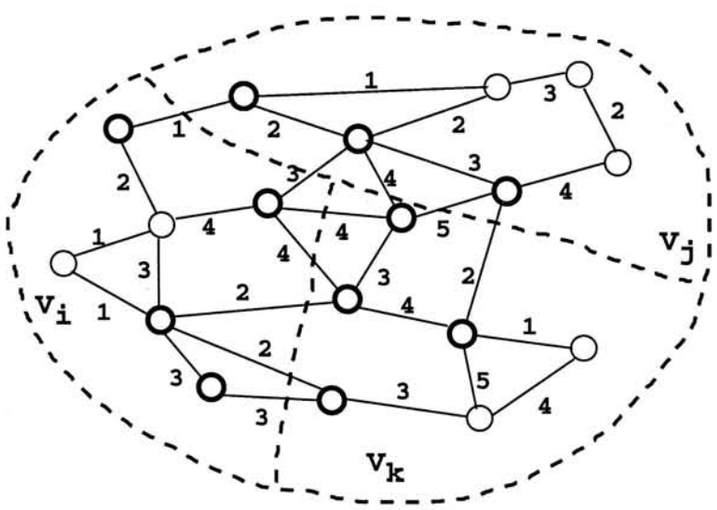


Figure 4.5: New partition

Let the workload weighting W_1 and the communication cost weighting W_2 be 0.5 and 0.5, respectively. Assume the workload for maintaining an avatar is 10.

Before we apply the LP algorithm,

$$\begin{aligned}
C_{\mathcal{P}}^{W} &= \sum_{j=1}^{P} \left(\left| \sum_{a_i \in V_j} w(a_i) - w^* \right| \right) \\
&= 10 \times (|7-6| + |5-6| + |6-6|) = 20 \\
C_{\mathcal{P}}^{L} &= C_{ij} + C_{ji} + C_{ik} + C_{ki} + C_{jk} + Ckj \\
&= (2+3) + (1+3) + (4+2+3) + (4+4+3) + (4+5) + (5+2) \\
&= 45 \\
C_{\mathcal{P}} &= W_1 C_{\mathcal{P}}^{W} + W_2 C_{\mathcal{P}}^{L} \\
&= 0.5 \times 20 + 0.5 \times 45 = 32.5
\end{aligned}$$

After we applied the LP algorithm,

$$\begin{aligned}
C_{\mathcal{P}}^{W} &= \sum_{j=1}^{P} \left(\left| \sum_{a_i \in V_j} w(a_i) - w^* \right| \right) \\
&= 10 \times (|6-6| + |6-6| + |6-6|) = 0 \\
C_{\mathcal{P}}^{L} &= C_{ij} + C_{ji} + C_{ik} + C_{ki} + C_{jk} + Ckj \\
&= (1+3) + (1+3) + (4+2+3) + (4+4+3) + (4+5) + (5+2) \\
&= 44 \\
C_{\mathcal{P}} &= W_1 C_{\mathcal{P}}^{W} + W_2 C_{\mathcal{P}}^{L} \\
&= 0.5 \times 0 + 0.5 \times 44 = 22
\end{aligned}$$

We can see that the cost is reduced by more than thirty percent. In Chapter 5, we illustrate that by using the layer partitioning algorithm there is a large reduction in the overall partition cost.

4.3.3 Communication Refinement Partitioning (CRP) Algorithm

The layering partitioning (LP) algorithm is to reduce the overall workload cost of the system. In the communication refinement partitioning (CRP) algorithm, the objective

is to re-assign some nodes (or avatars) to another partition so as to reduce the server-to-server communication cost.

Given the partitioned graph $G_{LP} = \{V_{LP}, E_{LP}\}$ from the LP algorithm, let us consider all the boarder nodes. Again, a node v_i is considered as a boarder nodes when there exists a node v_j such that 1) there is an edge from $e_{ij} \in E_{LP}$ and, 2) server_num$(v_i) \neq$ server_num(v_j). Let $\mathcal{S}_{bn}$ be the set of all boarder nodes. For each $v_i \in \mathcal{S}_{bn}$, we compute Γ_j, the communication cost to partition V_j, as:

$$\Gamma_j = \sum_{v_k \in V_j} l(e_{ik}) \qquad \text{for } 1 \leq j \leq P \tag{4.14}$$

Let $\Gamma_{j^*} = \max_{1 \leq j \leq P}\{\Gamma_j\}$. Then Γ_{j^*} is the maximum communication due to node v_i and this communication is to partition V_{j^*}. Then the node $v_i \in \mathcal{S}_{bn}$ will be assigned a label j^*, that is, communication_number(v_i)=j^*. The motivation of finding the communication_number is that if we move node v_i to partition V_{j^*}, then it is possible for us to reduce the communication cost. We repeat this process for all nodes in $\mathcal{S}_{bn}$. At the end of this communication number assignment, we have determined the possible partition assignment of all those boarder nodes so as to reduce the system communication cost.

In order to determine the final partition assignment, we can formulate the problem as a linear optimization problem. Let β_{ij} denotes the number of nodes in partition V_i that has the communication_number assignment equal to j. We define y_{ij} to be the decision variable of the number of nodes that we eventually move from partition V_i to V_j so as to reduce the communication cost of the system. We can formulate the problem as:

$$\text{Maximize} \sum_{1 \leq i \neq j \leq P} y_{ij} \tag{4.15}$$

subject to

$$0 \leq y_{ij} \leq \beta_{ij} \qquad \text{for } 1 \leq i \neq j < P \tag{4.16}$$

$$\sum_{1 \leq i < j} (y_{ij} - y_{ji}) = 0 \qquad (0 \leq j < P) \tag{4.17}$$

The constraint in Equation (4.16) is to ensure that the number of nodes that we move from partition V_i to partition V_j is less than or equal to the feasible number of candidate nodes. The constraint in Equation (4.17) is to ensure that the number of nodes that move into server i and the number of nodes that move out of server i is the same so as to maintain the workload balancing property.

Lemma 4 The complexity of Communication Refinement Partitioning Algorithm is $O(P^6)$.

Proof: This algorithm also uses linear programming technique. Let the number of partitions be P. We will have $P(P-1)$ variables and $P(P-1)+P = P^2$ constraints. The number of iterations required for linear programming is problem dependent. A better estimate is $2(P^2 + P(P-1))$. The time required for one iteration of the linear programming is $O(P(P-1) * P^2)$. Thus, the overall complexity for the linear programming is $O(P^6)$. ∎

Let us illustrate the refinement algorithm given that we have the partitioned graph G_{LP} in Figure 4.5. The optimization formulation is:

$$
\begin{aligned}
\textbf{Maximize} \quad & x_{ij} + x_{ik} + x_{ji} + x_{jk} + x_{ki} + x_{kj} \\
\textbf{subject to:} \quad & x_{ij} \leq 0; x_{ik} \leq 1; x_{ji} \leq 0; x_{jk} \leq 0; x_{ki} \leq 1; x_{kj} \leq 1 \\
& x_{ij} + x_{ik} - x_{ji} - x_{ki} = 0 \\
& x_{ji} + x_{jk} - x_{ij} - x_{kj} = 0 \\
& x_{ki} + x_{kj} - x_{ik} - x_{jk} = 0
\end{aligned}
$$

The solution is $x_{ik} = 1, x_{ki} = 1$, all other values are zero. Therefore, we can exchange one node between partitions V_i and V_k to reduce communication cost. The new partitioning is given in Figure 4.7.

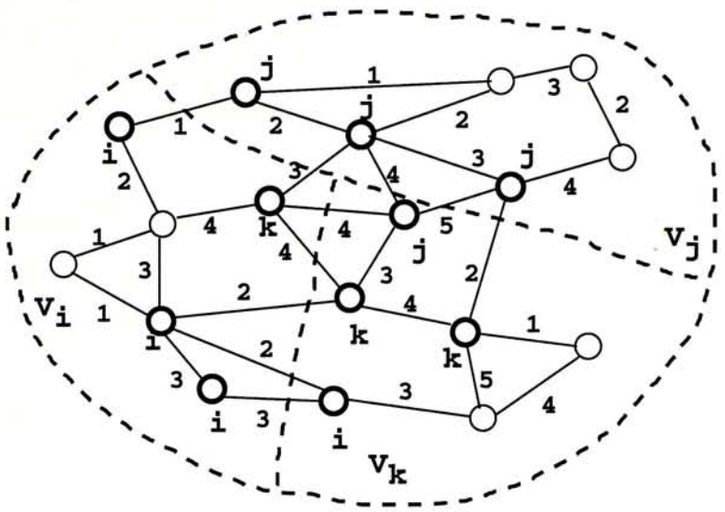


Figure 4.6: Before refinement

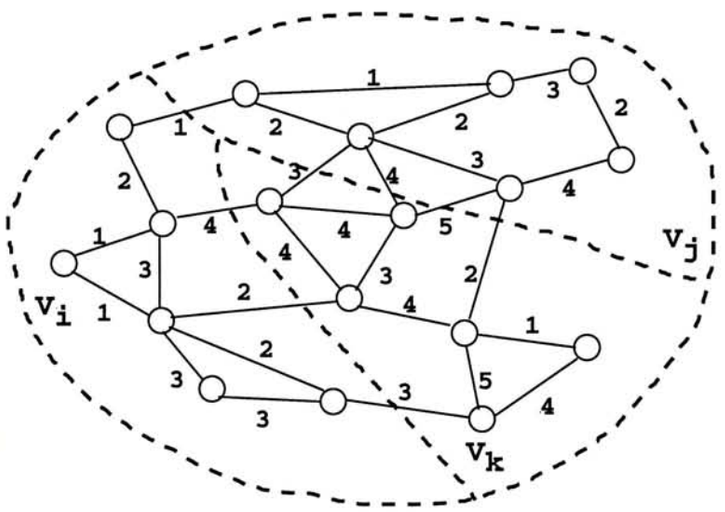


Figure 4.7: After refinement

After we apply the CRP algorithm,

$$
\begin{aligned}
C_{\mathcal{P}}^{W} &= \sum_{j=1}^{P} \left(\left| \sum_{a_i \in V_j} w(a_i) - w^* \right| \right) \\
&= 10 \times (|6-6| + |6-6| + |6-6|) = 0 \\
C_{\mathcal{P}}^{L} &= C_{ij} + C_{ji} + C_{ik} + C_{ki} + C_{jk} + Ckj \\
&= 1 + 1 + (4+2+3) + (4+2+3) + (4+5) + (3+5+2) \\
&= 39 \\
C_{\mathcal{P}} &= W_1 C_{\mathcal{P}}^{W} + W_2 C_{\mathcal{P}}^{L} \\
&= 0.5 \times 0 + 0.5 \times 39 = 19.5
\end{aligned}
$$

4.4 Parallel Approach

DVE systems are growing to include more entities and clients. As the number of clients places a heavy burden on both the networking resources and computational resources available to the system, we will have to add more servers to maintain the virtual world. As the number of clients and servers increases tremendously, the processing time of our previous algorithm also increases tremendously. To keep the system cost at a lower level, we need a partitioning algorithm with high efficiency. We introduce a parallel approach to improve the performance. In general, the parallel approach has the following steps:

Parallel Approach:

1. **begin**
2. Divide the virtual world into four smaller virtual worlds;
3. Distribute the smaller virtual worlds to different processors;
4. **for** each processor i **do** {
5. Use the *recursive bisection partitioning(RBP)*
6. algorithm to find the initial partition $\mathcal{P}_1$;
7. current_cost = $C_{\mathcal{P}_1}$;
8. difference = ∞;
9. **while** (difference > d^*){
10. Use the *layering partitioning algorithm(LP)* to find a new partition $\mathcal{P}_2$;
11. Given $\mathcal{P}_2$, use the *communication refinement partitioning(CRP) algorithm*
12. to find a new partition $\mathcal{P}_2'$;
13. difference = $|C_{\mathcal{P}_2'} - \text{current\_cost}|$;
14. current_cost = $C_{\mathcal{P}_2'}$;
15. }
16. }
17. Combine the smaller virtual worlds into one large virtual world;
18. Use LP algorithm to find a new partition $\mathcal{P}_3$;
19. Use CRP algorithm to find a new partition $\mathcal{P}_3'$ based on $\mathcal{P}_3$;
20. final partition is $\mathcal{P}_3'$;
21. **end**

Consider the partitioning of a virtual world with 16 servers. This can be completed in two stages: in the first stage, we can roughly divide the virtual world into four smaller

virtual worlds. In the second stage, we apply the partitioning algorithm we discussed in section 4.3 to partition each of the small virtual worlds. The partitioning processes can be carried out in different machines. From the experiments in chapter 5, we can see that the parallel approach can have a comparable partitioning cost as compare to the unparallel approach. The advantage of this approach is that the time required for applying RBP, LP and CRP algorithms to each small virtual world will be much less than the time required for applying those algorithms to the whole large virtual world. We may add one more stage to this approach. In this additional stage, we combine all small virtual worlds and then apply LP and CRP algorithms to the combined virtual world once. We can see that the result is much better than the result of unparallel approach.

4.5 Further Observation

We can integrate the filtering approaches we discussed in chapter 3 with our proposed partitioning algorithm to cancel much of the WAN traffic. We can first apply the proposed algorithm to partition/map clients and entities to different servers. We then use filtering approaches to further reduce communication and computation cost. Although the good intrinsic behavior induced by the initial mapping will be likely to deteriorate slowly over time (as avatars move and interact), the overall benefit of a good initial mapping will still be significant. For many DVE systems (meetings lasting only few hours or shows dominated by slow moving visitors), the positive impact of a good initial mapping will most likely persist throughout the duration of the exercise. This suggests that proper consideration needs to be given to the initial entity-to-host mapping during the preparation and planning for an exercise. Another related issue with similar potential benefits concerns entities belonging to the computer controlled objects. These entities are not associated with actual clients and do not therefore have to be tied up to a fixed machine. As these entities move and change state over time, a dynamic migration protocol can be used to remap each entity to the host in DVE systems that will most preserve the locality behavior mentioned above. The design and evaluation of this migration protocol is one task worthy of investigation.

Chapter 5

Experiments

In this chapter, we present experiments for the algorithms that we discussed in the previous chapter and apply them to both a small and a large virtual world. For the small virtual world experiment, since the problem state space is manageable, we can compare the performance of our proposed algorithm with the exhaustive partitioning algorithm, which guarantees to produce the optimal partition $\mathcal{P}^*$. We also carry out experiments to investigate the dynamic characteristics of avatars and the effectiveness of parallel approach. In general, we use three different methods to generate the position of each avatar in the virtual world. These methods are:

- *Uniform Distribution:* Let the position of an avatar be (x, y) and the values of x and y are uniformly distributed between $(0, V_x)$ and $(0, V_y)$ where V_x is the horizontal dimension of the virtual world and V_y is the vertical dimension of the virtual world.

- *Skewed Distribution:* Given the size of the DVE world as (V_x, V_y), we divide the number of avatars in the DVE system into four equal sized groups, namely, G_i, $i = 1, 2, 3, 4$. Let (x, y) be the position of the avatar in group G_i. The value of (x, y) is generated in such a way that x is uniformly distributed between $(0, \frac{iV_x}{4})$ and y is uniformly distributed between $(0, \frac{iV_y}{4})$. Under this scheme, most of the avatars will be positioned within the square area defined by the two coordinates $[0, 0]$ and $[\frac{V_x}{4}, \frac{V_y}{4}]$.

- *Clustered Distribution:* Given the size of the DVE world as (V_x, V_y), we will generate avatars around $k \geq 1$ clusters. First, we randomly generate k points

$(x_1, y_1), \ldots, (x_k, y_k)$ such that x_i and y_i are uniformly distributed between $(0, V_x)$ and $(0, V_y)$ respectively. Then we divide the number of avatars in the DVE system into k equal-sized groups, namely, $G_1, G_2, \ldots, G_k$. For each avatar in group G_i, we generate its position in (x, y) where

$$x = \begin{cases} 0 & \text{if } x_i + dx \times \Omega < 0 \\ V_x & \text{if } x_i + dx \times \Omega > V_x \\ x_i + dx \times \Omega & \text{otherwise} \end{cases} \tag{5.1}$$

$$y = \begin{cases} 0 & \text{if } y_i + dy \times \Omega < 0 \\ V_y & \text{if } y_i + dy \times \Omega > V_y \\ y_i + dy \times \Omega & \text{otherwise} \end{cases} \tag{5.2}$$

In our experiments, dx and dy are generated uniformly between $(-1, 1)$ and the parameter Ω depends on the size of the virtual world. For example, we set $\Omega = 0.4$ for the virtual world composed of 4×4 units and $\Omega = 3.0$ for the virtual world composed of 25×25 units.

5.1 Experiment 1: Small Virtual World

In this experiment, we use a small virtual world which is composed of 4×4 units with the total number of avatars equal to 13 and the number of servers P equal to three. We set both the workload weighting, W_1 and the communication cost weighting, W_2 to 0.5. The AOI of each avatar is equal to 1.

Table 5.1, 5.2 and 5.3 illustrate the experimental results under the uniform, skewed and clustered location distributions respectively. It is important to observe that the proposed algorithm can have a comparable partitioning cost as compare to the exhaustive algorithm and at the same time, only requires a very small fraction of processing time.

Algorithm	Exhaustive	RBP	LP	CRP
System Cost $C_{\mathcal{P}}$	3.47	3.47	3.47	3.47
Computation Time (secs)	1202.12	0.01	<0.01	<0.01

Table 5.1: Small virtual world under uniform distribution

Algorithm	Exhaustive	RBP	LP	CRP
System Cost $C_{\mathcal{P}}$	5.45	9.66	7.45	7.45
Computation Time (secs)	1290.88	0.01	0.01	<0.01

Table 5.2: Small virtual world under skewed distribution

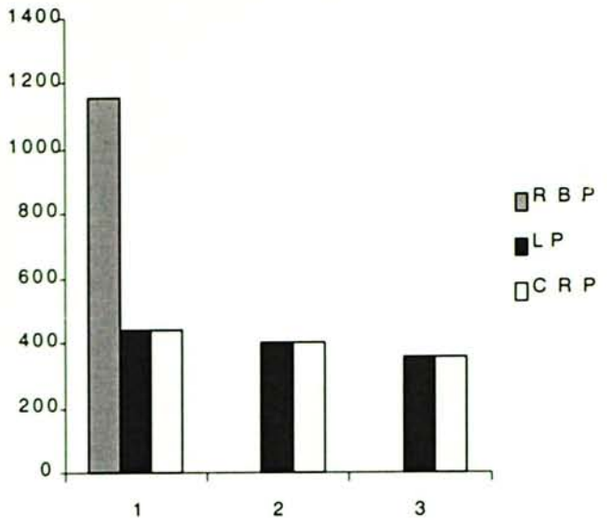
5.2 Experiment 2: Large Virtual World

In this experiment, we use a large virtual world which is composed by 25 x 25 units with the total number of avatars equal to 2500 and the number of servers, $P = 8$. The AOI of each avatar is equal to 0.5. Figure 5.1 illustrates this virtual world under three different location distributions.

Table 5.4, 5.5 and 5.6 illustrate the experimental results under the uniform, skewed and clustered location distributions respectively. Since the size of the virtual world is large, it is impossible to use the exhaustive algorithm (since it has 8^{2500} possible partition schemes). It is important to observe that by applying the proposed algorithm, we can generate a good partition with much lower execution time. For all location distributions, we iterate the partitioning algorithm three times. During the first iteration, we execute the RBP, LP and the CRP algorithms. For the second and third iteration, we only need to iterate the LP and the CRP algorithms. After three iterations, the system cost converges to a fixed value.

Algorithm	Exhaustive	RBP	LP	CRP
System Cost $C_{\mathcal{P}}$	4.39	9.42	7.12	7.12
Computation Time (secs)	1199.68	0.01	0.01	<0.01

Table 5.3: Small virtual world under clustered distribution



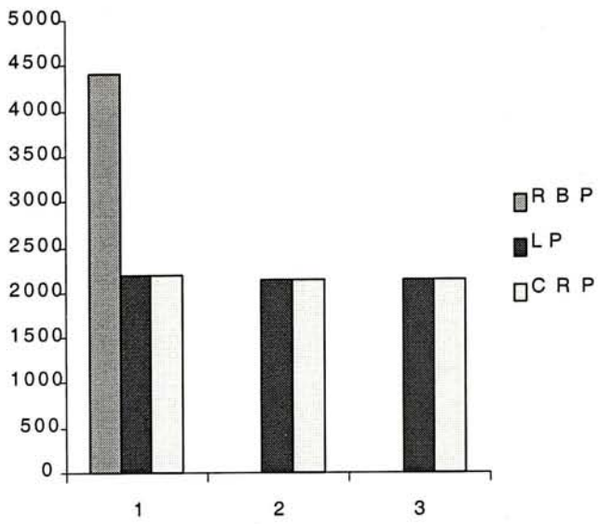
Iteration Count	1	1	1	2	2	3	3
Algorithm	RBP	LP	CRP	LP	CRP	LP	CRP
System Cost C_P	1160.03	443.58	441.06	399.61	399.61	358.07	358.07
Computation Time (secs)	162	0.41	0.38	0.4	0.26	0.35	0.24

Table 5.4: Experimental results under uniform distribution

5.3 Experiment 3: Moving of Avatars

In this experiment, we use the same setting as Experiment 2. We assume that the avatars in the second experiment will move to another position with a probability of 0.3 and stay in current position with a probability of 0.7. If an avatar should move, it will move to a random location which is within its AOI. Since the initial partition is already known (e.g., this is the partition we obtained from Experiment 2), all we need to perform is to iterate the LP and the CRP algorithms. Again, the system cost converges to a small value after three iterations.

Table 5.7, 5.8 and 5.9 illustrate the experimental results under the uniform, skewed and clustered location distributions respectively. It is important to observe that by applying the LP and CRP partitioning algorithms iteratively, we can get a good partition within short time intervals.



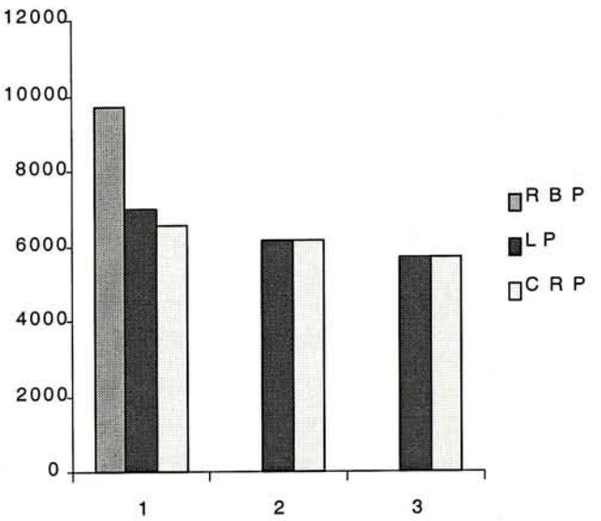
Iteration Count	1	1	1	2	2	3	3
Algorithm	RBP	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	4405.73	2188.83	2188.83	2143.36	2143.36	2136.24	2136.24
Computation Time (secs)	226.1	0.39	0.39	0.36	0.33	0.35	0.31

Table 5.5: Experimental results under skewed distribution

5.4 Experiment 4: Dynamic Joining and Leaving

In this experiment, we use the same setting as the one found in Experiment 3. That is, the avatars in the second experiment will move to another position with a probability of 0.3 and stay in current position with a probability of 0.7. Moreover, approximate 50 avatars left the virtual world and at the same time, approximate 150 avatars joined the virtual world. Again, the system cost converges to a small value after several iterations.

Table 5.10, 5.11 and 5.12 illustrate the experimental results under the uniform, skewed and clustered location distributions respectively. It is important to observe that within a short period of time (less than 5 seconds), we can get a good partition scheme.



Iteration Count	1	1	1	2	2	3	3
Algorithm	RBP	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	9714.83	6974.77	6531.38	6120.48	6114.32	5668.01	5668.01
Computation Time (secs)	166.5	0.33	0.39	0.36	0.31	0.36	0.27

Table 5.6: Experimental results under clustered distribution

Iteration Count		1	1	2	2	3	3
Algorithm	moved	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	4839.62	485.01	485.01	462.78	462.78	462.78	462.78
Computation Time (secs)	-	0.41	0.44	0.39	0.47	0.45	0.48

Table 5.7: Experimental results under uniform distribution after avatars moved

5.5 Experiment 5: Parallel Approach

In this experiment, we use a very large virtual world which is composed by 30 x 30 units with the total number of avatars equal to 25000 and the number of servers, $P = 16$. The AOI of each avatar is equal to 0.5.

Table 5.13, 5.14 and 5.15 illustrate the experimental results under the uniform, skewed and clustered location distribution using the same approach as experiment 2. Since the size of the virtual world is 10 times larger than the large virtual world in experiment 2, we need to add more servers to maintain the virtual world. As both clients and servers increase tremendously, the time consumed in partitioning the

Iteration Count		1	1	2	2	3	3
Algorithm	moved	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	4405.73	2188.83	2188.83	2143.36	2143.36	2136.24	2136.24
Computation Time (secs)	-	0.43	0.42	0.40	0.35	0.38	0.34

Table 5.8: Experimental results under skewed distribution after avatars moved

Iteration Count		1	1	2	2	3	3
Algorithm	moved	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	34317.51	9160.67	8817.35	6643.21	6643.21	6643.21	6643.21
Computation Time (secs)	-	0.36	0.39	0.35	0.25	0.36	0.31

Table 5.9: Experimental results under clustered distribution after avatars moved

virtual world also increases tremendously. To improve the performance, we use a parallel approach which is described in previous chapter. It is important to observe that by using the parallel approach, we can generate a good partition with much lower execution time. First, we divide the very large virtual world into four smaller virtual worlds. For all divided virtual worlds, we iterate the partitioning algorithm twice. During the first iteration, we execute the RBP, LP and the CRP algorithms. For the second iteration, we only iterate the LP and CRP algorithms. After two iterations, the parallel approach can have a comparable partitioning cost as compare to the unparallel approach used in experiment 2. At the same time, the parallel approach only requires a very small fraction of processing time. The result will be much better if we take more time to execute the LP and CRP algorithms once on the combined virtual world. Table 5.16, 5.17 and 5.18 illustrate the experimental results. Figure 5.2 illustrates the processing time under different approaches. Figure 5.3 illustrates the partitioning cost under different approaches.

Iteration Count		1	1	2	2	3	3
Algorithm	Dynamic	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	1469.15	1215.09	1055.30	1055.30	1019.90	1019.90	1013.36
Computation Time (secs)	-	0.60	0.40	0.70	0.44	0.59	0.46

Table 5.10: Dynamic join and leave under uniform distribution

Iteration Count		1	1	2	2	3	3
Algorithm	Dynamic	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	9066.94	6061.10	5849.96	5828.21	5461.38	5461.38	5456.71
Computation Time (secs)	-	0.63	0.44	0.62	0.52	0.64	0.50

Table 5.11: Dynamic join and leave under skewed distribution

Iteration Count		1	1	2	2	3	3
Algorithm	Dynamic	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	39662.02	17799.05	10889.81	10825.43	10359.41	9825.01	8873.03
Computation Time (secs)	-	0.43	0.39	0.43	0.32	0.44	0.34

Table 5.12: Dynamic join and leave under clustered distribution

	1	2	3
Algorithm	Bi-P	Layering	Refinement
Cost	73013.99	40453.82	38656.35
Time(s)	641.22	44.91	38.02

Table 5.13: Uniform, 30x30 cells, 16 partitions, 25000 avatars

	1	2	3
Algorithm	Bi-P	Layering	Refinement
Cost	195224.53	130151.50	126849.17
Time(s)	811.00	34.61	31.10

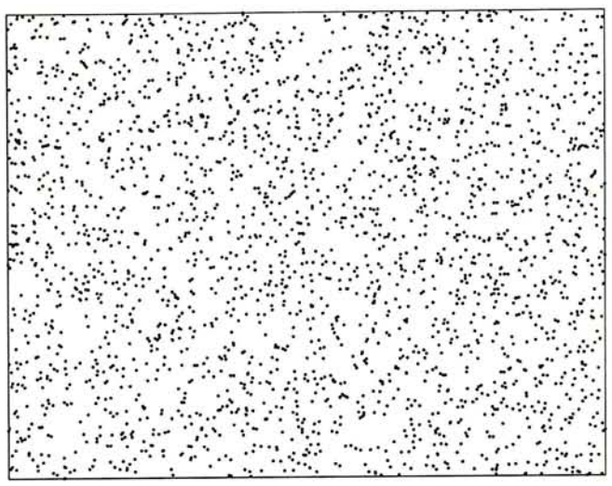
Table 5.14: Skewed, 30x30 cells, 16 partitions, 25000 avatars

	1	2	3
Algorithm	Bi-P	Layering	Refinement
Cost	265481.66	163869.56	157648.94
Time(s)	749.25	51.13	35.82

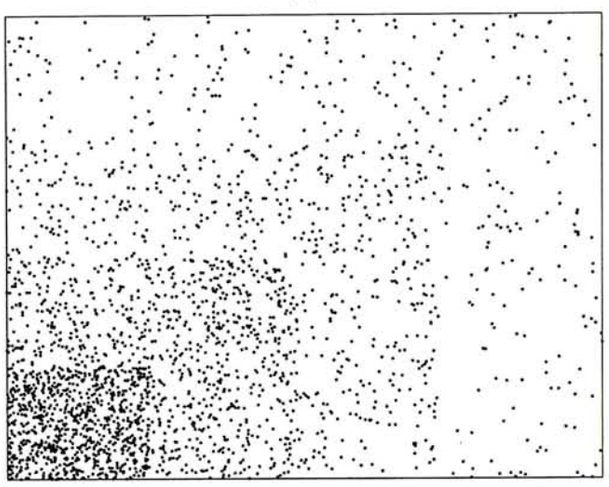
Table 5.15: Clustered, 30x30 cells, 16 partitions, 25000 avatars

	1	2	3
Algorithm	Parallel	Layering	Refinement
Cost	56931.76	37818.00	37694.41
Time(s)	7.00	44.61	27.59

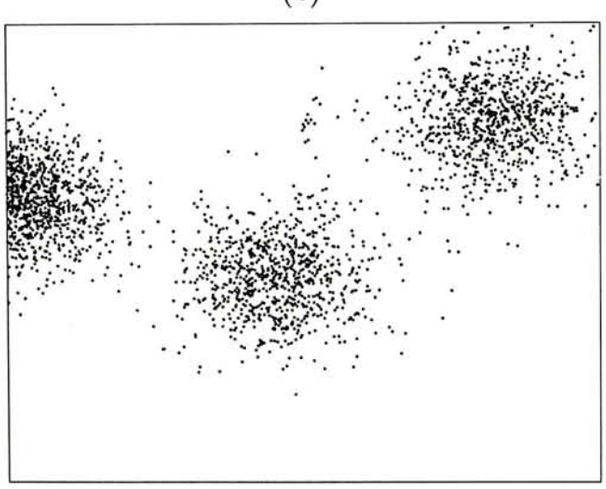
Table 5.16: Combined uniform world, 30x30 cells, 16 partitions, 25000 avatars



(a)



(b)



(c)

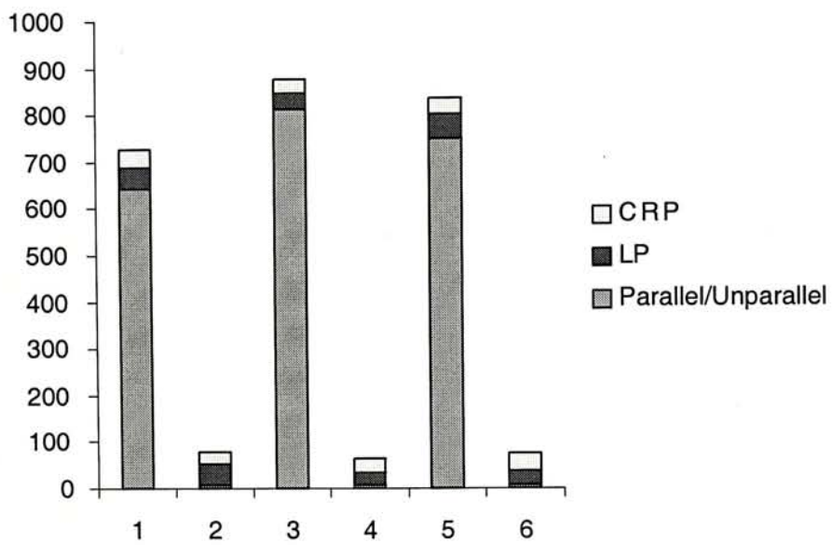
Figure 5.1: Virtual world with a 25×25 cells under under (a) Uniform (b) Skewed (c) Clustered location distribution

	1	2	3
Algorithm	Parallel	Layering	Refinement
Cost	128759.35	81014.57	80542.74
Time(s)	8.00	25.22	32.28

Table 5.17: Combined skewed world, 30x30 cells, 16 partitions, 25000 avatars

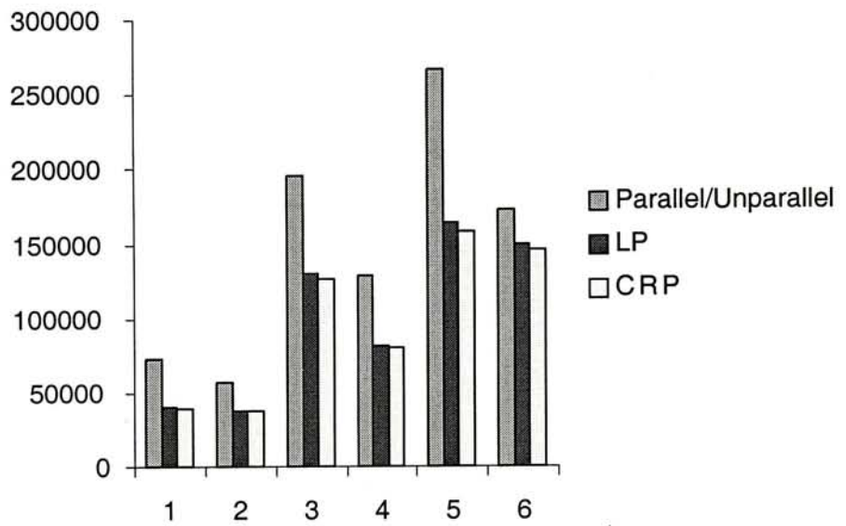
	1	2	3
Algorithm	Parallel	Layering	Refinement
Cost	172824.14	148949.02	145532.83
Time(s)	9.00	28.50	37.61

Table 5.18: Combined clustered world, 30x30 cells, 16 partitions, 25000 avatars



Distribution	Uniform		Skewed		Clustered	
Algorithm	Unparallel	Parallel	Unparallel	Parallel	Unparallel	Parallel
Time	641.22	7.0	811.00	8.0	749.25	9.0
Time(LP)	44.91	44.61	34.61	25.22	51.13	28.50
Time(CRP)	38.02	27.59	31.10	32.28	35.82	37.61

Figure 5.2: Processing time under different approaches



Distribution	Uniform		Skewed		Clustered	
Algorithm	Unparallel	Parallel	Unparallel	Parallel	Unparallel	Parallel
Cost	73013.99	56931.76	195224.53	128759.35	265481.66	172824.14
Cost(LP)	40453.82	37818.00	130151.50	81014.57	163869.56	148949.02
Cost(CRP)	38656.35	37694.41	126849.17	80542.74	157648.94	145532.83

Figure 5.3: Cost under different approaches

Chapter 6

Implementation Considerations

Software design and implementation is complicated by issues such as complex interactions among models, real time man-in-loop requirements, and transparently integrating real systems with virtual components so the DVE presents a consistent and realistic virtual environment. Having studied the related techniques of DVE, I mention some considerations in designing our DVE system.

6.1 Different Environments

Most work in distributed virtual environment has been done on vehicle simulation. However, the DVE we implemented is mainly indoor environment. Although there are many similarities between vehicle simulators and building walkthrough systems, there are several important differences [31]. First, typical vehicle simulator models contain terrain data augmented with plants, buildings, roads, etc. In these models, space tends to be "sparsely occluded". In contrast, building models typically contain walls, ceilings, and floors which partition space into rooms. These models tend to be "densely occluded". Second, in a vehicle simulator, the observer viewpoint navigation is limited to movements possible by the vehicle. During normal execution, the observer does not generally move sideways, or change direction suddenly. As a result, there is a large amount of coherence in the observer position from frame to frame, and it is relatively easy to predict future observer viewpoints from the current observer viewpoint and direction of travel. In addition, since the observer rarely travels close to detailed

model features, realistic-looking detail can be achieved using texture maps applied to relatively few, distant polygons. In contrast, in a building system, the observer may step in any direction, spin around quickly, or look very closely at any feature of the model. Therefore, many of the optimizations used by vehicle simulators based on assumptions of observer navigation are not possible in a indoor environment system. All these differences must be carefully studied in implementing indoor walkthrough system.

6.2 Platform

Our previous virtual environment implementation was developed in *performer* on SGI platform. The rendering quality is quite good by using this combination. However, our application could not run on other platforms because *performer* is only supported by SGI machines. As a result, we do not have enough machines to test our large scale DVE systems because there are less than fifty SGI machines in our department. To make our application run on cross platforms, we implement our new DVE by using Java and VRML. This work has been done by a group of final year students supervised by Professor John C.S. Lui.

New languages like Java and VRML [36, 37](Virtual Reality Modeling Language) provide innovative methods for building virtual worlds. Java may provide the capability to migrate processes and objects across diverse platforms by using active messaging. VRML is a language for describing multi-participant interactive simulations - virtual worlds networked via the global Internet and hyperlinked with World Wide Web. VRML merely describes a 3D scene and methods for interacting with models. Though VRML 2.0 allows the use of Java to provide object behaviors, VRML itself does not provide a mechanism for communication among distributed users. Java and VRML complement each other like no other creative tools in existence. As both of these languages are object orientated, we can create an easy to use and maintainable software library. Other people may use our software library to develop their applications or plug-ins for our DVE system.

In our DVE system, the virtual environment is built by using VRML, and the

client and server is coded in Java. The primary advantage is that Java is a powerful, portable language. We may take advantage of its multithreading capabilities, both in the client and in the server. We might also use Java's synchronization capabilities and its automatic memory management.

Since Java classes are portable, both our client and our server are able to run on any platform that has a Java VM and networking capabilities, and clients running on one platform are able to communicate with servers running on a different platform.

The primary disadvantage of using Java is performance. An all-Java solution is too slow to handle a very large number of clients, and doing real time streaming audio is out of the question. However, the situation is improved because SUN released a new Just-in-Time Compiler(named Hot Spot).

As we began to implement the test bed for our DVE algorithms by using Java and VRML, I found that VRML worlds were much slower than state-of-the-art 3-D computer games. The speed difference maybe the tradeoffs that have been made between speed and generality. Computer games are extremely limited in their capabilities relative to VRML, as a result, they can use rendering techniques that are not applicable to the more general-purpose worlds that are being built in VRML.

However, some of the techniques used in computer games can be adapted in the construction of VRML worlds, in a way that does not require anything more than a standard VRML browser with support for scripting in Java. In previous chapter, I described the most important of these techniques: spatial partitioning.

6.3 Lessons learned

The most important lesson learned during this work is that generating interesting, detailed models is difficult. In developing our previous system VINCENT, approximately six "person months" were spent during creating multi-resolution models for the floor of Department of Computer Science and Engineering. Clearly, modeling tools must be developed that are more user-friendly and more automatic in order to make interactive visualization of complex 3D virtual environments a reality, even a virtual reality. Recent research in image-based rendering techniques [38] allow us to walk through a

real world environment created by real photos. How to integrate these techniques with our DVE system is one task worthy of study.

Chapter 7

Conclusion

In this thesis, we discussed the scalability problem in DVE and presented related techniques to solve this problem. To build a scalable DVE system, we have to resort to the multiple servers DVE architecture. Under the MSDVE architecture, there is a necessity to balance the workload and at the same time, reduce the server-to-server communication of a DVE systems. We formulate the partitioning problem and show that it is NP-complete in general. We then propose a computation effective partitioning algorithm so that we can quickly obtain a good partitioning policy $\mathcal{P}$. Our experiments show that our proposed algorithm can achieve significant reduction in both communication and computation cost. We also illustrate that we can adopt the partitioning algorithm to a virtual world wherein users move from one position to another position and the situation that avatars can dynamically join or leave the virtual world. We also investigate the possibility of paralleling the partition algorithm so as to obtain a partition policy for a large virtual world that allows many clients and servers.

With increasing video resolution, network bandwidth, and processor speed, DVE systems are becoming increasingly common in the scientific, industrial, and entertainment industries. People's expectations of DVE systems have also increased considerably. More entities are expected to be supported. Moreover, there is a growing demand to increase the realism and fidelity of DVE systems. Modeling and implementing real-time atmospheric effects, including wind, smoke, clouds, haze, rain and snow, will produce a flood of traffic that may exceed state update messages in current DVE systems. We believe that the techniques we discussed and the partitioning algorithm we proposed can enable the scalability of DVE systems.

Appendix A

Simplex Method

A linear programming problem [7, 8] is designed to identify a set of nonnegative variables minimizing a linear objective function subject to a set of linear constraints. A *standard form of the linear program* is:

$$\begin{aligned} &\textit{minimize} && z = c^T x && \text{(A.1)} \\ &\textit{subject to} && Ax = b && \text{(A.2)} \\ & && x \geq 0 && \text{(A.3)} \end{aligned}$$

where A is a given matrix of order $m \times n, m \leq n, c$ is an n-row cost vector, b is an m vector, and x is an unknown vector of n components. The superscript T denotes vector transposition.

Consider a linear programming problem in its standard form[A.1, A.2, A.3], and assume that the rank of A is m. If B is any nonsingular $m \times m$ submatrix of A, and N is the remaining submatrix of A, we can write (A.2) as

$$\left[B, N\right]\left[\begin{array}{c} X_B \\ X_N \end{array}\right] = b \qquad \text{(A.4)}$$

where X_B and X_N have m and $n - m$ components, respectively. It is assumed for convenience that B consists of the first m columns of A.

If $X_N = 0$, the solution to (A.4) is said to be a *basic solution* $X_B = B^{-1}b$ and the nonsingular matrix B is the basis. If , in addition, $X_B \geq 0$, we say that X_B is a *basic feasible solution.*

Theorem 2 If there is a feasible solution [satisfying constrains A.2 and A.3], there is a basic feasible solution. Furthermore, if there is an optimal solution minimizing z, there is an optimal basic solution.

This fundamental theorem is crucial to developing the *simplex algorithm*, the most powerful solution method for the general linear programming problems. There exist several distinct versions of this method and many numerical implementations. We used the primal simplex method in its revised version, which is the most common linear program solver.

An iteration of the revised simplex algorithm proceeds as follows:

Step 1. Given the basis B such that

$$X_B = B^{-1}b \geq 0$$

Step 2. Solve

$$B^T\lambda = c_B$$

for the vector of simplex multipliers λ.

Step 3. Select a column a_k of N such that

$$p_k = c_{Nk} - \lambda^T a_k < 0$$

We may, for example, select the a_k which gives the largest negative value of p_k. If

$$p^T = c_N^T - \lambda^T N \geq 0$$

stop; the current solution is optimal.

Step 4. Solve for y the system of equations

$$By = a_k$$

Step 5. Find

$$\theta = \frac{x_{0l}}{y_l} = \min_{1 \leq i \leq m, y_i > 0} \left[\frac{x_{0i}}{y_i} \right]$$

If none of the y_i's are positive, then the set of solutions to $Ax = b, x \geq 0$ is unbounded and z can be made an arbitrarily large negative number. Terminate the computation.

Step 6. Update the basic solution:

$$\bar{x}_i = x_i - \theta y_i, \qquad i \neq k$$

$$\bar{x}_k = \theta$$

where $\bar{x}_k$ is the new basic variable.

Step 7. Update the basis B and repeat from Step 2.

Step 1 assumes that an initial feasible basic solution is available. In order to find such a solution to $Ax = b, x \geq 0$, consider an auxiliary minimization problem

$$\min \sum_{i=1}^{m} \hat{y}_i \tag{A.5}$$

$$\begin{aligned} \textit{subject to} \qquad Ax + I\hat{y} &= b \\ x &\geq 0 \\ \hat{y} &\geq 0 \end{aligned}$$

where $\hat{y}$ is a vector of *artificial variables.* If we can find an optimal feasible solution to (A.5) such that

$$\sum_{i=1}^{m} \hat{y}_i = 0$$

then we have also obtained a basis yielding solution X_B. If (A.5) has minimum value greater than 0, there is no feasible solution to $Ax = b, x \geq 0$. Problem (A.5) is easy to solve using the same simplex method since it has an obvious initial feasible solution $x = 0, \hat{y} = b$, for $B = I$. This *two-phase method* is implemented in our program to solve the optimization problem. Phase I is used to find a feasible solution to $Ax = b, x \geq 0$, or to determine that no feasible solution exists. Phase II uses the basic feasible solution generated in phase I and solves problem A.1, A.2, A.3.

Bibliography

[1] John C.S. Lui, M.F. Chan, *Efficient Partitioning Algorithm for the Distributed Virtual Environment System*, 6th International Conference on Distributed Multimedia Systems(DMS'99), 1999.

[2] John C.S. Lui, M.F. Chan, K.Y. So, T.S. Tam, *Balancing Workload and Communication Cost for a Distributed Virtual Environment*, The Fourth International Workshop on Multimedia Information Systems (MIS'98), 1998.

[3] John C.S. Lui, M.F. Chan, T.F. Chan, W.S. Cheung, W.W. Kwong. *Virtual Exploration and Information Retrieval System: Design and Implementation*, The third International Workshop on Multimedia Information Systems (MIS'97), 1997.

[4] John C.S. Lui, Oldfield K.Y.So, Peter T.S. Tam, *Deriving Communication Subgraph and Optimal Synchronizing Interval for a Distributed Virtual Environment System.* IEEE Multimedia System'99.

[5] Hanqiu Sun, M.F. Chan, K.K. Hung, T.S. Tam, *Feedback Mechanisms in Assisting the Performance of Virtual Ping-Pong Game*, extended version of the paper appeared in Proceedings of 1998 Workshop on Computer Graphics.

[6] Chao-Wei Ou, Sanjay Ranka, *Parallel Incremental Graph Partitioning.* IEEE Transactions on Parallel and Distributed Systems, Vol. 8, No. 8, August 1997.

[7] Wayne L. Winston, *Introduction to Mathematical Programming - Applications and Algorithms.* Duxbury Press, 1995.

[8] Maciej M. Syslo, Narsingh Deo, Janusz S. Kowalik, *Discrete Optimization Algorithms.* Prentice-hall, 1983.

[9] T. Ballardie, P. Francis, J. Crowcroft. *Core Based Tree: An Architecture for Scalable Multicast Routing.* ACM SIGCOMM '93, pp. 85-95, September 1993.

[10] Jim Williams and Ming-Hsing Chiu, *Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation*, ACM Transactions on Modeling and Computer Simulation, July 1997, Pages 293-331.

[11] W. Broll. *Distributed Virtual Reality for Everyone: a Framework for Networked VR on the Internet.* Proceedings of the IEEE Virtual Reality Annual International Symposium, March 1997.

[12] D. E. Comer. *Internetworking with TCP/IP: Volume I, Principles, Protocols, and Architecture.* Prentice Hall, 1995.

[13] Sandeep Kishan Singhal, *Effective Remote Modeling in Large Scale Distributed Simulation and Visualization Environments*, Ph.D. thesis, Department of Computer Science, Stanford University, 1996.

[14] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1978.

[15] P.A. Bernstein, V. Hadzilacos, N. Goodman *Concurrency Control and Recovery in Database Systems*, Addison Wesley, 1987.

[16] S.E. Deering, D.R. Cheriton. *Multicast Routing in Datagram Internetworks and Extended LANs.* ACM Transaction on Computer Systems, Vol. 8, pp. 85-110, May 1990.

[17] V. Firoiu, D. Towsley. *Call Admission and resource Reservation for Multicast Sessions.* IEEE INFOCOM Conference, San Francisco, 1996.

[18] R. Muntz, J.R. Santos, and S. Berson. *RIO: A Real-time Multimedia Object Server*, ACM Performance Evaluation Review, ACM Press, vol. 25, no. 2, p.29-35, September, 1997.

[19] R.R. Muntz, J. Renato Santos, S. Berson. *A Parallel Disk Storage System for Realtime Multimedia Applications*, accepted for publication in the Journal of Intelligent Systems. *For more information about the Virtual World Data Server project, please refer to http://dml.cs.ucla.edu/vwds/projects.*

[20] M. R. Macedonia, D.P. Brutzman, M.J. Zyda, D. R. Pratt, P.T. Barham, J. Falby and J. Locke. *NPSNET: A Multi-player 3D Virtual Environment over the Internet*, Proceedings of ACM Symposium on Interactive 3D graphics, April 1995.

[21] O. Stahl and M. Anderson, *DIVE: A Toolkit for Distributed VR Applications*, Swedish Institute of Computer Science, SICS. *http://www.sics.se/dive/description.html.*

[22] R. C. Waters, J. W. Barrus. *The rise of shared virtual environments*, IEEE Spectrum, pp. 20-25, March, 1997.

[23] Michael R. Macedonia, Michael J. Zyda, *A Taxonomy for Networked Virtual Environments*, IEEE Multimedia'96.

[24] Calvin, James O., *Application Control Techniques System Architecture*, Technical Report RITN-1001-00, MIT Lincoln Laboratories, Lexington, Massachusetts, February, 1995.

[25] McDonough, J., *Doorways to the Virtual Battlefield*, Proceedings of Virtual Reality'92, pp. 104-114.

[26] C. Carlsson and O. Hagsand, *DIVE - a multi-user virtual reality system*, Proc. IEEE VRAIS'93, pages 394-400, Sept 1993.

[27] Michael R. Macedonia and Donald P. Brutzman, *NPSNET: A Multi-Player 3D Virtual Environment over the Internet*, Proceedings of the ACM - 1995 Symposium on Interactive 3D Graphics, 9-12, April 1995.

[28] Katherine L. Morse, *Interest Management in Large-Scale Distributed Simulations*, Technical Report ICS-TR96-27, UC Irvine.

[29] Hai Xie, Sami Tabbane and David J. Goodman, *Dynamic Location Area Management and Performance Analysis*, IEEE conference, 1993.

[30] Behrokh Samadi and Wing S. Wong, *Optimization Techniques for Location Area Partitioning*, 8th ITC Special Seminar on Universal Personal Telecommunications, OCT 1992.

[31] Thomas Allen Funkhouser, *Database and Display Algorithms for Interactive Visualization of Architectural Models*, Ph.D. thesis, Computer Science, UC Berkeley, 1993.

[32] Fuchs, Kedem and Naylor, *On Visible Surface Generation by A Priori Tree Structures*, SIGGRAPH '80, pp124-133.

[33] Macedonia, M., Zyda, M., Pratt, D., and Barham, P., *Exploiting reality with multicast groups: a network architecture for large scale virtual environments*, Proceedings of the 11th DIS Workshop on Standards for the Interoperability of Distributed Simulation, Sept. 1994, pp503-510.

[34] Van Hook, D., Calvin, J., Newton, M., and Fusco, D., *An approach to DIS scalability*, Proceedings of the 11th DIS Workshop on Standards for the Interoperability of Distributed Simulation, Sept. 1994, pp347-355.

[35] Russo, K., Schuette, L., Smith, J., and Mcguire, M., *Effectiveness of various new reduction techniques in ModSAF*, Proceedings of the 13th DIS Workshop on Standards for the Interoperability of Distributed Simulation, Institue for Simulation and Training, Orlando, Sept. 1996, pp587-591

[36] Bernie Roehl, Justin Couch, and Geoff Brown, *Late Night VRML 2.0 with Java*, Ziff-Davis Press, 1997.

[37] Rodger Lea, Matsuda and Miyashita, *JAVA for 3D and VRML Worlds*, New Riders Publishing, 1996.

[38] Y.F.Chan, M.H.Fok, C.W.Fu, P.A.Heng and T.T.Wong, *A Panoramic-based Walkthrough System using Real Photos*, Pacific Graphics '99.