A Personal Profiling Approach for Caching and Prefetching in Distributed Virtual Environment

Jimmy H.P. Chim

Master of Philosophy

The Hong Kong Polytechnic University

Hong Kong

2001

Abstract

Over the last decade, continuous evolution on both Internet and multimedia technology has turned the aspiration to Distributed Virtual Environment (DVE) across the Internet into a reality. A virtual environment usually contains a huge number of virtual objects represented by object models which are stored in a database server. When a user navigates through the virtual environment, the required data is transmitted from the server to the client machine for display through the underlying network. In this thesis, we aim at improving the performance of a DVE system across the Internet, both in terms of responsiveness and visual quality, under the existing constraint of relatively low bandwidth of the Internet.

We propose the *multi-resolution caching mechanism* for effective manipulation of client-side caching. Our proposed *multi-resolution caching mechanism* exploits the progressive characteristics of the recently proposed *multi-resolution modeling* to improve caching performance. The caching mechanism is further complemented by a *user-profiling based prefetching mechanism* to predict objects that will be accessed in the future. To quantify the performance of the proposed mechanisms, we perform a number of simulated experiments. The simulation results show that our proposed caching mechanism out-performs the traditional one and the prefetching mechanism is useful in reducing the response time of the system. To validate the simulation study, we implement a prototype. We conduct a series of experiments based on the prototype, along with appropriate quantitative analysis. From the results, we identify scenarios where our approach is beneficial, resulting in shorter response time, better cache hit ratio and reduced latency. These results demonstrate the effectiveness of our proposed mechanisms for use in DVE.

Acknowledgement

Thanks GOD for giving the wisdom and health to me so that I can accomplish this thesis.

I would like to express my sincere thanks to my supervisor, Dr. Hong-Va Leong and co-supervisors, Dr. Rynson Lau, and Dr. Antonio Si. Without their enlightenment, help, and encouragement, this research project would have not been completed. Also, I would like to thank our research group members, Mr. Boris Chan, Mr. Ken Lee, and Mr. Stanley Yau for their invaluable suggestions. They have spent much time brainstorming and sharing experiences and knowledge with me. I would like to thank Mr. Danny To for sharing with me his *multi-resolution modeling* programming library.

Last but not the least, I have to thank my family for their endless supports and forever loves. This thesis is dedicated to all of them.

The research life is an endless journey. The end of one stage is the beginning of another . . . .

Contents

Chapter 1

Introduction

1.1 Distributed Virtual Environment

Over the last decade, the continuous evolution of both Internet and computing technologies has turned Distributed Virtual Environment (DVE) across the Internet into a reality. Formally, a Virtual Environment (VE) is defined as an interactive three-dimensional (3D) computer-synthesized environment containing a significant number of interesting objects. Participants or users of the VE are able to perform navigation, exploration, manipulation and interaction with virtual objects in the VE as if they were in the real world [27]. Typically, each user accesses his/her own computer workstation or console as a user interface to the content of the VE by controlling a hypothetical virtual viewer located within the VE. These environments aim to provide users with a sense of realism by providing realistic 3D graphics to create an immersive experience. If images are rendered smoothly and rapidly enough, the illusion of real-time exploration of a virtual world can be achieved.

A DVE system is different from a traditional VE system in the way that users are not limited to accessing locally available VE only. By contrast, users are able to explore various remote virtual environments which may be located geographically apart via the underlying network. A DVE system also allows different users to share and interact within a common VE. Each VE contains a huge number of geometric models which will be stored and maintained by a database server. In a typical DVE session, data describing the environment is transmitted from remote database servers to the user's local machine. The data will then be stored, processed and rendered into images for display. Users should not be noticeable of whether he/she is accessing a VE or a DVE.

The use of DVE can shorten the distance between geographically boundaries. By using DVE, numerous kinds of application can be realized. Virtual museums and virtual art galleries, for example, visitors can examine different valuable collections of artworks, sculptures and antiques by using his/her computer without the need of traveling thousand-miles apart. Virtual libraries enable searching of information in a user friendly manner. Through virtual shopping malls, users are able to virtually touch the goods before purchasing them. Virtual prototyping can be developed using DVE. It can reduce the need for costly and timely physical mock-ups. In addition, DVE is beneficial to the development of scientific visualization for scientists at different locations by enabling them to enter into the world of data, to perform visual simulations and experiments, and to circumvent the time consuming model building task. The applications of DVE are not limited to the above mentioned examples. Other applications, such as tele-operations, tele-robotics, multi-user games and virtual training, can also be constructed using DVE.

1.2 Motivation

DVE systems, such as BrickNet [76], DIVE [7], MASSIVE [29], NPSNET [21] and SIMNET [5], focus on supporting large number of users sharing the VE. The content of the VE is usually replicated or pre-distributed at the client machine prior to the start of the application. These systems do not consider time-critical distribution of the VE across the network. The scalability and complexity of the VE are usually restrictive as the geometric primitives used are simple and limited in number. For example, several boxes are used to represent a human. This could prevent excessive transmission time across the network (as shown in Figure 1.1). More complex systems are only affordable by universities, large military or industrial institutions which are equipped with high-speed local area networks and powerful high-end graphics workstations.

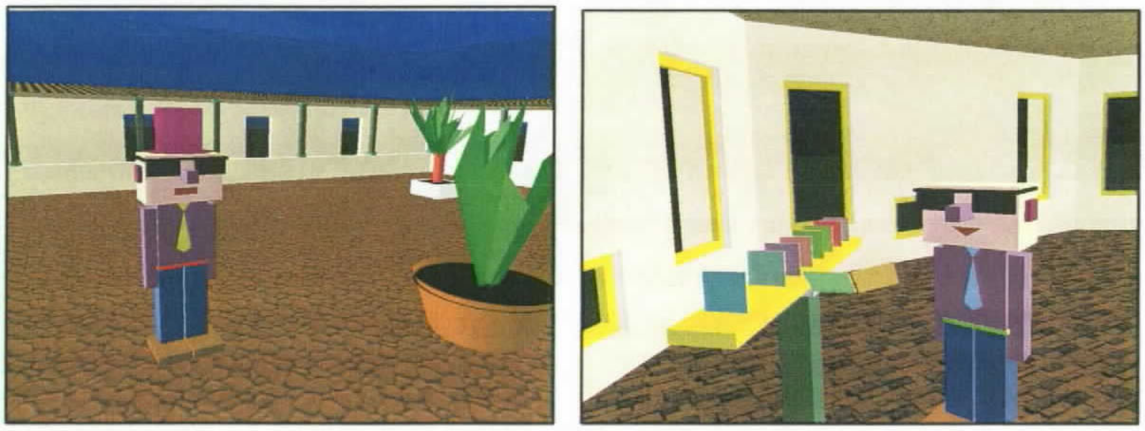


Figure 1.1: DVE example from DIVE.

In order to increase the realism of a DVE system, it is necessary to use a large number of geometric objects with good details. However, this will make the VE too large to be transmitted across the network. One associated problem is that the content of the VE may exceed the capacity of the computer's local storage. The problem is even worse when the network connections are notoriously slow, e.g., the Internet, and long transmission delay means that the systems are unsuitable for interactive use. Given today's personal computer systems with high-speed CPUs, fast system buses and slow Internet connections, it is very reasonable to assume that the network is the most constrained resource of the system.

To support DVE across the Internet, it is desirable to devise a solution for efficient distribution of geometric objects with minimal network bandwidth consumption and avoid the complete replication of a VE at each client machine. It is also indispensable to provide interactive frame rate and with good image quality, so that users can easily be immersed into the virtual world. The ultimate goal is to allow multiple users to share a common VE across the Internet.

1.3 Problem Statement

Large, detailed VEs are far too complex to be transmitted across the Internet. This makes provision of realistic images at interactive frame rates on currently available personal computer systems with Internet connections infeasible. A VE usually contains a tremendous amount of geometric virtual objects. The size of a VE database may contain gigabytes of data and thus it may take an extremely long period of time in transmitting the whole VE across the Internet. Such a large VE is inherently not favorable for transmission across the Internet.

In order to support interactive exploration and navigation of such a large VE across the Internet, a system must be able to request, store and render only a small portion of the VE at runtime. As a hypothetical viewer controlled by a user moves through the VE, the relative positions and orientations of objects may change with respect to the viewer. Some objects within the VE become visible while others are not; some appear larger and some smaller, etc. Such changes should be reflected into rendered images in a timely fashion.

To support interactive real-time rendering of a VE, most existing systems use multiple geometric description, called *Level-of-Details* (LOD), for representing virtual objects. This can reduce the rendering time required for distant virtual objects [7, 21, 24, 66]. However, this method greatly increases the network bandwidth consumption because extra amount of data is used to represent a single object. The amount of storage required to maintain these virtual objects however also increases. Thus, it is essential to employ another modeling method, if any, with minimal network bandwidth consumption for DVE systems across the Internet.

The challenge is to reduce the network bandwidth consumption and cache storage requirement while maximizing the frame rate and image quality for DVE systems across the Internet.

1.4 Basic Approach

To overcome the problem of large network bandwidth requirement of a DVE system, we propose a *multi-resolution caching mechanism* and a *user-profiling based prefetching mechanism* to effectively cache and prefetch virtual objects at the client machine. Other techniques employed include *Multi-resolution modeling* and *Object Scope* and *Viewer Scope.*

The choice of a modeling method for virtual objects is very important for preventing the use of redundant information in representing a single virtual object. Virtual objects must be modeled in good details, but in a compact form so as to reduce the amount of storage required and the amount of time required for transmitting them across the network. A compact modeling form of an virtual object also has the benefit of fast retrieval from secondary storage, both at the server and the client sides. Moreover, this can reduce the rendering time and increase the frame rate which can then increase the realism of a DVE system. However, overly compact virtual objects could increase the overhead in compression and decompression. The recently developed *multi-resolution modeling* [33, 42] technique, which facilitates progressive transmission of objects with only minimal overhead and network bandwidth consumption, could be employed here.

As a VE often consists of a huge number of object models, the size of the object database is generally very large, e.g. several gigabytes of data is not uncommon. The amount of data to be processed by the client machine may be far beyond its capability. As a result, extra time and processing are required for data manipulation and image rendering. The response time would then be increased. In order to reduce it, we must identify a small portion of the VE to be viewed by the client. One popular solution to this problem is based on *area of interest* (AOI) of viewers [21, 52, 66]. AOI defines how far a viewer can see. We adopt and further enhance this method by considering also sizes of virtual objects. We refer to the original method as *Viewer Scope* and the enhancement as *Object Scope.*

To reduce the amount of data being requested and transmitted across the Internet, caching of suitable objects with high affinity is commonly done. As the cache storage is usually limited in size, a replacement policy is used to discard objects from the cache when the storage is exhausted. Although caching is commonly used in many networked applications, a general conclusion on the performance of different replacement policies cannot be made. Rather, the semantics of data access is more important in defining the replacement policy [72]. We propose the use of *multi-resolution caching mechanism* in this environment. Caching is commonly complemented with prefetching to improve caching performance [8]. For this purpose, We propose a *user-profiling based prefetching mechanism.* In short, a separate personal profile capturing the characteristics of individual user's walk pattern is maintained at the client machine. This profile will then be used for predicting the future movements of the user. In this way, the data can be downloaded in advance.

To evaluate the proposed mechanisms for use in a DVE across the Internet, a series of simulated experiments has been performed. The results show that our proposed mechanisms out-perform traditional mechanisms. A system prototype has also been developed to further verify the proposed mechanisms under a physical environment.

1.5 Outline of the Thesis

In this chapter, we have briefly looked at Distributed Virtual Environment and the major problems associated with DVE across the Internet. We have also discussed the motivation, the problem statement and the basic approach of this dissertation. The rest of the thesis is organized as follows:

- Chapter 2 presents a review of related works including the origin of DVE, Network Design, Real-time Rendering, Visibility Processing, Modeling Methodology, Caching Mechanism and Prefetching Mechanism.
- Chapter 3 gives a brief description of the *multi-resolution modeling*. The idea of *Viewer Scope* and *Object Scope* will also be presented in this chapter.
- Chapter 4 presents our proposed *multi-resolution caching mechanism* for maintaining objects for possible future use in local storage. A *user profiling based prefetching mechanism*, which is complementary to the caching mechanism are also presented for further improvement.
- Chapter 5 gives a description of the simulation model and results for evaluating the proposed caching and prefetching mechanisms.
- Chapter 6 describes the system prototype design and implementation, as well as evaluation experiments and results.
- Chapter 7 concludes the thesis and outlines a summary of future work.

Chapter 2

Background

2.1 Origin of DVE

A distributed virtual environment (DVE) is a real-time simulation of a real or imaginary world where users navigate and interact with one another within it [30]. The construction of DVE has long been a challenge across several research disciplines. The ultimate goal is to create an immersive VE enabling multiple users, which may be separated across geographical boundaries, to participate, meet and interact with one another within the virtual environment. Remarkable DVE systems developed by Military Institution include *SIMNET* and *DIS* and academic DVE systems include *NPSNET*, *DIVE*, *BrickNet*, *MR Toolkit Peers Package*, and *MASSIVE*.

2.1.1 SIMNET

The Simulator Network (SIMNET) [5, 55] was one of the first DVE system developed by the Department of Defense (DoD) of the US Government. The project began in 1983 and was delivered to the US Army at 1990. The objective of the SIMNET project was to develop a relatively low-cost DVE for battlefield simulation and training. In SIMNET, the VE is modeled as a collection of objects interacting with each other via different events. Object representing either a vehicle or a weapon system can interact across the network and each is usually managed by a single machine. Object sends information about state changes, such as position and orientation, and the *dead reckoning* algorithm is used to extrapolate the states of different objects. SIMNET relies on a high bandwidth and low latency network infrastructure and generates images using special purpose Delta Graphics image generator hardware. The image generator uses the pre-distributed local copy of the terrain database and small number of runtime-distributed dynamic objects for rendering.

2.1.2 DIS

The Distributed Interactive Simulation (DIS) is derived from SIMNET and has been adopted as the IEEE 1278 standard at 1993 [36]. The DIS standard defines 27 different Protocol Data Units (PDU) for information exchange among different entities of the network. Examples of the PDU include, the Entity State PDU for position, orientation and velocity changes, the Fire PDU for firing of a weapon and the Collision PDU indicating that a vehicle has a collision with something. By using DIS, any computer capable of reading and writing DIS PDUs can fully participate in a DIS environment. The objective of DIS is to define an infrastructure for linking simulations of various types at multiple locations to create realistic, complex, virtual worlds for the simulation of highly interactive activities. This allows different systems and platforms to interoperate.

2.1.3 NPSNET

The NPSNET [21, 51] is developed by the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School. The first version, NPSNET-I, was debuted at 1990 for reading and rendering of SIMNET terrain database. NPSNET-II and III were utilized to explore better and faster rendering methods, as well as to extend the size of the terrain databases possible. The project NPSNET-IV started at 1993. It incorporates the DIS protocol, IP Multicast using dedicated Multicast Backbone (MBONE) and heterogeneous parallelism to develop large-scale VE. It can be configured by the user as a simulator for an air, ground, nautical, or virtual vehicle or human.

2.1.4 DIVE

The Distributed Interactive Virtual Environment (DIVE) [7, 30] by the Swedish Institute of Computer Science (SICS) is one of the earliest academic DVEs developed. DIVE uses different process groups to simulate a large shared memory over a local or wide-area network. It uses a distributed, fully replicated database which is similar to SIMNET and DIS-compliant systems. Each participant possesses a copy of the distributed database and propagates changes to other participants with reliable multicast protocols. One of the major contributions of DIVE is that its entire database is dynamic and it allows adding new objects and modifying existing database in a reliable and consistent fashion. DIVE uses reliable multicast protocols and concurrency control via a distributed locking mechanism to accomplish database updates which increases the cost for communications.

2.1.5 BrickNet

BrickNet [76] was initiated in early-1991 at the Institute of Systems Science of the National University of Singapore. BrickNet does not have a replicated database model as SIMNET, DIS and DIVE. Instead, it partitions the virtual environment among various VE clients. Requests for objects are mediated by servers and interactions on distant objects are accomplished by an object-request broker on a server. BrickNet is aimed at collaborative design environments in which a complete design task is distributed among multiple client workstations. The primary contribution of BrickNet is that it exploits the client/server architecture and each client does not require to hold a complete copy of the VE database.

2.1.6 MR Toolkit Peers Package

The MR toolkit Peer Package (MRTPP) [70] is an extension of the Minimal Toolkit (MR) Toolkit [71] developed at the Department of Computer Science of the University of Alberta. MRTPP allows multiple independent MR applications to communicate application dependent data or device data across the Internet. In MRTPP, local copies of shared data are maintained in a distributed fashion. Since it maintains a complete graph connection topology, each MR process must open a connection when communicating with other processes. MR Toolkit only supports simple geometric objects.

2.1.7 MASSIVE

MASSIVE [29] stands for "Model, Architecture and System for Spatial Interaction In Virtual Environments" and is developed at the Department of Computer Science of the University of Nottingham. It is designed for virtual collaboration of multiple users within a shared VE. To reduce the overhead for object interactions, MASSIVE introduced the concept of *aura* and *nimbus*. An aura of an object defines the extent to which interaction with other objects is possible and it is expressed as a function of position and other object attributes. Interactions are only enabled when aura collision between two objects occurs. When two objects collide, the objects themselves are responsible for controlling these interactions. Nimbus is defined as a sub-space in the VE, in which an object makes itself available to others.

2.2 Communication Model

Network is one of the major components of a DVE system. The communication model being employed strongly affects the performance and the scalability of a DVE system. In order to render the VE which is visible from the user's hypothetical viewpoint, data representing the VE should be available locally at the client machine before rendering. Four major communication models are commonly used, which are *Broadcast*, *Multicast*, *Unicast* and *Client-Server*.

2.2.1 Broadcast

Most of the early DVE systems, such as SIMNET [5, 55] and VERN [3], are based on broadcasting (as shown in Figure 2.1). In this topology, a single source of transmission can reach all the hosts in the network and it has the advantage of easy implementation. However, broadcasting has a major disadvantage of inefficient use of network resources. This topology is only suitable for local network or subnetwork. Too much network traffic would be generated if it is used for inter-network communication. In addition, all hosts in the network are required to examine every packet even if the information is not intended for them. Great performance penalty would be incurred for the host because it must be interrupted for examining broadcasting packets. The network can easily be overloaded. Broadcasting is unsuitable for use in the Internet environment.

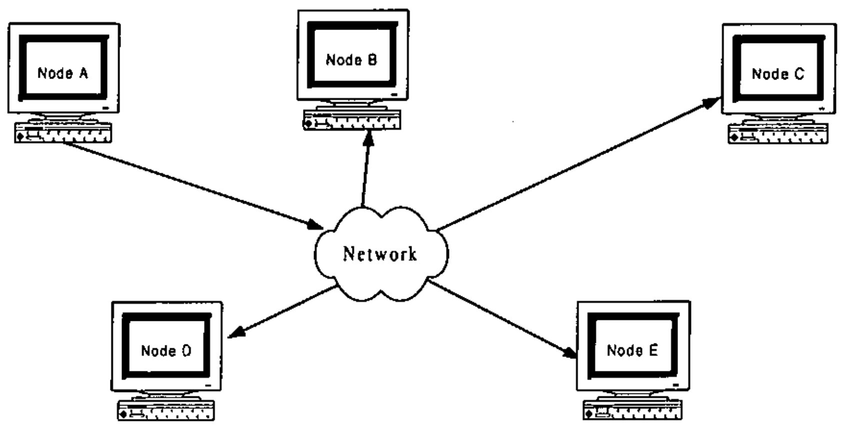


Figure 2.1: Broadcast communication model.

2.2.2 Multicast

Multicasting is used by NPSNET [21, 51] and DIVE [7, 30]. As shown in Figure 2.2, data is sent from one host to many different hosts, but not to everyone. The data only goes to clients that have expressed an interest in the data by joining a particular multicast group. Multicasting is better than broadcasting because not all hosts in the network would be interrupted during transmission and less network traffic would be generated. However, different users of a DVE would navigate through the VE under its own distinct path. Thus, data that is required by one user may not be needed by another. It is unnecessary to distribute data using multicasting under this context.

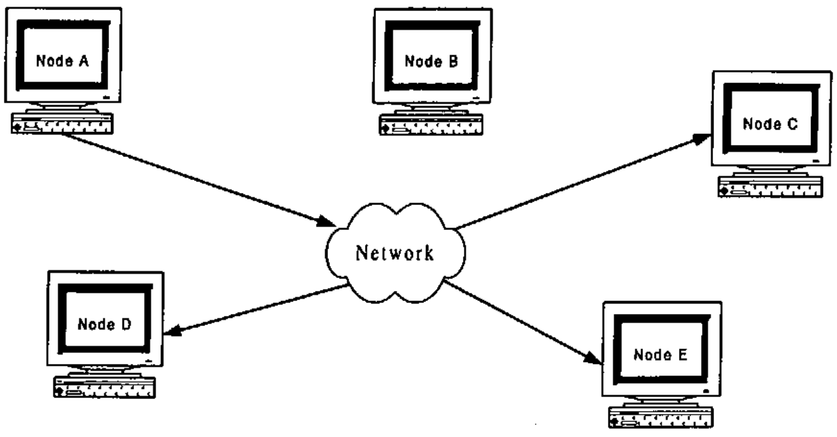


Figure 2.2: Multicast communication model.

2.2.3 Unicast

VEOS [4] and MR Toolkit [71] use unicast peer-to-peer design (as shown in Figure 2.3). For a large VE with good details, it is not feasible for each user to maintain a complete copy or replica of the whole VE. The whole VE may be too large for each client machine to maintain and the VE may exceed the storage capacity of a client machine. Thus, unicasting may not be suitable for a large VE.

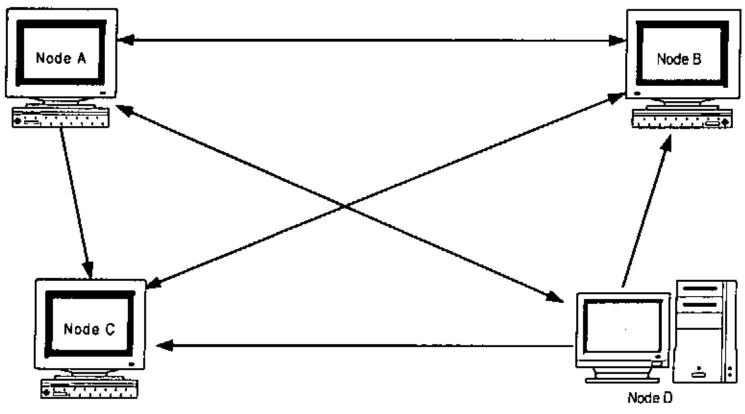


Figure 2.3: Unicast communication model.

2.2.4 Client-Server

BrickNet [76], DOOVie [2], RING [24] and WAVES [39] employ client-server architecture. A central server is used to maintain the content of a VE for client access. Communications between different clients can also be mediated by the server which can reduce the network load. While consistency can be made easier through a central server, the server will easily become a bottleneck. This can be solved by using hybrid system design like the one proposed by Funkhouser [25] which utilizes multiple servers and each server is responsible for a region of the VE (as shown in Figure 2.4). To further maximize the throughput, load balancing algorithm could be used to even out the workload of individual server [75].

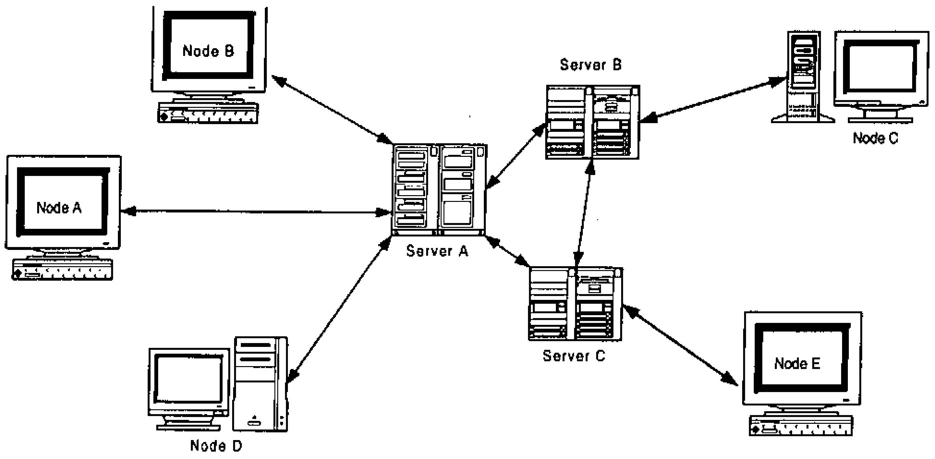


Figure 2.4: Client-server communication model.

2.3 Real-time Graphics Rendering

The cornerstone of an immersive VE system is the visual component of the system. Ideally, we would like to be able to generate images representing the VE that always match or exceed the limitations of human visual system. A sufficiently high frame rate is essential for producing a smooth and continuous motion within the VE. However, the ever-increasing demand of better image quality is usually bounded by the limited performance of image generators available. Thus, we have to trade off between the image quality and frame rate of the application by reducing graphical complexity. The two basic approaches include *Geometry-based Rendering* and *Image-based Rendering.*

2.3.1 Geometry-based Rendering

Geometry-based rendering is optimized and implemented in most graphics acceleration hardware. The rendering pipeline can be divided into several stages as follows:

- *Database Traversal* is typically performed on the CPU in which the geometric description of the scene is sent to the graphics pipeline (as shown in Figure 2.5).
- *Polygon Transformation* includes vertex transformations, lighting calculations and polygon shading.
- *Pixel Processing* involves operations such as depth buffer testing, anti-aliasing, texture-mapping and alpha blending.

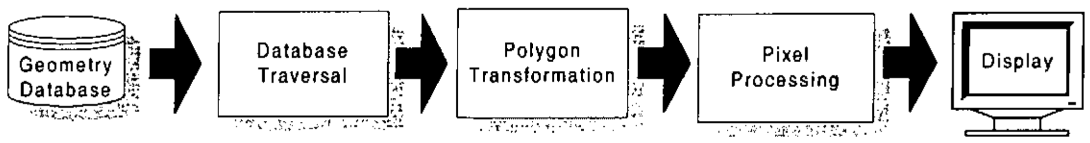


Figure 2.5: The Rendering pipeline.

The scene complexity is directly proportional to the number of geometric primitives involved. As the performance of an image generator is usually bounded by a fixed hard limit, it is desirable to reduce the scene complexity for geometry-based rendering. Two commonly used methods for achieving this goal are *Visibility Processing* (Section 2.4) and *Multi-Resolution Modeling* (Section 2.5).

2.3.2 Image-based Rendering

The relatively new approach of *image-based rendering* (IBR) exploits the advantage of finite image complexity as compared with the potentially unbounded geometric complexity of a scene. The display algorithms typically require least computational costs. The sources of images can be photographs as well as geometric models.

The concept of *imposter* is introduced by Maciel and Shirley [53]. In this method, a geometric object is preprocessed and rendered into a set of images based on various potential viewpoints of a user. The images are then texture-mapped onto different polygons. Schaufler improves this method by generating imposters dynamically during runtime [64]. In this method, the imposter is generated by rendering the object using the graphics hardware at runtime and then texture mapped onto a rectangular box replacing the original object. Schaufler [65] and Shade [69] extend the method using hierarchical image cache for accelerated walkthrough of very large polygonal scenes. These methods spatially partition the scene into different cells using a BSP tree or octree tree.

Talisman [77] is the first proposed hardware support for image-based rendering. The architecture employs several small image layers instead of the commonly used frame buffer. During rendering, these image layers are transformed and composed to simulate 3D motion. The rendering performance is improved by re-using the image layers. In Quicktime VR [9], an image is mapped cylindrically or spherically in real-time in order to simulate camera panning and zooming. This method works very efficiently for low-end platform.

To support IBR for DVE systems, two different approaches can be used, namely, *server-side IBR* and *client-side IBR*. In *server-side IBR*, the server machine transforms and renders the VE into a set of images either at the preprocessing stage or dynamically during runtime. Clients navigating the VE request images instead of geometric objects from the server for display. However, preprocessing of the VE at the server side prohibits dynamic objects within the VE and runtime generation of images will seriously increase the workload of the server, so that the real-time requirement of a DVE system cannot be fulfilled. By contrast, *client-side IBR* is more suitable for use in DVE. In this approach, the client still needs to request geometric objects from the server. The client machine can then render the VE using the IBR methods stated above.

2.4 Visibility Processing

A VE often consists of a large number of object models. The amount of data to be processed by the client machine may be far beyond its capability, i.e., the data cannot be fit into the memory and/or local storage of the client. As a result, the response time would then be much increased. In order to reduce the response time, we must limit the amount of data to be processed by the client for the application.

An approach based on the *area of interest* (AOI) of the viewer [21, 52, 66] is commonly used to limit the amount of data to be processed by the client. In this approach, if an object falls inside the AOI of the viewer, the object is considered visible to the viewer. The set of visible objects can easily be determined using this method. This method can also control the amount of data to be processed by a client during run-time. Since the AOI defines how far a viewer can see, the size of the AOI also determines the amount of data to be processed. If the amount of data to be processed is beyond the capability of the client, we can reduce the size of the

AOI such that the amount of data is within the capability of the client. Although these methods can quickly eliminate invisible objects, they do not consider the sizes of the objects. Hence, a large mountain located just outside the AOI of the viewer is considered invisible, while a tiny mosquito located just inside the AOI is unlikely to be visible to the viewer, but is considered for visibility. The former situation may result in sudden appearance of a large object, and the latter situation may result in a waste of processing time. Thus, it is necessary to consider the size of objects in this approach. This leads us into the definition of *Object Scope* and *Viewer Scope* (Section 3.4).

2.5 Multi-resolution modeling

The response time of a DVE system can be reduced by shortening the rendering time required for each object model. *Multi-resolution modeling* is commonly used and it is based on the fact that for two objects with the same dimensions, the distant object appears smaller than the nearby object after projection from the perspective of a viewer within the VE. Thus, most of the details of the distant object may not be perceived by the user. Hence, it is not necessary to represent the distant object at its full resolution. Instead, a simplified object at a resolution that is just high enough for the given viewing distance and/or viewing direction from the viewer can be used for rendering (Figure 2.6). Figure 2.7 and Figure 2.8 show the geometric complexity of the objects at different resolution. This would reduce the rendering time required to display the object model. Since a required model is used instead of the original model, it takes less time to transmit the simplified model across the network. There are many methods developed for generating multi-resolution models [17, 34, 67, 78] and can be classified into two main types, which are *level-of-details (LOD)* and *progressive mesh (PM)*.

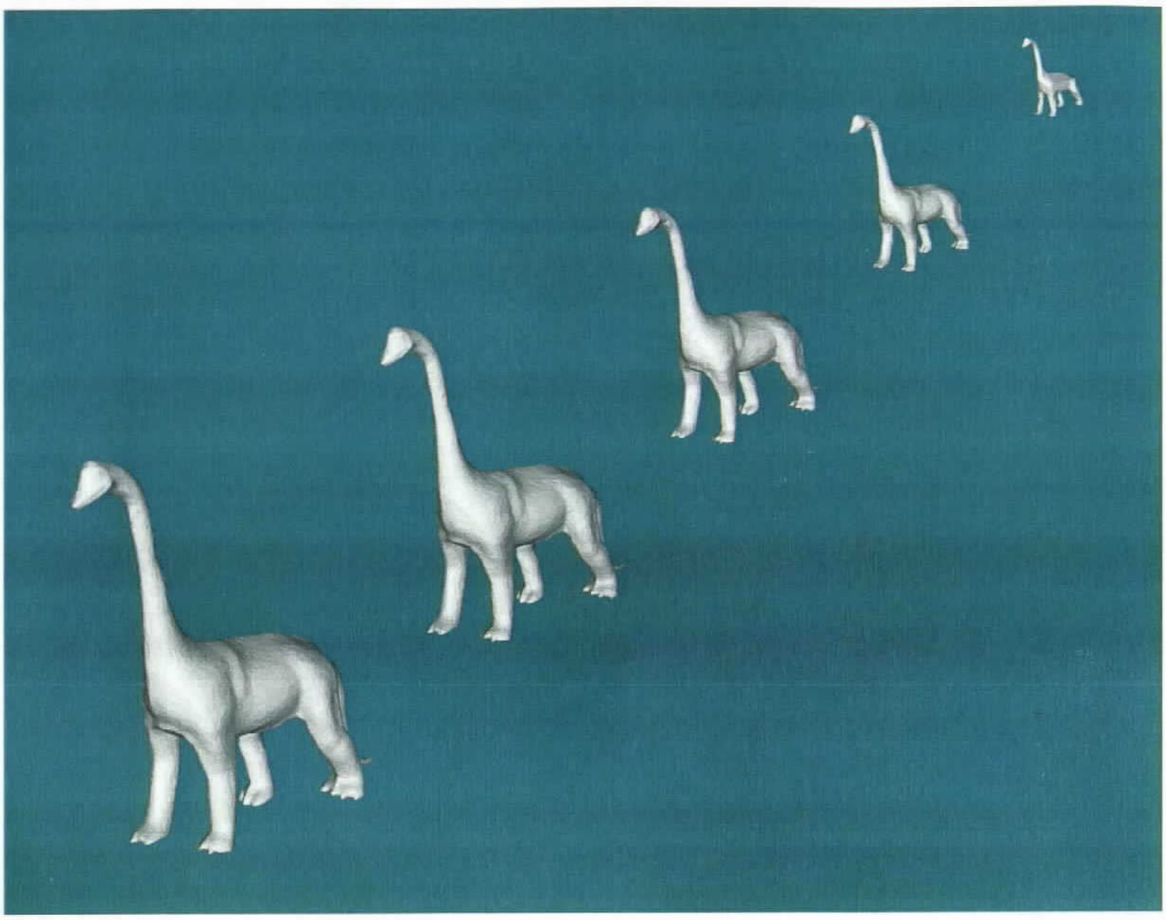


Figure 2.6: Multi-resolution modeling.

2.5.1 Level-Of-Details

Almost all existing distributed virtual reality applications [7, 21, 24, 66] use the method of *level-of-details (LOD)* for modeling. This method is also known as *discrete multi-resolution method*, in which a set of simplified key-models is pre-generated based on the original object model. Each key-model (also know as a LOD) is at a distinct resolution level which is lower than the original model. During the execution of the application, a particular key-model is chosen for each object in the scene based on the distance from the viewer to be used for rendering [14]. This would reduce

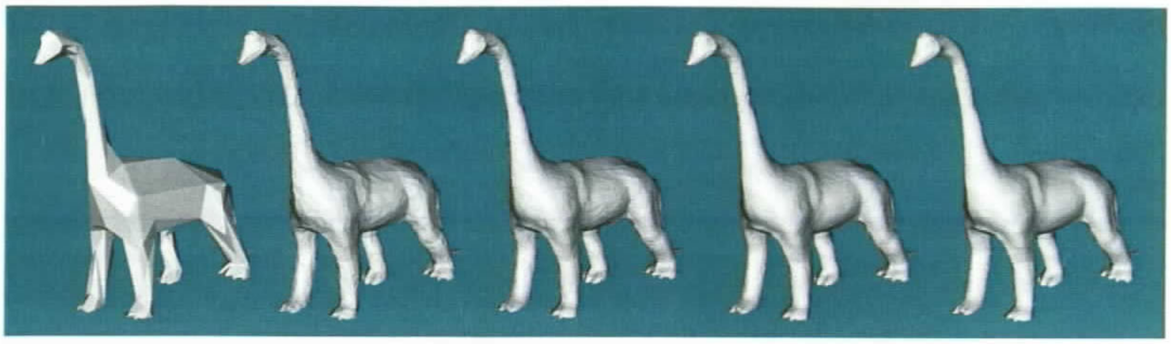


Figure 2.7: Multi-resolution modeling in shaded rendering.

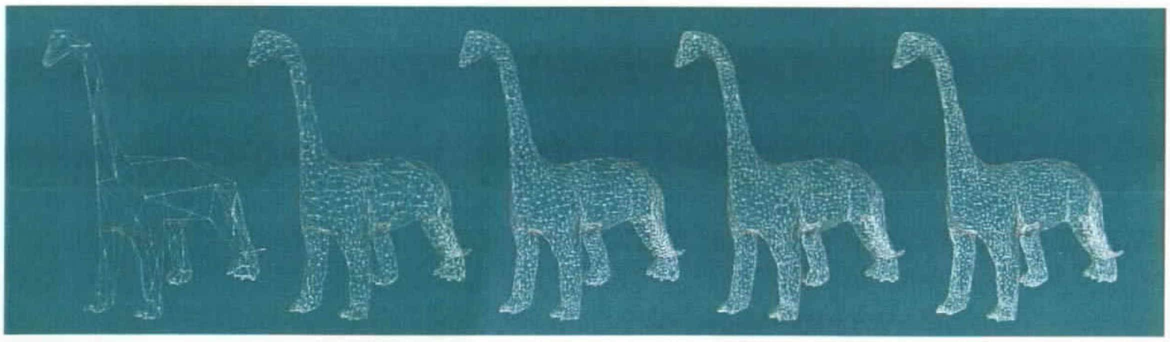


Figure 2.8: Multi-resolution modeling in wireframe rendering.

the rendering time required for the application. Although this method is fast and simple to implement, it has two major limitations. First, there is a sudden change in model resolution when the distance between the object and the viewer exceeds a pre-defined threshold value. A noticeable visual discontinuity would be observed by the viewer which would destroy the immersiveness of the application. Second, since each key-model of an object is independent of each other, the overall amount of data for representing the virtual world is increased. In a distributed environment, this will further increase the network bandwidth requirement and the average network transmission latency for the application because more data needs to be transmitted across the network.

Progressive Mesh

The method of *level-of-details* may be good for distributed virtual reality applications across a fast network. However, for DVE systems across the Internet, we need to use a method which should further reduce the transmission latency. Another multi-resolution modeling method called *progressive mesh* was recently proposed [33, 35]. This method allows progressive transmission of multi-resolution object model across the network and is fast for rendering. In this method, each object is modeled as a *base mesh* and a sequential list of *progressive records*. The resolution of an object model in the form of *progressive mesh* can be adjusted dynamically during runtime.

The method of *progressive mesh* is very suitable for use in DVE system across the Internet. Since this method is fast for rendering, there is no extra overhead in decompressing the object model as in the method of *geometric compression*. Besides, the network bandwidth requirement is kept minimal because no redundant information is used to model the virtual objects within the virtual world as in the method of *level-of-details*. Moreover, this method allows progressive transmission across the network, so that the user can still view the virtual world while the object models are being transmitted. This would provide the application with fast response time. Our proposed *multi-resolution caching mechanism* exploits the progressive characteristics of this method to improve caching performance.

2.6 Caching Mechanism

Caching has widely been used in various types of distributed systems, such as World-Wide Web [47, 61], client/server database systems [6, 16, 23], distributed file systems [28, 40, 45], distributed shared memory [46, 58] and mobile environment [8, 72, 74]. A similar scenario for applying caching is in the management of large graphical database that cannot be fit into the memory and have to be paged in from disk, e.g., NPSNET [21], and the work by Funkhouser et al. [26]. In order to achieve high performance in a distributed system, a multi-level caching mechanism could be employed by caching data in a client's local memory and/or local storage for possible future use. The establishment of client-side caching can reduce the network transmission latency incurred when transmitting large amount of data across the network. This would also reduce the cost of communication and the amount of network traffic generated. Moreover, a storage cache has a benefit of persistence and improved data availability at the client. This allows disconnected operations on the data that is cached in the client's local storage.

A caching mechanism is characterized by the caching granularity and the replacement policy used. There are different types of caching granularity classification. In [16], the caching granularity of *Page-based, Semantic-based and Tuple-based* are proposed for use in relational DBMS, while in [8], the granularity of *Object-based, Attribute-based* and *Hybrid-based* are proposed for use in object-oriented DBMS. Most of the traditional distributed systems are page-based [6, 22]. This is mainly because the server's storage is also page-based. The overhead for transmitting one item or a page is similar across a local area network. In general, a page-based mechanism requires a high degree of locality among the items within each page to be effective [18]. However, in a VE, virtual objects are represented using *object models* and are usually very complex and large in size, occupying possibly multiple data pages. The overhead required to transfer an object model (or simply object) in its entirety via the Internet with narrow bandwidth is very high. Furthermore, we might not always need to render an object at its full resolution (see Section 3.5). In addition, different viewers may have different moving paths in the VE. A physical organization that favors the locality exhibited by one viewer may result in poor locality for another. It is therefore inappropriate to transfer object models at the page level in this environment. Rather, it is necessary to consider caching at other granularity.

If a client can provide unlimited disk storage for caching, all objects within the virtual world can be cached in its local storage once they are received from the server. However, a more realistic situation is that the available storage for caching and the available pre-loading time are limited [56]. In order to retain objects for possible future use, a cache replacement policy must be employed. In [20], various cache replacement policies have been proposed and their suitabilities in a conventional database system have been examined, including optimal, WORST, Least Recently Used (LRU), CLOCK, and Least Reference Density (LRD). A general conclusion on the performance of the replacement policies cannot be made. In practice, the replacement policy is often approximated by the LRU policy in conventional caching [6, 22, 73]. In [8], it was shown that LRU policy is not appropriate in a context when the objects accessed by a client might change over time. Rather, the semantics of data access is more important in defining the replacement policy [8]. We, therefore, need to develop a more suitable replacement policy based on the semantics of accesses in a VE.

The use of caching mechanism can reduce the response time and the network bandwidth requirement for a distributed system. As the bandwidth of local storage is much higher than that of the Internet, it takes less time to transmit data from the local storage to the main memory if the data is cached.

2.7 Prefetching Mechanism

Although the use of caching mechanism retains some of the needed objects at the client, a portion of the required objects might still be needed from the database server. The transmission of data across the network would introduce some delay and increase the response time of the application. In order to further reduce the latency introduced, the technique of prefetching can be used. Whenever a client requests data from the database server, the database server should reply with the requested set of objects, as well as the set that should be visible to the viewer in the near future. By shipping the predicted set of objects together with the requested set, the response time could be minimized. Some examples for applying prefetching mechanisms include file systems [41, 63], distributed database applications [8, 15], broadcast disk [1] and distributed virtual reality applications [26, 66].

Prefetching data before they are needed could reduce latency introduced by the network communication, but prefetching of useless data would introduce additional delay and consume network bandwidth. Thus, a good prefetching mechanism should be used. The technique for prefetching depends on the type of application used and are based on some prediction algorithms. In [1], a prefetching mechanism based on a *Simple Prefetching Heuristic* is proposed, while in [60], a prefetching mechanism based on a pattern matching approach is proposed. In [15], a prefetching mechanism which is based on a data compression technique is proposed. In a distributed virtual reality application, a prefetching mechanism that is able to predict the next position of a viewer should be used. In [26], the prefetching technique is based on a visibility analysis that determines which cells should be visible to the viewer in the future, while in [66], a linear movement of the user is assumed.

In a DVE system, different users will exhibit different navigation paths. Due to the diverse nature of users' walk pattern, a prefetching mechanism that assumes a linear movement of users is not good enough. Instead, we propose the use of a *user-profile based prefetching mechanism* in which the history of individual user's movement would be stored as different user-profiles for prefetching. The user-profiles could then be used to predict the next location of the user more accurately.

Chapter 3

Multi-Resolution Modeling

3.1 Introduction

In a realistic virtual environment (VE), it is common to have an enormous amount of geometric objects contained within the environment. The computational and rendering costs needed for such a highly complex environment is considerable. Even with the use of powerful graphics workstations, a vast amount of delay can be introduced into the system. As a result, the VE may not be able to display at the interactive frame rate. The delay can detrimentally affect the user and destroy the immersiveness of the virtual experience. The solution to this problem is not likely to come from hardware development alone. Algorithmic solutions must also be employed in order to reduce the delay introduced by the rendering process.

To reduce the rendering overhead for a highly complex VE, the technique of *multi-resolution modeling* is attracting a lot of attention. This technique is based on the fact that distant objects occupy smaller screen areas after perspective projection than adjacent objects. Most of the geometric features of distant objects will not be perceived by the user. It would be a waste of computing power and graphics hardware in rendering distant objects in full details. Using simpler representation for distant objects can improve graphics performance without significant loss of visual fidelity, and thus enables real-time manipulation and navigation of a highly complex VE. Such reduction of geometric complexity using less number of vertices and polygons not only decreases the geometry processing at rendering, but also shortens the retrieval time from secondary storage, both at the server and the client.

3.2 Related Work

A *multi-resolution model* is a model which captures a wide range of levels of details of an object and can be used to reconstruct any one of those levels on demand. *Simplification* algorithms are employed to reduce the geometric complexity of a model automatically. In Section 3.2.1, we will first introduce different existing *simplification* algorithms for constructing *multi-resolution model*. Then, we will further discuss the two main types of *multi-resolution modeling*, which are *level-of-details* and *progressive mesh*, in Section 3.2.2.

3.2.1 Simplification

Simplification is the process of generating a simplified model with reduced geometric complexity from an original detailed model automatically. The ultimate goal is to reduce the number of polygons used to represent a model while preserving the features and appearance of the original model as much as possible. In order to support real-time rendering of models in a highly complex VE, it is necessary to employ a fast simplification algorithm. Simplification algorithms can be classified into *Clustering*, *Refinement*, *Decimation* and *Multi-resolution Analysis*.

Clustering

Vertex clustering is a simple method in which the vertices of the original model are grouped into clusters by coordinate quantization (round-off) [62] (Figure 3.1). This method ignores the topology of both the input and output models and can achieve arbitrary high compression ratio for any kind of geometry. To generate the vertex clusters, the bounding box of the input model is uniformly subdivided into different 3D cells. All vertices falling within a single cell are merged to form a new single representative vertex. A simplified model is then constructed from these representative vertices according to the original topology. This method can be regarded as a signal processing algorithm, in which the vertices are filtered, resampled and reconstructed. This method can greatly simplify the simplification process and is simple and efficient for implementation. However, the major weakness of this method is that it suffers from severe loss of detail and is unable to preserve important features due to the removal of high frequency details. This method can easily be generalized by using adaptive grid structure [49, 50].

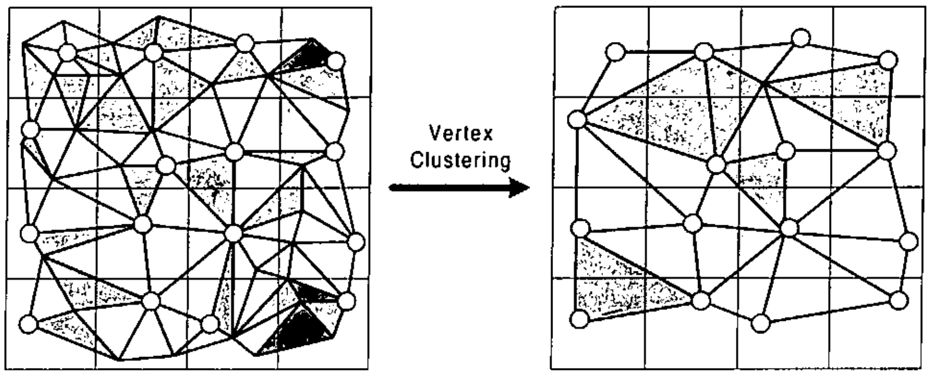


Figure 3.1: Simplification by Vertex Clustering.

Polygon clustering algorithms group nearby polygons and regenerate them into a simplified representation (Figure 3.2). Hinker et al. [32] proposed a single pass geometric optimization algorithm which identifies patches with nearly co-planar polygons. After the removal of interior vertices within these patches, the patches are retriangulated to remove the holes being generated. This algorithm is ineffective for surfaces with high curvature. Kalvin et al. [38] developed another algorithm that clustered polygons into *superfaces* for simplification within a given error tolerance. The algorithm segments the original model into different patches by selecting a polygon at random and combining neighboring polygons until the polygons can no longer be fit into a plane. The patches are then simplified by the Douglas-Peucker algorithm and the resulting patches are retriangulated into polygons.

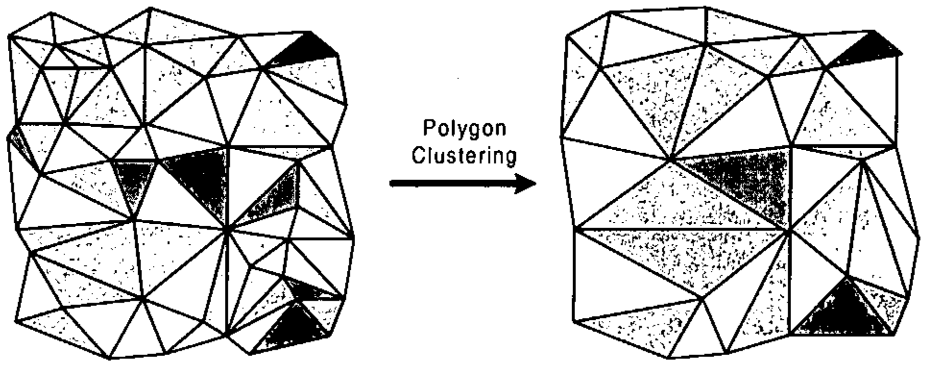


Figure 3.2: Simplification by Polygon Clustering.

Refinement

Refinement algorithms improve the accuracy and quality of an initially built minimal approximation of the original model by iteratively inserting vertices into the initial approximation. The refinement process is repeated until the error is smaller than a desired value or until it reaches a desired vertex count. DeHaemer et al. [17] uses an adaptive subdivision method to recursively subdivide each quadrilateral polygon into four while keeping the error within a given tolerance. The subdivision is performed such that the resulting error after the subdivision is minimum. Turk [78] proposes a method to optimize a model by distributing a number of vertices over the surface of the original model according to its curvature. The newly introduced vertices are retriangulated to generate the resulting model with the specified number of vertices. Although the method performs well for curved surfaces, it is slow and may fail to preserve the geometry of model with high curvature.

Hoppe et al. [34] optimizes a model based on the definition of an energy function that balances the conflicting goals of mesh simplification and error minimization. The energy function is used for minimizing the distance between the new approximating model and the original model. This method allows better preservation of geometry, but is extremely time consuming. He et al. [31] proposes a signal processing method for eliminating high frequency details. In this method, a three-dimensional voxel grid is placed over the input model and a low pass filter is then applied to the sampling grids. The Marching Cubes algorithm [48] is then applied to reconstruct a polygonal model from the volumetric representation. Although the method is very robust, the resulting model is still over-triangulated as a result of the marching algorithm and is unsuitable for models with sharp discontinuities.

Decimation

In contrast to the *refinement* algorithms, *decimation* algorithms employ a top-down approach to simplify a model. Starting from the entire original model, these algorithms iteratively delete vertices, edges, polygons or other geometric features from the model to form a simpler representation until the desired level of approximation is achieved. *Vertex clustering* and *polygon clustering* algorithms can be considered as restricted forms of *decimation* algorithms. Schroeder et al. [67] proposes a multiple-pass vertex decimation algorithm for simplification. In this method, all vertices that are not on a boundary or crease and have error below the threshold are deleted, and the neighboring polygons are retriangulated. This method is relatively faster, but is memory intensive and does not preserve geometric features well.

Hoppe et al. [33] proposed an idea called *progressive mesh*, which can be considered as an*edge decimation* algorithm. In this method, a sequence of *edge collapse* operations is selected (Figure 3.3). The vertices between the selected edges are iteratively merged into the same location. Typically, two triangular faces are removed for each *edge collapse* operation. This method allows fast rendering and rapid modification of resolution, which is extremely suitable for real-time graphics applications. In addition, this method allows progressive transmission across the network and generates a lossless, continuous-resolution representation of the original model. The topology of the original model can be preserved using this method. A similar algorithm based on *simplification list* has been developed independently and concurrently [35]. Our proposed *multi-resolution caching mechanism* is based on this method. We will further discuss these methods in Section 3.2.2.

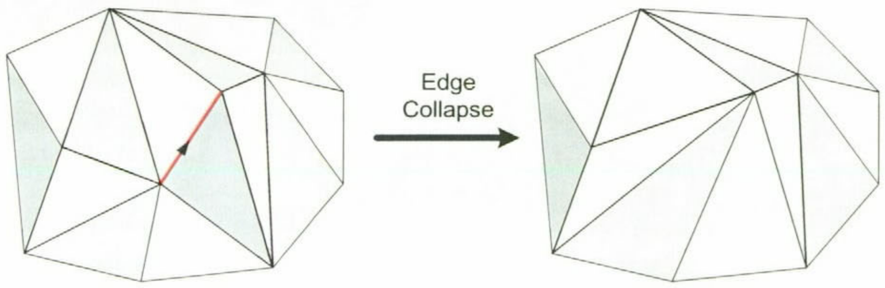


Figure 3.3: Simplification by Edge Collapse.

Multi-resolution Analysis

Multi-resolution analysis is a relatively new class of *multi-resolution modeling.* It involves structured mathematical decomposition of functions into multiple levels of representation [19]. Through the use of wavelet transforms, a hierarchical representation of functions can be obtained by iteratively decomposing the function into a coarser representation and a corresponding set of wavelet coefficients. The set of wavelet coefficients allows recovery of the original representation from the coarse representation. During reconstruction from the wavelet representation, sufficiently small wavelet coefficients can be discarded, resulting in a coarser approximation to the original data.

Although this method allows high compression ratio and progressive transmission of the original model, it suffers from some serious drawbacks. Before the wavelet representation can be built, the surface must be remeshed so that it has subdivision connectivity. This process alone introduces error into the approximation and is computational expensive. In addition, the topology of the model must remain fixed at all levels of representation. The wavelet representation is also unable to adequately preserve sharp corners and other discontinuities on the surface.

3.2.2 Model Representation

The representation of *multi-resolution modeling* can be classified into two main categories, which are *level-of-details* (LOD) and *progressive mesh* (PM).

Level-Of-Details

Level-of-details (LOD) was first introduced by Clark in 1976 [13] (Figure 3.4 and Figure 3.5). In this method, a set of distinct and independent LOD with reduced geometric complexity is generated for an object model at the preprocessing stage. Each LOD with reduced number of polygons closely resembles the original model. By using lower LOD for rendering distant objects, the rendering performance can be improved. Almost all existing distributed virtual reality applications [7, 21, 24, 66] use the method of *level-of-details (LOD)* for modeling. This method is also known as *discrete multi-resolution modeling.*

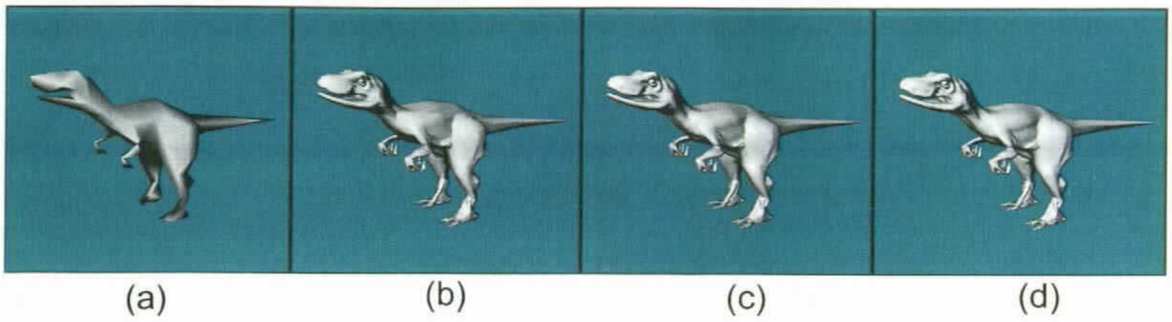


Figure 3.4: Level-of-detail in shaded rendering at increasing resolution.

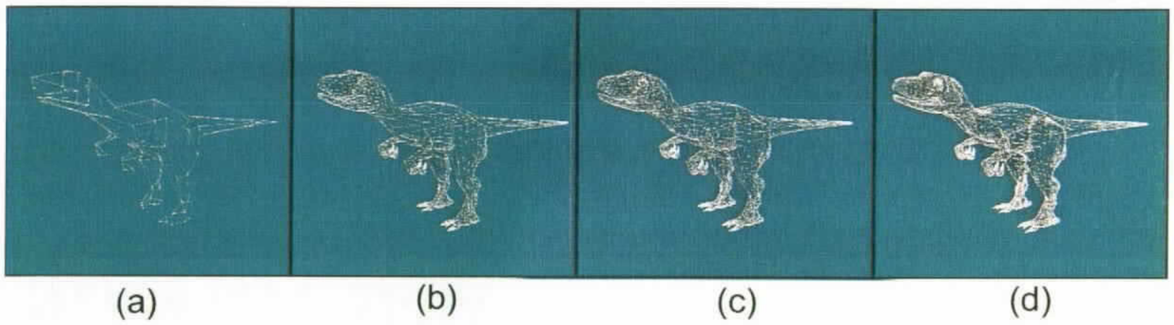


Figure 3.5: Level-of-detail in wireframe rendering at increasing resolution.

The selection of LOD is commonly based on the distance between the object and the viewer. This means that a higher LOD is used when the object is close to the viewer; conversely, it is substituted by a lower LOD as the object moves away. However, a noticeable flicker is commonly incurred when performing hard switching between two successive LODs. This is also known as the *popping effect* which may destroy the immersiveness of the application. To reduce the visual discontinuity during switching, one solution is to have a transition period during which a smooth interpolation between two successive LODs is performed to generate models of intermediate resolution [78]. However, this method further increases the cost of computation during the transition period because two models have to be processed at the same time. This method only works for LOD with well-defined geometric correspondence.

Although the use of LOD can reduce the rendering overhead of distant objects, this method has a great impact on network bandwidth consumption for DVE systems. The transmission time required for individual LOD across the network can be shortened when compared with the original model, but the overall network bandwidth requirement is increased. This method also has the disadvantage of extended storage consumption as each LOD has to be stored. Thus, this method is unsuitable for use in the Internet environment.

Progressive Mesh

The technique of *progressive mesh* (PM), which is also known as *continuous multi-resolution modeling*, is recently proposed [33]. In this method, each object is modeled as a *base mesh* and a sequential list of *refinement records*. The resolution of an object model can be modified at runtime by applying the operation of *edge split* or *edge collapse* to the object model using the list of *refinement records* as shown in Figure 3.7. The *base mesh* represents the original object model which is in its minimal resolution. If the list of *progressive records* is applied to the *base mesh* in order, the resolution of the *base mesh* is increased until the object model restores back to the original resolution. Conversely, if the list of *refinement records* is applied to the original model in reverse order, the resolution of the original model is reduced until it becomes the *base mesh*.

Progressive mesh has several advantages for use in DVE systems across the Internet when compared with *LOD*. First, no redundant information is transmitted across the network and the list of *progressive records* can be reused. The network bandwidth

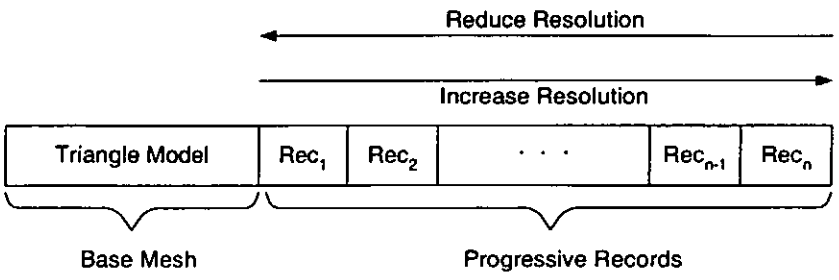


Figure 3.6: The structure of a progressive mesh.

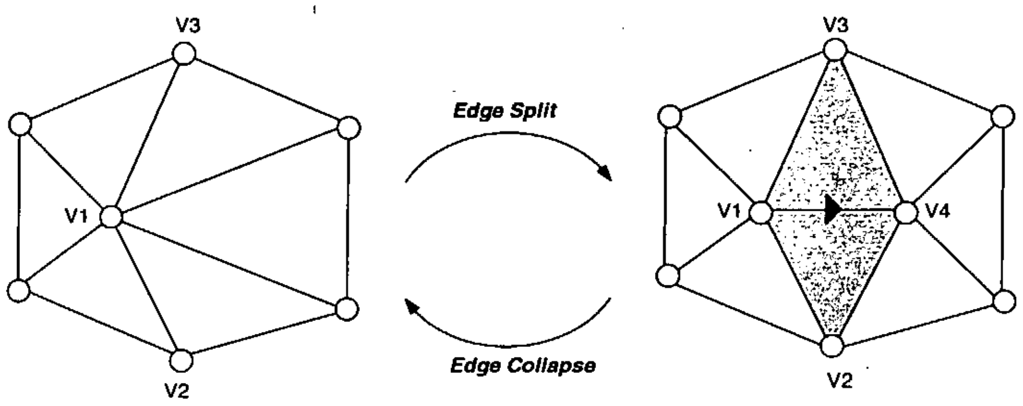


Figure 3.7: The operations of *edge split* and *edge collapse*.

consumption is kept minimal in this method. Second, this method allows progressive transmission across the network. The compact *base mesh* is transmitted first and followed by a stream of *progressive records*. Upon receiving these *progressive records*, the model can be reconstructed incrementally. The response time is then reduced significantly. Third, there is a smooth transition for increasing or decreasing resolution of an object model without any additional cost. As the object model is progressively refined or simplified, the changes are animated gradually. There is no noticeable visual discontinuity or *popping effect* to the user.

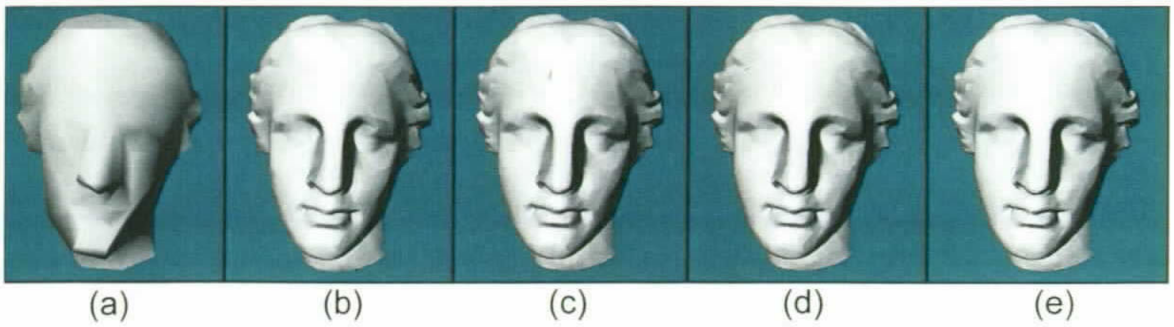


Figure 3.8: A progressive mesh in shaded rendering: (*a*) Base Mesh, (*b*) 25%, (*c*) 50%, (*d*) 75%, and (*e*) 100%.

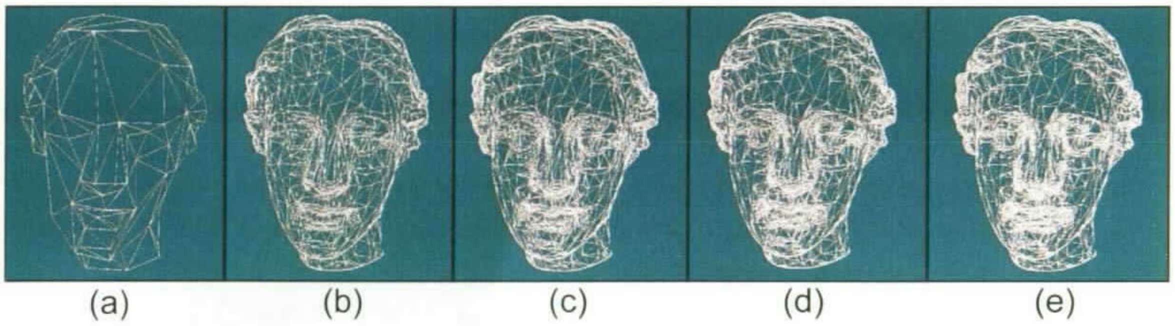


Figure 3.9: A progressive mesh in wireframe rendering: (*a*) Base Mesh, (*b*) 25%, (*c*) 50%, (*d*) 75%, and (*e*) 100%.

3.3 Overview of Our Method

In our DVE system for the Internet environment, each virtual object, o, is stored in the database server at its maximum resolution, $\hat{\mathcal{R}}_o$, in the form of a progressive mesh. Since the multi-resolution method we use here is based on *edge collapse*, the maximum resolution $\hat{\mathcal{R}}_o$ of o reflects the total number of edges that can be collapsed from its maximum resolution. Each object has a base mesh at its minimum resolution, $\check{\mathcal{R}}_o$. We say that the base mesh $\check{\mathcal{R}}_o$ of o represents o at resolution level 0, denoted as $\check{\mathcal{L}}_o$, which has a value of 0. Each progressive record increases the resolution level by one. Therefore, the maximum resolution $\hat{\mathcal{R}}_o$ represents o at the highest resolution, denoted as $\hat{\mathcal{L}}_o$, when all progressive records are applied.

The user of the system controls a hypothetical virtual viewer within the VE. Each object and viewer in the VE is defined with a *scope*. Intuitively, the *object scope* of an object defines the area within which the object is visible. It is roughly proportional to its size. The *viewer scope* defines the depth of sight of the viewer. A viewer can see an object only when the viewer scope intersects with the object scope. The goal is to eliminate the effect of sudden appearance of large objects and to reduce unnecessary overhead from transmitting and rendering small objects.

We denote the viewer scope for viewer, V, by $\bigcirc_V$ and the object scope for object, o, by $\bigcirc_o$. If the two scopes overlap, we need to determine the *optimal resolution* of the object for rendering. We denote the optimal resolution of an object, o, by $\mathcal{R}_o$ and its resolution level by $\mathcal{L}_o$. This optimal resolution of an object is determined according to the object's distance from the viewer, and its angular distance from the viewer's line of sight. If the object is rendered at a resolution higher than this optimal resolution, the additional details may not easily be noticeable to the viewer. By contrast, if the object is rendered at a resolution lower than this optimal resolution, the image quality of the rendered object as perceived by the viewer drops rapidly. Such a perceived image quality is called the *visual perception*. The relationship between visual perception and the resolution of an object is shown in Figure 3.10.

The interaction between a viewer and the VE is illustrated in Figure 3.11. In addition to the viewer scope, each viewer, V, has a *viewing direction*, $\vec{v}_V$, and a location, loc_V. The viewing direction defines the line of sight of the viewer. Given the location of a viewer, all virtual objects whose object scopes intersect with the viewer scope are rendered at their optimal resolutions. We refer to these objects as *renderable objects*. The *viewing region* defines the viewing angle of the viewer and is a sub-space of the viewer scope. All renderable objects within the viewing region are

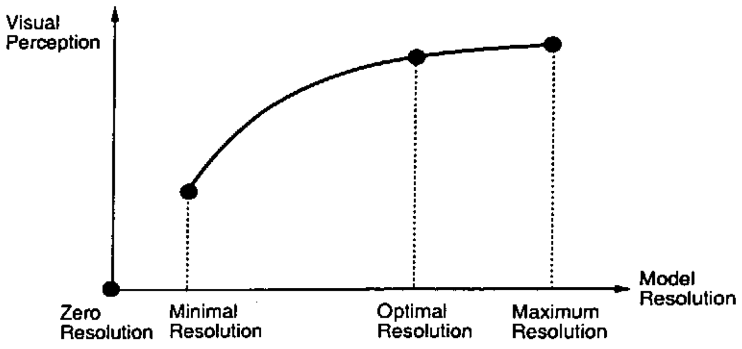


Figure 3.10: The effect of model resolution on the viewer's visual perception of an object.

visible to the viewer if they are not obscured by another object. We refer to these objects as *visible objects.*

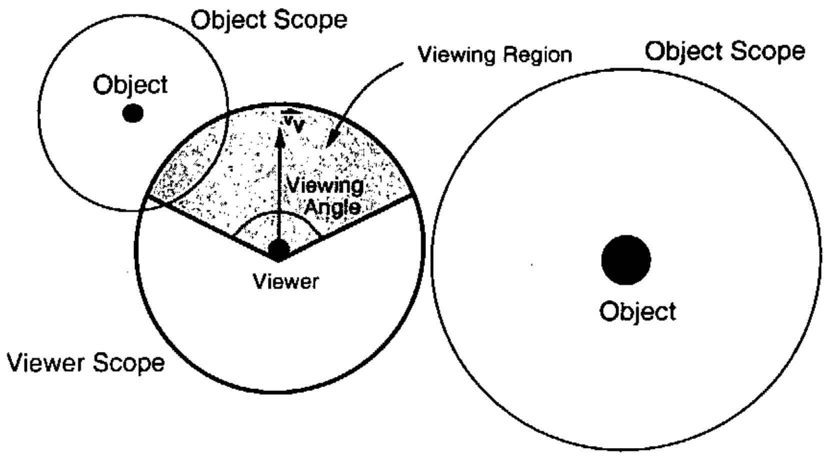


Figure 3.11: Interaction between a viewer and the surrounding objects.

During navigation within the VE, we continuously determine those objects in the VE whose scopes overlap with the viewer scope, i.e., the renderable objects. For each of those objects, the object model at its optimal resolution will be transmitted to the client, if it is not already cached in the local storage. The received models will be cached in the client's local storage. If there is not enough cache storage, we will throw away some progressive records of object models that are not likely accessed in the near future in order to accommodate the incoming models; i.e., we try to decrease the resolution of some existing cached objects.

We also attempt to further improve the performance of the DVE system by prefetching objects from the database server in advance. By predicting the next location and viewing direction of the viewer based on historical movements of the viewer, models of objects whose scopes overlap with the viewer scope with the viewer scope at the predicted location will be transmitted at their optimal resolutions to the client as well.

The advantages of our method can be summarized as follows:

- In [66], a discrete multi-resolution method is used for model transmission. Redundant information is sent through the network, when multiple models of the same object at different resolutions need to be transmitted. Our method applies the progressive mesh technique for model transmission. No redundant information needs to be sent across the network.

- The importance of an object is calculated based not only on the distance of the object from the viewer, but also on the size of the object concerned and the angular distance from the viewer.

- Our caching mechanism differs from conventional caching mechanisms [6, 22, 73] in that objects can be cached at multiple levels of granularity. Replacement is also based on object access patterns rather than conventional LRU scheme.

- We further improve the performance of the DVE system by predicting the future movement of the viewer and prefetch objects in advance.

3.4 Viewer Scope and Object Scope

To minimize the amount of data handled, most existing methods consider only the *area of interest* (AOI) of the viewer [21, 52, 66]. If an object falls inside the AOI of the viewer, the object is considered visible to the viewer. Otherwise, the object is considered too far away to be visible. Although these methods can quickly eliminate invisible objects, they do not consider the sizes of the objects. Hence, a large object located just outside the AOI of the viewer may still be visible to the viewer, but is considered as invisible, while a tiny object located just inside the AOI of the viewer is unlikely to be visible to the viewer, but is considered as visible. The former situation may result in a sudden appearance of a large object, and the latter situation may result in a waste of processing time.

To overcome this limitation, we generalize the AOI concept to both viewers and objects. We call them the *viewer scope* and the *object scope* [10]. The definition of the viewer scope is similar to the definition of the AOI. A viewer scope indicates the depth of sight of the viewer, i.e., how far the viewer can see. A viewer with good eye-sight or equipped with a special device may be able to see objects that are further away, and therefore may be assigned with a larger scope. A short-sighted viewer may only be able to see nearby objects, and therefore may be assigned with a smaller scope. The definition of object scope is different. An object scope indicates how far an object can be seen. A large object has a large scope, while a small object has a small scope. An object may be visible to a particular viewer only when its scope overlaps with the viewer scope. When the two scopes overlap, the distance between the object and the viewer, and the angle between the object and the viewer's line of sight can be used to determine the optimal resolution of the object. In general, a viewer may also be considered as an object and assigned an object scope in addition to the viewer scope. This object scope of the viewer will define how far the viewer can be seen by another viewer within the same virtual environment. This approach is somewhat similar to the one proposed by [29].

In our implementation, we define a scope as a circular region. Therefore, each scope (object or viewer) is characterized by a radius. We denote the radius of the object scope for object o, i.e., $\bigcirc_o$, as $r_{\bigcirc_o}$ while the radius of the viewer scope for viewer V, i.e., $\bigcirc_V$, as $r_{\bigcirc_V}$.

3.5 Optimal Resolution

There is no unique way to characterize the best resolution level for a group of objects contained within a VE, since it is hard to quantify human vision using a single mathematical equation. Heuristics based on distance, size, line of sight, velocity or combination of them are commonly used to determine the resolution for objects within the VE with respect to the viewer.

In our method, the optimal resolution of an object model is determined according to the *visual importance* of the object to the viewer. In [43], different factors that may affect the visual importance of an object have been identified. Here, we only consider two of those factors, which are relevant to the current context. The first one is the distance factor. If an object is far away from the viewer, the object may be considered as visually less important. The second factor is the line of sight. Studies have shown that when an object is located outside the line of sight, the viewer is unable to perceive much detail from the object [59, 79]. Degradation of peripheral visual detail can improve rendering performance and reduce perceptual impact. There are many eye tracking systems available for detecting line of sight [37]. Since most of these systems are still too expensive for the general public, some applications simply assume that the viewer's line of sight is always at the center of the screen.

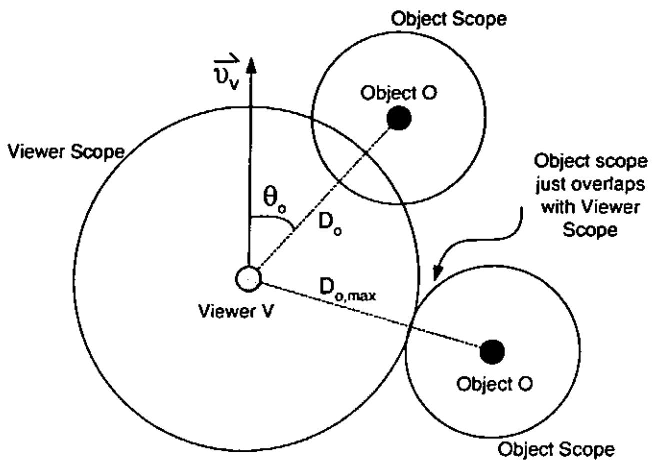


Figure 3.12: Visual importance of an object with respect to a viewer.

Figure 3.12 depicts the visual importance of an object, o, to a viewer, V. In the figure, D_o indicates the current distance of the object from the viewer, while $D_{o,max}$ is the distance between the object and the viewer when their scopes just overlap. Since a scope is defined as a circular region, $D_{o,max}$ is equal to the sum of the radii of the viewer scope and the object scope. The angular distance of the object from the viewer's line of sight, i.e., its viewing direction, $\vec{v}_V$, is denoted as $\theta_{o,V}$ or simply θ_o $(-\pi \leq \theta_o \leq \pi)$. The visual importance of o to a viewer can be defined with the following equation:

$$I_o = \left(\frac{D_{o,max} - D_o}{D_{o,max}}\right)^2 e^{-K_o|\theta_o|} \qquad (0 \leq D_o \leq D_{o,max}) \qquad (3.1)$$

The first term models the relationship between the distance of the object from the viewer and the visual importance of the object. The higher the value of D_o is, the lower the value of I_o will be. The second term models the relationship between the viewer's line of sight and the visual importance of the object. We employ an exponential relationship between them which is similar to the method proposed by Ohshima et al. [59] in modeling the decline in visual acuity with the line of sight. The higher the value of θ_o is, the lower the value of I_o will be. Because this relationship is not a linear one, a constant K_o is introduced for adjusting the decremental rate of the model resolution due to the increase in θ_o. In our implementation, we do not want the line of sight factor to dominate the distance factor. Hence, we use a small value of K_o. Note that we may include more terms in this equation to model other factors that affect the visual importance of the object such as the moving speed of the object. For the sake of clarity, we only consider the line of sight and distance factors here.

To incorporate this idea into the progressive multi-resolution method described in [33], the visual importance, I_o, is used to determine the optimal resolution of the object model. We use a linear mapping to define the optimal resolution, $\mathcal{R}_o$, of an object, o, as the product of its maximum resolution, $\hat{\mathcal{R}}_o$, and the visual importance, I_o, with respect to the viewer, i.e. $\mathcal{R}_o = \hat{\mathcal{R}}_o * I_o$. When the object scope just touches the perimeter of the viewer scope, the object is considered renderable to the viewer, and the base mesh of the object is used for rendering. This base mesh will provide the minimal resolution of the object. As the object moves closer to the viewer or to the viewer's line of sight, the resolution level of the object increases.

Determination of K_o

We would like to associate the viewing angle, $\hat{\theta}_v$, of the viewer in determining the value of K_o. Since K_o adjusts the decremental rate of model resolution due to the increase in θ_o and objects falling beyond $\hat{\theta}_v$ will not be visible to the viewer, it is then possible to determine the constant K_o according to $\hat{\theta}_v$.

Recall the visual importance equation (3.1):

$$\begin{aligned} I_o &= \left(\frac{D_{o,max} - D_o}{D_{o,max}}\right)^2 e^{-K_o|\theta_o|} \\ &= \left(1 - \frac{D_o}{D_{o,max}}\right)^2 e^{-K_o|\theta_o|} \\ &= (1-r)^2 e^{-K_o|\theta_o|} \qquad (where\ r = \frac{D_o}{D_{o,max}}) \end{aligned} \tag{3.2}$$

Let Φ_v be the volume of I_o within $\hat{\theta}_v$ and Φ_o be the volume of I_o beyond $\hat{\theta}_v$

Find K_o such that the Φ_v is maximum

$$\boxed{\max_{K_o} \left(\int\int \Phi_v dx - \int\int \Phi_o dx\right)} \tag{3.3}$$

$$\begin{aligned} \int\int \Phi_v\, dx &= \int_0^1 \int_0^{\hat{\theta}_v} (1-r)^2 e^{-K_o|\theta_o|}\,(r)\, dr\, d\theta \\ &= \int_0^1 [r(1-r)^2]\, dr \bullet \int_0^{\hat{\theta}_v} e^{-K_o|\theta_o|}\, d\theta \\ &= \int_0^1 (r - 2r^2 + r^3)\, dr \bullet \left[-\frac{1}{K_o} e^{-K_o|\theta_o|}\right]_0^{\hat{\theta}_v} \\ &= \left[\frac{r^2}{2} - \frac{2r^3}{3} + \frac{r^4}{4}\right]_0^1 \bullet \left[-\frac{1}{K_o}\left(e^{-K_o|\hat{\theta}_v|} - 1\right)\right] \\ &= \left(\frac{1}{2} - \frac{2}{3} + \frac{1}{4}\right)\left[-\frac{1}{K_o}\left(e^{-K_o|\hat{\theta}_v|} - 1\right)\right] \\ &= \frac{1}{12}\left[-\frac{1}{K_o}\left(e^{-K_o|\hat{\theta}_v|} - 1\right)\right] \end{aligned} \tag{3.4}$$

$$\begin{aligned} \int\int \Phi_o\, dx &= \int_0^1 \int_{\hat{\theta}_v}^{\pi} (1-r)^2 e^{-K_o|\theta_o|}\,(r)\, dr\, d\theta \\ &= \frac{1}{12}\int_{\hat{\theta}_v}^{\pi} e^{-K_o|\theta_o|}\, d\theta \\ &= \frac{1}{12}\left[-\frac{1}{K_o}\left(e^{-K_o\pi} - e^{-K_o|\hat{\theta}_v|}\right)\right] \end{aligned} \tag{3.5}$$

Substitute Equation (3.4) and (3.5) into (3.3):

$$\begin{aligned} f(K_o) &= \underset{K_o}{max} \left(\int\int \Phi_v dx - \int\int \Phi_o dx \right) \\ &= \underset{K_o}{max} \left(-\frac{1}{12K_o} \left[\left(e^{-K_o|\hat{\theta}_v|} - 1 \right) - \left(e^{-K_o\pi} - e^{-K_o|\hat{\theta}_v|} \right) \right] \right) \\ &= -\left[\frac{2e^{-K_o\hat{\theta}_v} - 1 - e^{-K_o\pi}}{12K_o} \right] \end{aligned} \tag{3.6}$$

$$\begin{aligned} f'(K_o) &= -\frac{1}{12} \left[\frac{\left(2\,e^{-K_o\hat{\theta}_v} - 1 - e^{-K_o\pi} \right) - K_o \left(-2\,\hat{\theta}_v e^{-K_o\pi} + \pi\, e^{-K_o\pi} \right)}{{K_o}^2} \right] \\ &= -\frac{1}{12} \left[\frac{2\,(e^{-K_o\hat{\theta}_v})(1 + K_o\hat{\theta}_v) - e^{K_o\pi}(1 + K_o\pi) - 1}{{K_o}^2} \right] \end{aligned} \tag{3.7}$$

Solve $f'(K_o) = 0$:

$$2\,(e^{-K_o\hat{\theta}_v})(1 + K_o\hat{\theta}_v) - e^{K_o\pi}(1 + K_o\pi) - 1 = 0 \tag{3.8}$$

By solving $f'(K_o) = 0$ using numerical analysis, the results of K_o against $\hat{\theta}_v$ can be obtained (Table 3.1). Human vision is limited to have a maximum viewing angle of 180° and 120° for stereo vision. In order to maximize the visual importance for objects within the stereo vision, we use 120° for viewing angle and 1.5317 for K_o in our *multi-resolution modeling* method.

$\hat{\theta}_v$	20°	40°	60°	80°	100°
K_o	9.6162	4.8080	3.2040	2.3935	1.8900
$\hat{\theta}_v$	120°	140°	160°	180°	360°
K_o	1.5317	1.2508	1.0134	0.7999	0.0

Table 3.1: Results of K_o against $\hat{\theta}_v$.

Visual Importance Distribution

Figure 3.13 shows the top view of the visual importance distribution within a viewer scope with K_o equal to 0.0 and 1.5317 respectively. In both figures, the viewer located at the center of the viewer scope. With K_o equal to 0.0, the visual importance value is the highest at the center of the viewer scope and the value decreases as the distance from center increases (Figure 3.13(a)). With K_o equal to 1.5317, the visual importance value is not only affected by the distance, but also by the viewing direction of the viewer.

Figure 3.14, 3.15 and 3.16 show the front, side and perspective view of the visual importance distribution respectively. With K_o equal to 0.0, the visual importance distribution is symmetric and is independent of the viewing direction (Figure 3.14(a), 3.15(a) and 3.16(a)). By contrast, with K_o equal to 1.5317, the visual importance distribution becomes view-dependent (Figure 3.14(b), 3.15(b) and 3.16(b)).

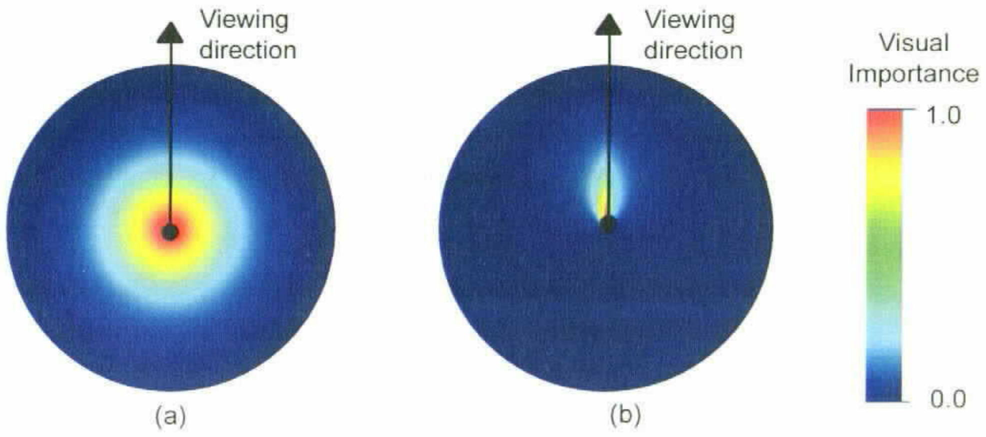


Figure 3.13: Top view of Visual importance distribution: (a) K_o = 0.0 and (b) K_o = 1.5317.

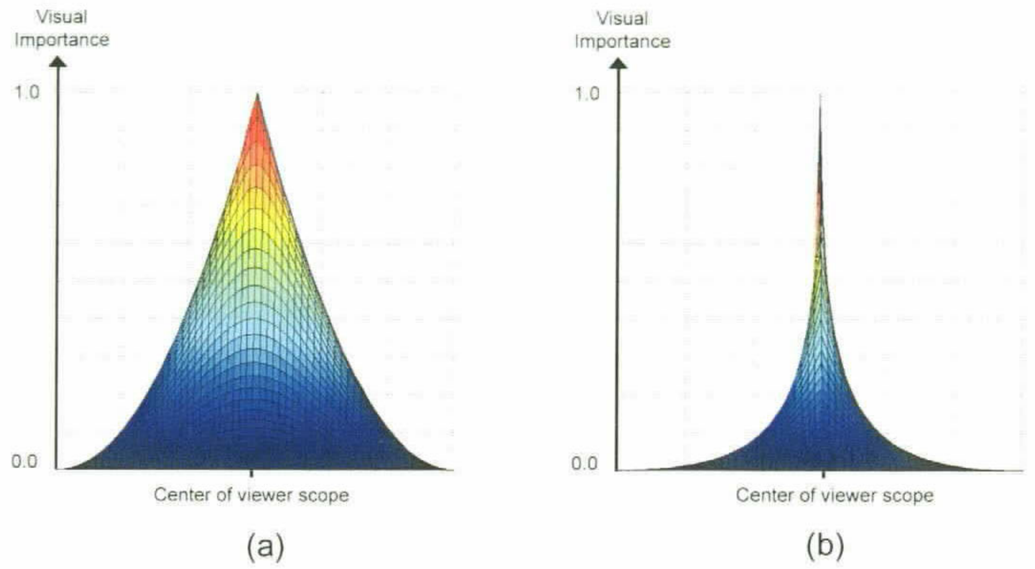


Figure 3.14: Front view of Visual importance distribution: (a) $K_o = 0.0$ and (b) $K_o = 1.5317$.

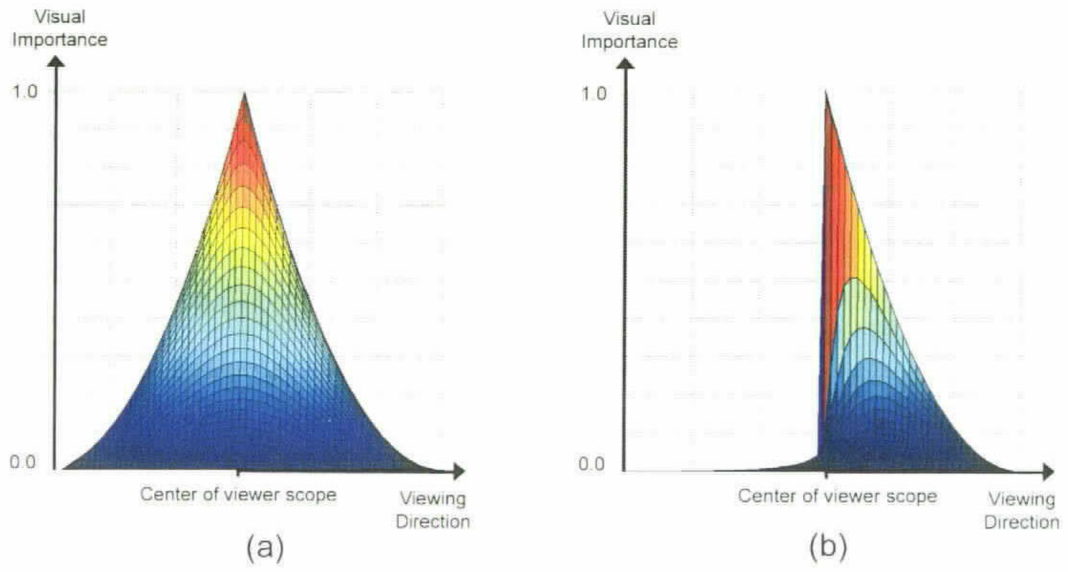
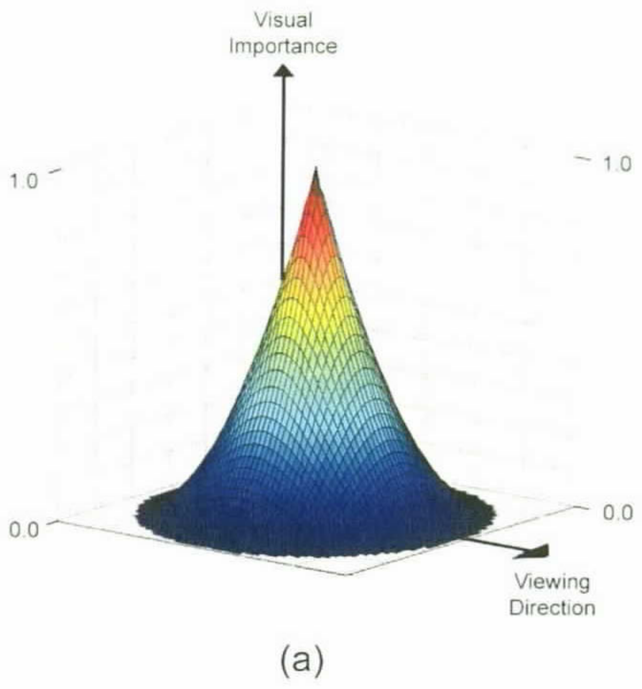


Figure 3.15: Side view of Visual importance distribution: (a) $K_o = 0.0$ and (b) $K_o = 1.5317$.



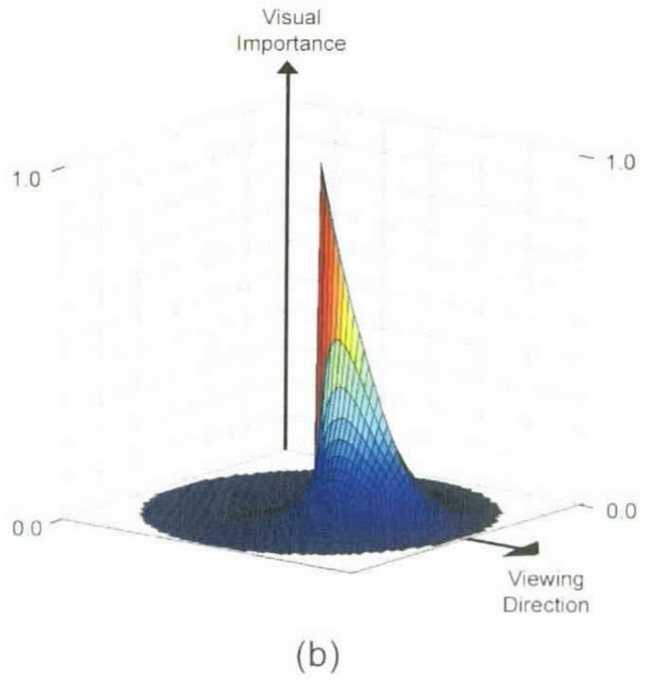


Figure 3.16: Perspective view of Visual importance distribution: (a) $K_o = 0.0$ and (b) $K_o = 1.5317$.

Chapter 4

Multi-Resolution Caching and Prefetching Mechanism

4.1 Introduction

In order to achieve a high throughput and scalability in client-server systems, it is essential to effectively utilize the computational and storage resources of the client machines. The architecture for this kind of client-server systems is known as *data-shipping* [16]. In a data-shipping architecture, data processing is mainly performed at the clients and the data residing at the server is replicated on demand at the clients. To reduce the overhead for data transmission across the network and the need for future interactions with the server, it is a common practice to *cache* the received data at the client's local memory and/or disk for possible future reuse.

When caching is incorporated into a data-shipping architecture, servers are mainly used to service cache misses and this kind of client-server interaction is typically known as *fault-driven* [16]. Clients request *specific* data items from the server when they cannot be located in the local caches. If the clients can provide unbounded storage for caching, all data received from the server can be cached. However, with the constraint of limited storage for caching, it is necessary to effectively manage the cache storage and a suitable *caching mechanism* is indispensable.

In the context of DVE across the Internet, the data-shipping architecture can be employed. Since each VE usually contains a huge number of geometric objects with fine details, powerful servers ought to be used to store and maintain the VE database for clients access. Clients should request data from the remote database server on demand via the underlying network infrastructure. Upon receiving the requested data, the clients can then process and render the data into images for display.

Multi-resolution modeling allows the remote database server to transmit an object model at its optimal resolution for rendering at the client machine when the viewer is navigating through a virtual environment. This could save the scarce Internet bandwidth from transmitting details of an object, which would unlikely be visible to the user. To further reduce the dependency on the Internet and the transmission delay, a *caching* and *prefetching* mechanism is needed to retain objects of high affinity and predict those that will most likely be accessed in the near future. In addition, caching at secondary storage has a benefit of persistence and improved data availability at the client. This allows certain degree of disconnected operation on the data that has already been cached in the client's local storage.

4.2 Related Work

A *caching mechanism* is characterized by the *replacement policy* and the *caching granularity* being used. The replacement policy and the caching granularity impact the performance of caching mechanism employed. The ultimate goal is to utilize the limited cache storage for maximizing the performance of the system while minimizing the overhead for data transmission across the network.

4.2.1 Replacement Policy

If a client can provide unbounded storage for caching, it will be possible to cache all incoming objects received from the remote server. However, a more realistic situation is that the available storage for caching is often limited [56]. Thus, a *replacement policy* must be employed to retain objects for possible future reuse. It dictates how victims are selected when the cache storage is exhausted, and is necessary for reclaiming storage space that is required for incoming objects. A replacement policy will usually associate an access score to each of the cached items and choose victims with the lowest values for replacement. The value function is typically based on *temporal locality* and is based on the assumption that items that have been referenced recently are likely to be referenced in the near future.

In [20], various cache replacement policies have been proposed and their suitability to a conventional database system have been examined, including optimal, WORST, Least Recently Used (LRU), CLOCK and Least Reference Density (LRD). A general conclusion on the performance of replacement policies cannot be recommended. In practice, the replacement policy is often approximated by the LRU in conventional caching [6, 22, 73]. LRU can further be generalized into LRU-k which identifies the replacement victim according to the time of k^{th} previous access of a page and LRU is, thus, equivalent to LRU-1. As described in [8], LRU is inappropriate in a context when client's access may change over time. Rather, the semantics of data access is more important in defining the replacement policy.

In the context of DVE, the access patterns exhibited by different users do change over time. Therefore, we have to investigate the suitability of conventional cache replacement policies in this changing access pattern and to develop a more suitable replacement policy for better performance. As compared to the *temporal locality* being assumed in LRU, *spatial locality* would be more appropriate for defining the replacement policy here.

4.2.2 Caching Granularity

In any system that is based on the data-shipping architecture, caching granularity is the minimum unit for cache management and is a key performance concern. Page-based caching granularity is employed by almost all conventional distributed relational DBMS; while *object-based* granularity is another commonly used caching granularity [8].

Page-based Granularity

In a page-based architecture, the unit of transfer between servers and clients is a data page. When a data item is needed at a client, a request will be processed locally down to the level of individual page. If the required page is not present in the local cache, a request of the missing page will be sent to the server. In servicing such a request, the server obtains the requested page from disk when necessary and sends the page back to the client. Upon receiving the result from the server, the requested page is stored in local cache, and an existing page in the cache is replaced if the cache storage is exhausted.

Using page-based granularity, the spatial locality among different items within a page is exploited and a high degree of locality is required among them to be effective [18]. However, in a DVE, different users navigate through the VE with distinct access patterns. Clustering of objects according to the spatial locality exhibited by one user may result in poor locality for another. It is almost impossible to exploit spatial locality among different items and thus, page-based granularity is ineffective in this context. It is therefore inappropriate to transfer objects at the page level in this environment. Rather, it is necessary to consider caching at other granularity.

Object-based Granularity

Object-based, also known as tuple-based, caching granularity is in many ways analogous to page-based granularity. The main difference is that the client cache is maintained in terms of individual objects rather than pages. Caching granularity at this level allows fine tuning of cache contents based on the access locality properties of applications [18]. However, numerous requests of individual object can lead to performance degradation due to the communication overhead in sending large numbers of small messages. Thus, an object-based caching system must be able to group different client requests into a single message and server results into blocks in order to solve this problem.

Most of the existing DVE systems use the method of *level-of-details* (LOD) for representing virtual objects in order to reduce the rendering overhead for distant objects. However, this method has serious drawbacks on caching performance because extra among of data is used to represent a single object. Objects represented by LOD are usually very large in size and occupy multiple data pages. As a result, the page-based granularity is ineffective in cache management using LOD.

4.3 The Cache Model

A cache model indicates how data are acquired at the clients from the remote server. With the establishment of client-side caching, only objects that cannot be located at the local cache have to be requested and transmitted from the server via the network. A cache model that can minimize both the communication and computational costs is necessary. It is possible to have two alternative cache models for use in this environment, the *Push-based* and the *Pull-based* cache model.

4.3.1 The Push-based Cache Model

The *Push-based* cache model is illustrated in Figure 4.1. When a viewer, V, moves within the virtual environment, the client machine, C, transmits the current viewing direction, $\vec{v}_V$, and the current location, loc_V, of V to the server, S. This is a query to the database server for all visible objects. S then evaluates the query and stores the results temporarily. Concurrently, C identifies the visible objects among the cached objects stored in its local storage cache.

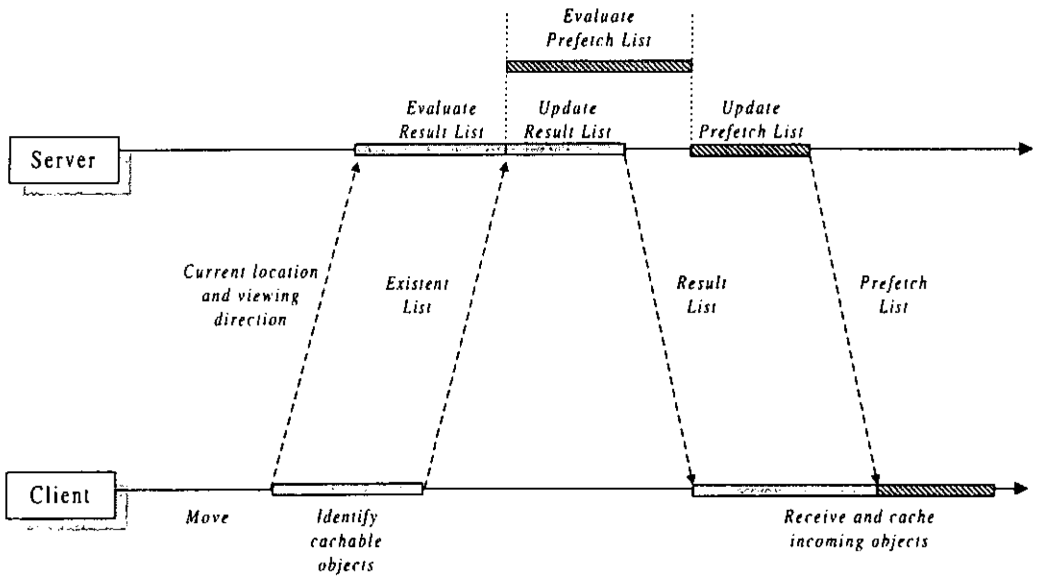


Figure 4.1: The Push-based Cache Model.

Each cached object, o, is associated with a resolution, $\mathcal{R}_o^*$, indicating the current highest possible resolution level of the model available for rendering. This resolution level depends on the number of progressive records, $\mathcal{L}_o^*$, cached in C and is less than or equal to the maximum number of the progressive records, $\hat{\mathcal{L}}_o$, for the object, i.e., $\mathcal{L}_o^* \leq \hat{\mathcal{L}}_o$. C then submits an *existent list* to S, i.e. a list of $\langle o, \mathcal{L}_o^* \rangle$ pairs about those visible objects cached in C's local storage. Visible objects with $\mathcal{L}_o^* \geq \mathcal{L}_o$ do not need to be transmitted as C can render them at the optimal resolution, $\mathcal{R}_o$, from locally cached data.

Upon receiving the existent list from C, S updates the *result list* by subtracting the existent list from it. It is because only those visible objects not being cached in C or those not at the required optimal resolution, $\mathcal{R}_o$, have to be transmitted to C. The result list is in the form of $\langle o$, progressive mesh$\rangle$ pair for transmission. Such a progressive mesh only contains enough progressive records to define the optimal resolution of the object. When prefetching is enabled, S updates the personal profile of V and predicts the next viewing direction, $\vec{v}_{V+1}$, and location, loc_{V+1}, of V. Subsequently, S concurrently evaluates a *prefetch list* based on the prediction while updating the result list. The prefetch list is then updated by subtracting the existent list and the result list and transmitted to C. Both the result list and prefetch list are sorted such that all the base meshes are transmitted prior to the progressive records.

When C receives the result list and the prefetch list from S, C may cache the objects in its local storage. If the storage is exhausted, a replacement policy identifies the victim objects to be discarded. For each object, o, an *access score* indicating the prediction of its future access probability is determined. The higher the access score is, the higher is the probability that o will be accessed again soon. If an incoming object has a score higher than that of some currently cached objects, the storage space of some cached objects is reclaimed and allocated to the new object.

4.3.2 The Pull-based Cache Model

In order to reduce the workload of the server and increase the scalability of the system, we adopt the *pull-based cache model*. In this model, most of the processing is performed at the client machine and the server is mainly used for servicing cache misses. Thus, the server is effectively *stateless* and is more appropriate for supporting larger number of clients. The pull-based cache model is illustrated in Figure 4.2.

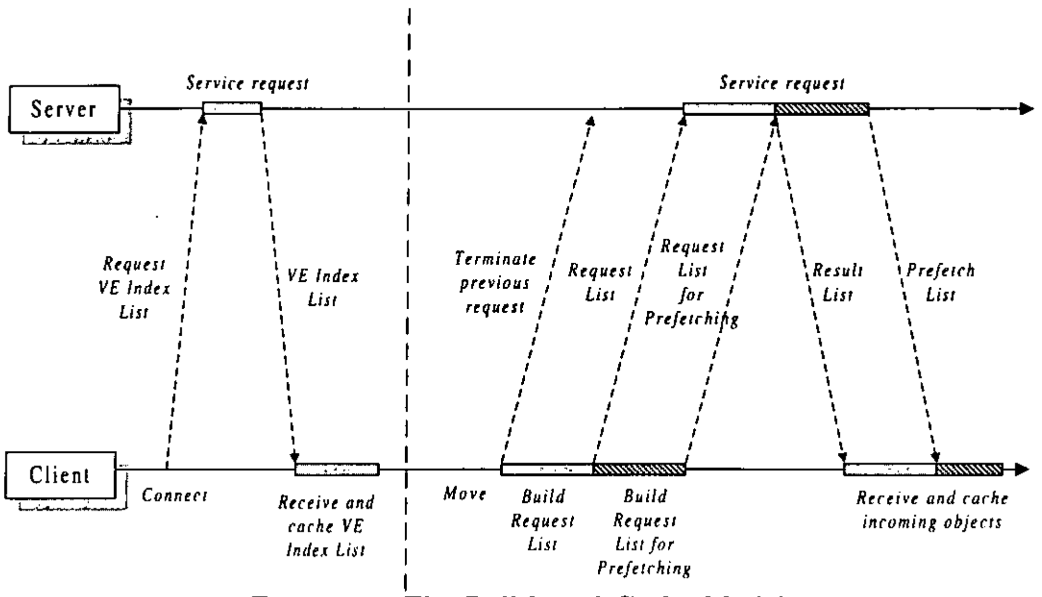

Server
Service request
Service request
Request VE Index List
VE Index List
Terminate previous request
Request List
Request List for Prefetching
Result List
Prefetch List
Client
Connect
Receive and cache VE Index List
Move
Build Request List
Build Request List for Prefetching
Receive and cache incoming objects

Figure 4.2: The Pull-based Cache Model.

When a viewer, V, connects to the server, S, the client machine, C, sends a request to S for the *VE index list.* The VE index list contains the ID, position, object scope and color material for each of the objects within the VE, such that C can request missing objects from S. C caches the VE index list upon receiving it from the server.

When V moves, C sends a message to S indicating the termination of the previous request. This can prevent the transmission of outdated data from S and can save network bandwidth. C then evaluates a list of visible objects, o, within the viewer scope and determines the optimal resolution level, $\mathcal{L}_o$, for each of the visible objects based on the VE index list. Subsequently, C identifies the maximum resolution level, $\mathcal{L}_o^*$, of the cached objects. All the visible objects not cached in C or those not at the required $\mathcal{L}_o$ are requested from S. A request list in the form of $\langle o, \mathcal{L}_o^*, \mathcal{L}_o \rangle$ is then sent to S. After sending the request list, C prefetches objects from S if prefetching is enabled. C first updates the local personal profile of V and predicts the next viewing direction and location of V. A list of objects for prefetching is then identified according to the prediction and a request is sent to S.

4.4 Transmission Order

To retrieve virtual objects for rendering, there can be several ways of arranging the objects for transmission from the database server to the client. We consider two possibilities for transmission order, which are *circular transmission* and *view-dependent circular transmission.*

Circular Transmission

Circular transmission allows objects to be transmitted based on the distance between the object and the viewer. Thus, objects that are closer to the viewer are transmitted first, followed by the distant objects. This ordering is more suitable for a viewer who explores objects around him/her frequently. However, this may increase the response time of the system as the visible objects are multiplexed with the invisible objects during transmission.

View-dependent Circular Transmission

Under the *view-dependent circular transmission*, objects that are within the viewing region of the viewer are identified and transmitted prior to the invisible objects. The visual importance described in Section 3.5 is a natural criteria for the actual transmission ordering. Thus, objects that are closer to the viewer and along the viewing direction are transmitted first. This can reduce the response time of the system.

4.5 Multi-Resolution Caching Mechanism

4.5.1 Replacement Policy

The most straightforward approach is to adopt an existing cache replacement policy for storage caching. The most popular cache replacement policy is LRU, as an approximation to the optimal replacement policy [20]: the least recently accessed entries are replaced. LRU is usually implemented within an operating system and is usually page-based, due to the logical mapping between physical and logical storage. When implementing LRU in our context, it is more reasonable to implement the caching mechanism at the granularity of an object. In LRU, the access score of a cached object is estimated by its last access time. An object, which has not been accessed for a long period of time, has a lower access score, and therefore has a higher probability of being replaced. It has been illustrated that LRU is inappropriate in a context where the objects accessed by a client might change over time [8]. We therefore expect the performance of the LRU-based caching mechanism to be not very promising in this context of DVE across the Internet.

In the context of a virtual environment, we can adopt the semantics of a navigational path which relates the location of o, loc_o, the location of V, loc_V, and the viewing direction of U, $\vec{v}_U$. The farther is an object from the viewer, the lower the resolution it should be at and the longer it takes for the viewer to move to view the object in greater detail. Consequently, its probability of being rendered at a higher resolution and hence its value of being cached in the storage are lower. Similarly, the larger is the angle of an object from the viewing direction, the lower the resolution it needs to be rendered and the longer it takes for a client to rotate to view the object directly in front. Consequently, its probability of being rendered at a higher resolution and its value of being cached in the storage are also lower. The access score of an object could, therefore, be based on this kind of semantics. We call this the *Most Required Movement* (*MRM*):

$$S_{o,V} = \omega\left(1 - \frac{D_o}{D_{o,max}}\right) + (1-\omega)\left(1 - \frac{|\theta_o|}{\pi}\right) \qquad (0 \le \omega \le 1)$$

where D_o is the distance between object o and viewer V while θ_o is the angle between object o and the current viewing direction of viewer V, $\vec{v}_V$, expressed in terms of radians ($-\pi \le \theta \le \pi$). $D_{o,max}$ is defined as the sum of the radii of the viewer scope and the object scope. The divisions by $D_{o,max}$ and π in the formula normalize the contributions by the distance component and the angle component respectively, so that the quantities can be added in a weighted formula. The first term indicates the distance component while the second term indicates the angular component. The weighting factor, ω, adjusts the ratio between the two components in determining the access score, $S_{o,V}$, and it generalizes the formula to $S = a(...) + b(...)$. We use a single weighting factor for simple and fast computation. Notice that the resolution detail of an object is inherently captured by the distance and the angle between the object's location and the viewer's location. The optimal value of the weighting factor, ω, is determined through the simulation study in Chapter 5.

4.5.2 Caching Granularity

As described in Section 3.3, the use of *multi-resolution modeling* allows objects to be represented in different resolution progressively. Thus, it is possible to have a finer caching granularity other than page-based or object-based. We proposed a *multiple-level* caching granularity for cache management in our *multi-resolution caching mechanism.*

When the cache storage is exhausted, it is necessary to reclaim storage space that is needed for the incoming objects. Instead of simply removing individual cached object, the replacement process is performed in multiple levels. The first level is to reduce the resolutions of cached objects to their optimal resolutions according to their access scores. The object with the lowest access score is selected first for replacement. If there is still not enough space to accommodate the incoming objects, the object with the next lowest access score is then selected for replacement and this process will be iterated.

When there is still not enough room to accommodate the new objects even after all cached objects have been reduced to their optimal resolutions, the next level of replacement is performed. The object with the lowest access score is selected again and all its progressive records are removed, leaving only its base mesh. Again, this process will be repeated for other objects. Finally, the base mesh of the object with the lowest access score is removed if there is still not enough space. This process will be iterated again until enough room is allocated for all the incoming objects.

Our multi-resolution caching mechanism tries to retain at least the base meshes of the cached objects in the client's cache storage for possible future reuse. This provides the viewer with a much better visual perception since all or most of the visible objects could be seen instantaneously, even though they may only be visible at a coarse resolution.

4.6 Prefetching Mechanism

We proposed a *user-profile based prefetching mechanism* in which the movement history of individual viewer would be stored in a separate profile for prefetching. As the viewer moves, the moving pattern of the viewer would be captured in his/her user-profile. This would increase the accuracy of the prefetching mechanism and reduce the additional cost of transmitting useless data.

For each viewer, V, the server maintains a separate profile, containing the set of historical locations of V, $\mathcal{P}_V = \{loc_1, loc_2, \ldots, loc_{n-1}\}$. When a viewer moves to a new location, loc_n, and requests the server for renderable objects, the $n+1^{th}$ location and viewing direction of the viewer will be deduced. Based on the deduction, renderable objects at both the current location, loc_n, and the predicted location, loc_{n+1}, are transmitted. This would save future requests to the server if the prefetched objects are indeed required by the client. Precisely, we would like to predict the next location loc_{n+1} of the viewer from its previous locations in the form of a *movement vector*. Each movement vector indicates a previous moving step of the viewer, containing a moving direction and a moving distance.

When a viewer navigates through a virtual environment, he/she commonly has a specific place of interest and navigates towards that location. However, the viewer may explore neighboring locations before reaching the target location. This results in a moving pattern, which shows a trend with some deviations. To predict the next location loc_{n+1} of the viewer more accurately, we can employ some smoothing techniques which are commonly used in forecasting. There are two main types of forecasting methods [54], which are *Averaging Methods* and *Exponential Smoothing Methods*. *Averaging Methods* use equal weightings in previous observations to predict the next outcome while *Exponential Smoothing Methods* use unequal weightings which are decayed exponentially from the most recent observation to the most distant one. *Mean* and *Single Moving Averaging* (also known as Window) are examples of *Averaging Methods*, while *Single Exponential Smoothing* (also known as Exponential Weighted Moving Average, EWMA) is an example of *Exponential Smoothing Methods*. Based on the consideration of computational costs, three schemes (Mean, Window, and EWMA) are selected to study the applicability of these schemes in predicting the next location of the viewer in this context. The semantics of these three schemes are depicted in Figure 4.3.

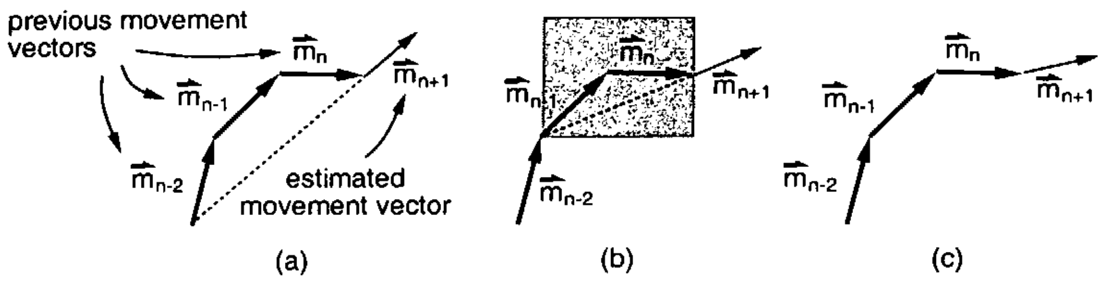


Figure 4.3: Predicting next moving direction: (*a*) Mean, (*b*) Window, and (*c*) EWMA.

In the Mean scheme, the next movement vector, $\vec{m}_{n+1}$, is predicted as the average of all the previous n movement vectors, as depicted in Figure 4.3*a* with three movement vectors. In the Window scheme, each viewer is associated with a window of size W, holding the previous W movement vectors. The next movement vector is predicted as the average of the W most recent vectors. This is indicated in Figure 4.3(*b*), showing a window of size $W = 2$. The Window scheme has a problem of equal weightings of all movement vectors within the window on the prediction.

To better approximate the real-world movement, and to adapt quickly to changes in the viewer's moving pattern, we employ the EWMA scheme to predict the next location of the viewer. It assigns a weight to each of the previous movement vectors so that recent vectors have higher weights and the weights tail off as the vectors become aged. A parameter is the exponentially decreasing weight, α. The most recent vector will receive a weight of 1; the previous vector will receive a weight of α; the next previous one will receive a weight of α^2, and so on. This idea is depicted in Figure 4.3(*c*), indicating the predicted moving direction. The new predicted vector can be computed as follows:

$$\hat{m}_{n+1} = \frac{1}{S_n}\left(\sum_{i=1}^{n} \alpha^{n-i}\vec{m}_i\right) \qquad (0 \le \alpha \le 1)$$

where $S_n = \sum_{i=1}^{n} \alpha^{n-i} = \frac{1-\alpha^n}{1-\alpha}$. As $n \to \infty$, $S_n \to \frac{1}{1-\alpha}$; therefore, $\hat{m}_{n+1}$ can be approximated by $(1-\alpha)\sum_{i=1}^{n} \alpha^{n-i}\vec{m}_i$. The new $n+1^{th}$ movement vector is estimated as: $\hat{m}_{n+1} = \alpha\hat{m}_n + (1-\alpha)\vec{m}_n$.

EWMA has been shown to be quite effective in predicting access probabilities of data items in database applications by adapting rather quickly to changes of access patterns [72]. However, it might not perform as satisfactory in this new context of predicting the next viewer location. This is because the access probability to a data item is bounded between 0 and 1. EWMA is trying to incorporate the effect of the change into the new estimate and the estimation error would normally not diverge. In this new context here, we are using EWMA to predict a vector, whose direction is an angle with an unbounded scope, i.e., the angle can increase indefinitely, for example, through continuous rotation in a circle. Thus, EWMA may not be able to cope with the "non-stationary" changes. We need to explicitly correct the prediction with adjustment from residuals or error predictions.

Let us denote the n^{th} movement vector be $\vec{m}_n$ and the predicted $n+1^{th}$ movement vector be $\hat{m}_{n+1}$. The residual in each prediction is $\vec{e}_n = \hat{m}_n - \vec{m}_n$. We consider the angle between $\hat{m}_n$ and $\vec{m}_n$, denoted as $\phi_n = arg(\hat{m}_n) - arg(\vec{m}_n)$, where $arg(\vec{m})$ is the argument of the vector $\vec{m}$ in a complex plane. $\vec{m}_n$ can be predicted by rotating $\hat{m}_n$ through an angle of $-\phi_n$, i.e., a multiplication by $e^{-\imath\phi_n}$. Since we do not really know ϕ_{n+1} when we predict $\hat{m}_{n+1}$, we must try to predict ϕ_{n+1} as well. There can be different ways of predicting ϕ_{n+1} from the previous values of ϕ_i, namely, Mean, Window and EWMA. Again, we propose to use EWMA to compute the prediction of ϕ_i at each step as we compute $\hat{e}_i$. Thus, $\hat{\phi}_{n+1} = \alpha\hat{\phi}_n + (1-\alpha)\phi_n$, and $\vec{m}_{n+1} = \hat{m}_{n+1}e^{-\imath\hat{\phi}_{n+1}}$. We call this the EWMA-R (EWMA with residual adjustment) scheme, since it involves the adjustment to the direction of each movement.

In order to determine the optimal value of the exponentially decreasing weight, α, in both EWMA and EWMA-R schemes, a set of simulated experiments has been conducted. The details of the simulated experiments are shown in Chapter 5.

Chapter 5

Simulation Study

5.1 Introduction

We have developed a simulation model and have conducted some extensive simulated experiments to quantify and evaluate the performance of our proposed *multi-resolution caching mechanism* with *MRM* replacement policy and *user-profiling based prefetching mechanism*. We have chosen the representative experiments to be presented here. We have already conducted some preliminary simulated experiments to study the impact of caching granularity on the performance of caching mechanism. We have compared our proposed *multi-resolution caching granularity* with *LOD*, which is commonly used in most existing DVE systems. We have already concluded that our proposed *multi-resolution caching granularity* out-performs *LOD* under most cases [11].

The simulation study is conducted around three main objectives. First, we would like to compare our *MRM* replacement policy with conventional LRU replacement policy. Second, we would like to examine the effectiveness of various prefetching schemes. Third, we would like to determine the optimal value of the weighting factor,

ω, in the *MRM* replacement policy and the exponentially decreasing weight, α in the *EWMA* and *EWMA-R* prefetching schemes.

5.2 Simulation Model

We have developed a simulation model which allows us to experiment the behavior of our proposed caching and prefetching mechanisms under diverse situations easily. The simulation model is implemented using CSIM [68] and STL [57]. The architecture of the simulation model is depicted in Figure 5.1.

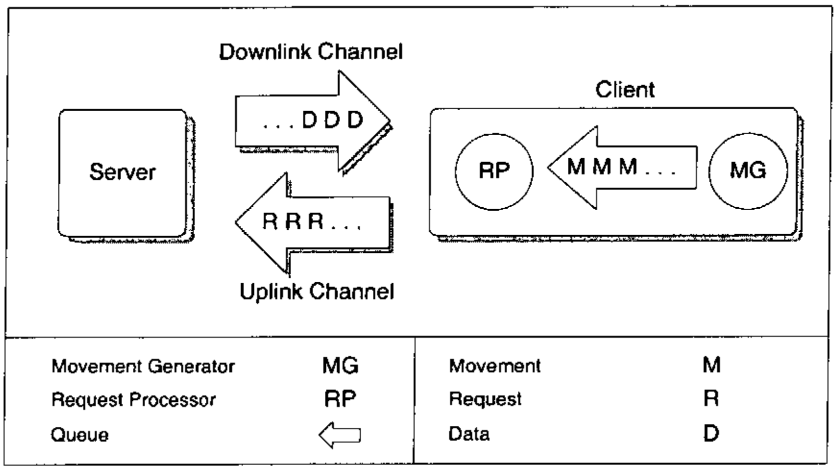


Figure 5.1: Simulation model.

In our simulation model, the client module contains a *Movement Generator*, *MG*, which continuously generates different movements, *M*, simulating user inputs. The movements are then queued and passed to the *Request Processor*, *RP*. For each movement, *RP* looks up the client's cache storage and formulates a request, *R* for all missing data at the viewer's current location. Subsequently, the request is sent to the server through the *Uplink Channel*. Upon receiving requests from the client, the server retrieves data from its local storage and sends the requested data, *D*, back to the client through the *Downlink Channel*. When the client receives data from the server, the data is cached and replacement is performed when the cache storage is exhausted.

5.3 Experimental Environment

Notation	Description
N	Number of virtual objects
n	Size of storage cache (percentage of database)
R_{disk}	Replacement Policy for storage cache
ω	Adjustment parameter for MRM
F_{disk}	Prefetching scheme for storage cache
W	Window size in prediction
α	Exponentially decreasing weight
P	Moving patterns of the viewer
T	Transmission pattern of renderable objects

Table 5.1: Parameters listing for simulated experiments.

The experimental setting of the simulation model is outlined in Table 5.1. In the experiments presented below, we focus on an environment with one database server and a single client. In our experiments, the client and the database server communicate via a network with a bandwidth of 1Mbps, modeling the Internet environment. The virtual world is constrained in a dimension of 10000×10000 units. In our model, there are N virtual objects stored at the database server. The N virtual objects are uniformly distributed over the entire virtual world. Each square unit of the virtual world, therefore, contains an average of $\frac{n}{10^8}$ objects. The viewer is assumed to be located at the center of the viewer scope that is modeled as a circle. The viewing angle, that determines the viewing region of the viewer scope, is set to 120 degrees, i.e., $\frac{2\pi}{3}$ radians. The radius of the viewer scope is fixed at 500 units. The radius of the object scope for each object ranges uniformly from 75 units to 125 units with a mean of 100 units.

The virtual objects in the database server are represented by multi-resolution model. Each object contains a base mesh and a list of progressive records. The base mesh has a size of 31KB. It contains 695 vertices and 1684 triangles. There are totally 6944 progressive records associated with each object. Meanwhile, the size of the progressive records is 329KB and the size of individual progressive record is approximately 49 bytes. The size of an entire object is, therefore, 360KB. The resolution required for each virtual object is based on the *optimal resolution* as described in Section 3.5 and K_o used in defining the optimal resolution is 1.5317.

In our study, only storage cache at the client is modeled. The bandwidth of disk is set to 40Mbps modeling a fast SCSI disk. The size of the client cache is equal to $n\%$ of that of the database. We experiment two replacement policies, R_{disk}, LRU and MRM. We denote the MRM replacement policy with adjustment parameter, ω, of 0.1, 0.5 and 0.9 as MRM-0.1, MRM-0.5 and MRM-0.9 respectively. We also experiment three different prefetching schemes, F_{disk}, namely EWMA, Mean and Window. For the EWMA scheme, we experiment both with and without residual adjustment enabled. For notational convenience, we refer to the EWMA scheme with residual adjustment enabled as EWMA-R, and the one with residual adjustment disabled as EWMA-NR. The exponentially decreasing weight for determining access score in EWMA is denoted as α. We further denote a window scheme with window size, W, as Win-W. For instance, a window scheme with W=1 will be denoted as Win-1. In addition, we model two transmission patterns, T, in transmitting the requested objects to the client. The two transmission patterns are circular transmission (Cx) and view-dependent circular transmission (Vx) as described in Section 4.4.

Moving Pattern

We study four moving patterns, P, experienced by the viewer. The moving patterns are depicted in Figure 5.2. Each pattern is composed of a sequence of movement steps. Each movement step changes the location of the viewer, that is a translational movement, and/or the viewing direction of the viewer, that is a rotational movement. Every step in a translational movement makes a net progress of 10% of the radius of the viewer scope, i.e., 50 units.

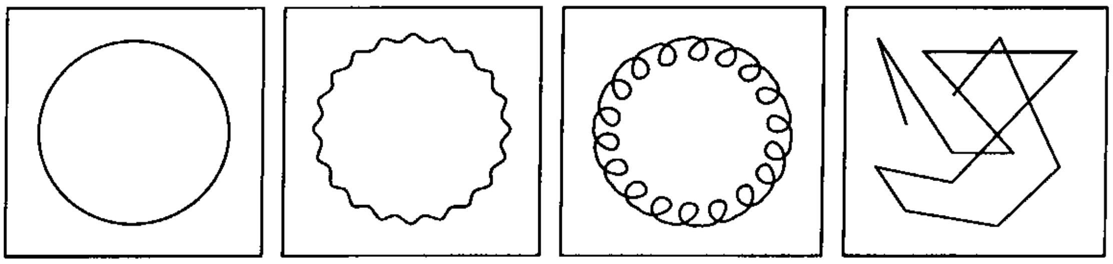


Figure 5.2: Moving patterns: (*a*) CP, (*b*) CT, (*c*) CR, and (*d*) RW.

The first moving pattern models a constant circular translation pattern (CP), as depicted in Figure 5.2(*a*). The viewer moves along a circular path with a diameter of 8000 units. The start and the end location are the same. Each movement step includes a translation of 50 units along the viewing direction and a rotation of 0.716 degrees, i.e., 0.012 radians. The total number of movement steps is 503 in one complete movement. The second pattern (Figure 5.2(*b*)) models the same pattern as the circular pattern except that after each movement step is made, the viewer rotates from 60 degrees ($\frac{\pi}{3}$ radians) to -60 degrees ($-\frac{\pi}{3}$ radians). We call this circular turn pattern (CT). This models a situation in which the viewer explores nearby objects for every movement step. The third pattern (Figure 5.2(*c*)) models a similar pattern as the circular pattern except that after each movement step is made, the viewer performs six rotations, each of 60 degrees ($\frac{\pi}{3}$ radians) Thus, after the rotations are complete, the viewer turns back to the original direction, that means, it rotates in 360 degrees (2π radians). The viewer then moves along the viewing direction. This models a situation in which the viewer examines all virtual objects around him/her for every movement step. We call this the circular rotation pattern (CR).

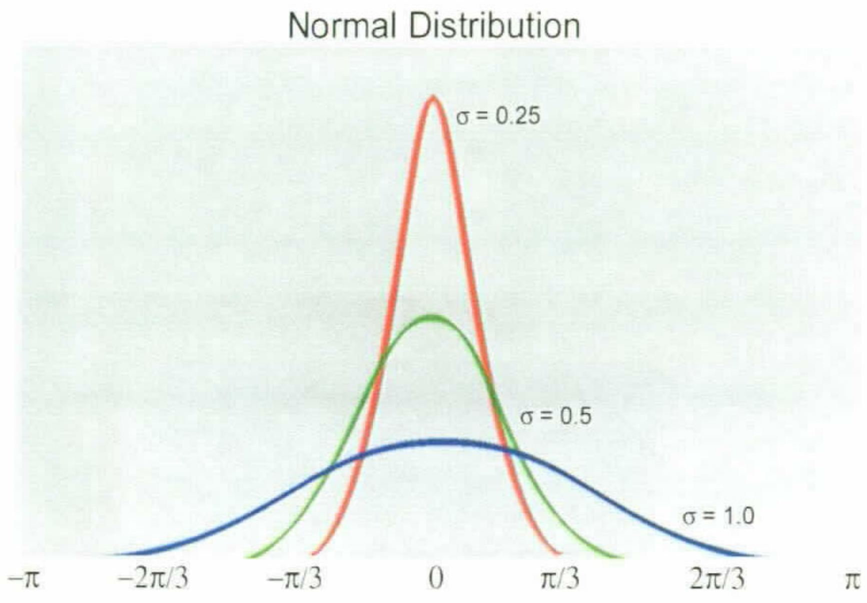


Figure 5.3: Normal distribution with σ 0.25, 0.5, and 1.0.

Other than those circular moving patterns, we study the random moving pattern (random walk or RW) as depicted in Figure 5.2(d). The viewer, starting at the center of the virtual environment, i.e., $\langle 5000, 5000 \rangle$, moves with a translation of 50 units along the viewing direction. After each translational movement, the viewer rotates randomly with a minimum angle of -180 degrees ($-\pi$ radians) to a maximum angle of 180 degrees (π radians). We model three random moving patterns with different probability distribution functions of rotational angle (5.3). It is a normal distribution function with σ equal to 0.25, 0.5 and 1.0 respectively. The smaller the σ value, the smaller the probability viewer rotates. In our study of random moving patterns, the total number of movement steps is 1000. The simulated random patterns are shown in Figure 5.4. For notational convenience, we refer to the random patterns with σ equals to 0.25, 0.5 and 1.0 as RW1, RW2 and RW3 respectively.

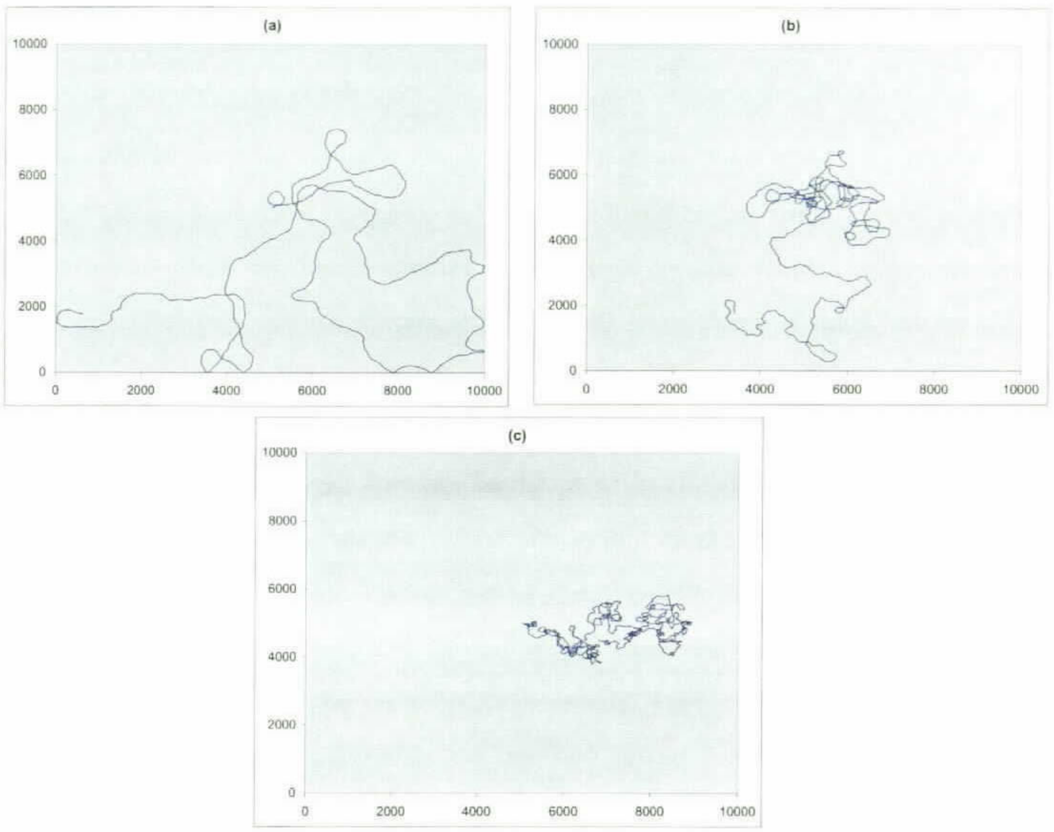


Figure 5.4: Random walk patterns: (a) RW1 ($\sigma = 0.25$), (2) RW2 ($\sigma = 0.5$), and (c) RW3 ($\sigma = 1.0$).

5.4 Performance Evaluation

We characterize the performance of the caching and replacement schemes with two sets of performance metrics. The first set quantifies the absolute performance by the *cache hit ratio* and the *response time*. Cache hit ratio measures the portion of the visible objects, i.e., those within the viewing region, that could be retrieved from the local cache of the client. Response time measures the average time (in seconds) required to retrieve all progressive records from the server for representing the visible objects in their optimal resolution.

The second set of metrics is concerned with the image quality perceived by a viewer, which is quantified by the *visual perception* and the *latency*. Visual perception measures how good the image quality experienced by the viewer. For each visible object, o, cached in the local storage, its visual perception is modeled as a cubic function: 1-$(\frac{B_o - B_o^*}{B_o})^3$, where B_o is the expected size of object o at its optimal resolution and B_o^* is the size of the object currently cached. A visual perception of 100% is assumed when the the cached model could provide the optimal resolution while a visual perception of 0% is assumed when the model cannot be located in the cache. We use a logarithmic-like function to model visual perception because when a viewer makes a move in the virtual world, he/she would experience a high visual perception if all visible objects could be seen instantaneously, even at a coarse resolution. By contrast, the viewer would experience a low perception if he/she needs to wait for a long time before all visible objects could be observed. Latency measures the average time (in seconds) required to retrieve the base meshes of all visible objects, i.e., from the moment the viewer makes a move, to the moment when there is a coarse image of all visible objects. For each experiment conducted, the average of each metric is determined from all movement steps.

5.5 Experiments on Caching Mechanism

In this section, we are going to discuss the performance of the proposed caching mechanism as compared to conventional ones. Here, we select representative sets of experiment to illustrate the performance of our proposed caching mechanism. The parameter setting of the experiments presented is summarized in Table 5.2.

Parameter	Values for each experiment					
Experiment	#1	#2	#3	#4	#5	#6
N	5000				2500, 5000, 10000, 20000,	5000
n	0%, 0.014%, 0.028%, 0.043%, 0.057%, 0.071%, 0.085%, 0.114%, 0.142% 0.170%, 0.227%, 0.284%,		0.057%	0%, 0.014%, 0.028%, 0.043%, 0.057%, 0.071%, 0.085%, 0.114%, 0.142% 0.170%, 0.227%, 0.284%,		
R_{disk}	LRU, MRM				MRM	
ω	0.1		0.1, 0.5, 0.9		0.1	
F_{disk}	No Prefetch					
W	N/A					
α	N/A					
P	CP	RW3	CP, CT, CR, RW1, RW2, RW3	CP		CP, CT, CR, RW1, RW2, RW3
T	Cx, Vx		Vx			

Table 5.2: Parameter values for simulation experiments #1 to #6.

5.5.1 Experiment #1

In the first experiment, we would like to study the effect of two transmission patterns on caching mechanism. In this experiment, there are 5000 objects in the database, i.e., the database is approximately 1.72GB. The size of the storage cache varies from 0 to 5MB, i.e, 0.0% to 0.284% of N. The moving pattern is CP. We experiment LRU and MRM replacement policies, each is evaluated based on two transmission patterns (Cx and Vx). The adjustment parameter, ω, of MRM is fixed at 0.1. The measurements of the four metrics which are organized as an array of graphs are depicted in Figure 5.5. The top row (Figures 5.5(a) and 5.5(b)) depicts the hit ratio and response time while the bottom row (Figures 5.5(c) and 5.5(d)) depicts the visual perception and latency measurements.

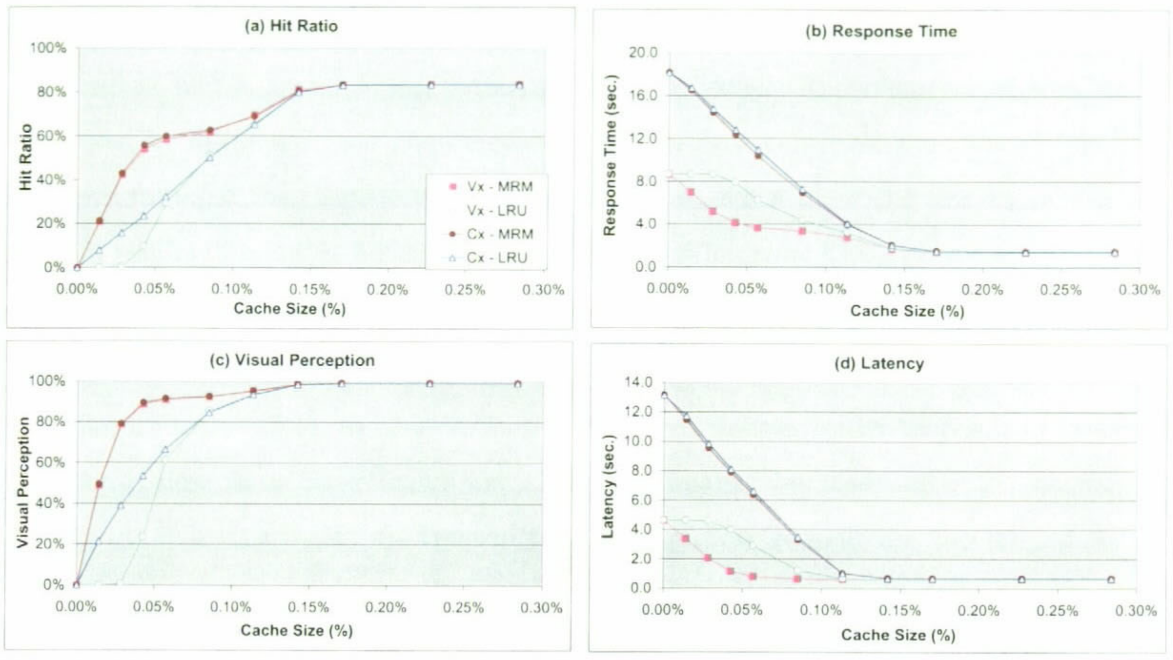


Figure 5.5: Performance measurements of Experiment #1.

From Figure 5.5(a), we observe that when the cache size is small (below 0.085%), LRU results in a smaller hit ratio under Vx than the Cx counterpart. When the cache size is too small (below 0.028%), LRU cannot hold all the visible objects and it results in a hit ratio of almost 0%. Under Vx, objects within the viewing region are transmitted to the client earlier than those beyond the viewing region. LRU would pick up those objects within the viewing region as replacement victims, yielding a seemingly anomalous result. These objects, however, are likely to be accessed in the near future as the viewer moves along the viewing direction. This results in a lower hit ratio. This also explains why LRU results in a lower visual perception under Vx than those under Cx when the cache size is small, as depicted in Figure 5.5(c). When the cache size enlarges, more space is available to retain all objects within the viewing region and thus, the hit ratio increases.

We also notice that MRM results in a similar hit ratio and visual perception regardless of the transmission patterns. This is because the replacement victims selected by MRM are most likely beyond the viewing region, and thus not affecting the hit ratio nor the visual perception. Furthermore, when the cache size diminishes, MRM results in a higher hit ratio and visual perception than LRU. This is due to the same reason that LRU tends to replace objects within the viewing region, but MRM would avoid their replacement.

From Figure 5.5(*b*), we observe that replacement policies under Vx result in lower response time than those under Cx. This is primarily because under Vx, objects within the viewing region are transmitted prior to those beyond the viewing region. However, under Cx, objects within and beyond the viewing region are multiplexed during transmission, resulting in a higher response time. This also explains why the latency under Vx is much lower than that under Cx, as depicted in Figure 5.5(*d*). We also observe that MRM results in lower response than that of LRU under the two transmission patterns, especially when the cache size is small. This is due to the higher hit ratio from MRM.

From this experiment, we can conclude that the Vx transmission pattern is very effective in reducing the response time and latency of the DVE system.

5.5.2 Experiment #2

In the second experiment, we would like to further study the effect of the two transmission patterns on the performance of the caching mechanism. In this experiment, the moving pattern is fixed at RW3 and all the other parameters are the same as those in Experiment #1. The results are depicted in Figure 5.6.

From Figure 5.6(*a*) and Figure 5.6(*c*), we observe that when the cache size is small (below 0.085%), LRU results in a smaller hit ratio and visual perception under Vx than the Cx counterpart. When the cache size is too small (below 0.028%), LRU cannot

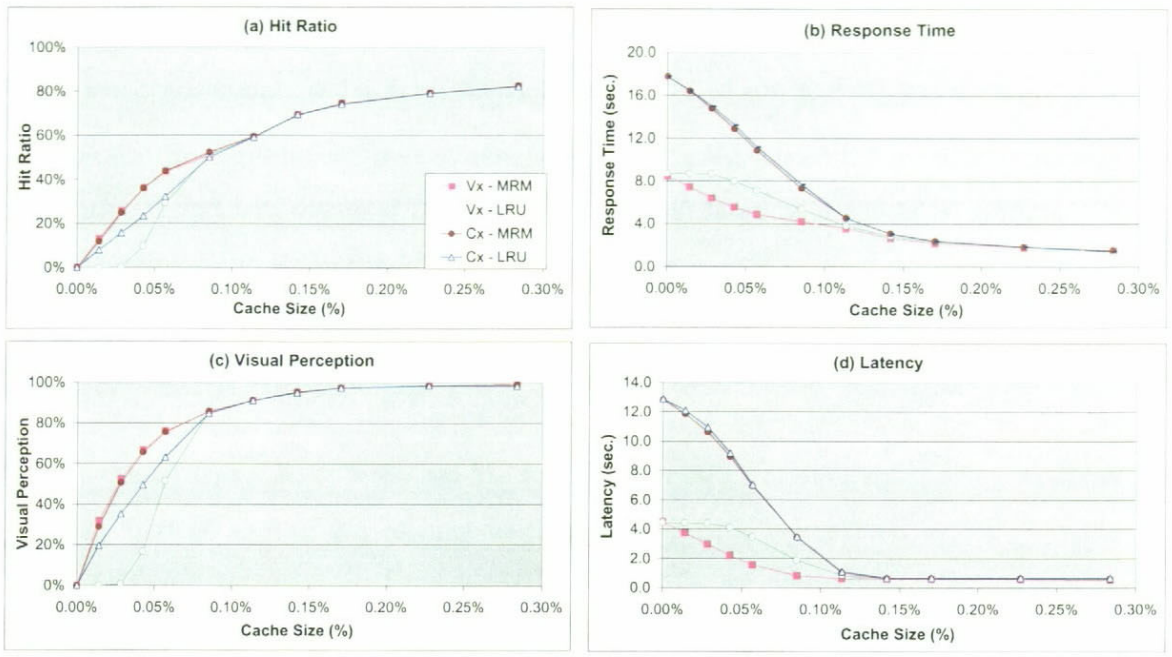


Figure 5.6: Performance measurements of Experiment #2.

hold all the visible objects and it results in a hit ratio of almost 0%. As described in Experiment #1, LRU tends to replace objects within the viewing region prior than those beyond the viewing region due to the transmission order in Vx. This results in a lower hit ratio and visual perception under Vx than those under Cx. We also notice that MRM results in a similar hit ratio and visual perception regardless of the transmission patterns. It results in a higher hit ratio and visual perception than LRU because MRM is capable of retaining objects within the viewing region regardless of the transmission patterns.

From Figure 5.6(*b*) and Figure 5.6(*d*), we observe that replacement policies under Vx result in lower response time and latency than those under Cx. This is because under Cx, objects within and beyond the viewing region are multiplexed during transmission. In addition, we observe that the results in this experiment exhibit identical trends as the previous one. This can show that performance of MRM and LRU under the two transmission order is relatively independent of the moving patterns.

From this experiment, we can conclude that the Vx transmission pattern is more effective than Cx in reducing the response time and latency of the DVE system. We therefore focus on Vx in the following experiments.

5.5.3 Experiment #3

In the third experiment, we would like to study the effect of different moving patterns on the performance of the caching mechanism. With n being fixed at 5000 objects (1.72GB) and the cache size fixed at 1MB, i.e. 0.057%, we experiment different moving patterns and the two replacement policies, LRU and MRM. The setting of adjustment parameter, ω, for MRM includes 0.1, 0.5 and 0.9. We only present the metrics for different policies under the Vx transmission pattern that is shown to perform better in the previous experiments. The results are depicted in Figure 5.7.

We observe from Figure 5.7(a) that LRU results in a smaller hit ratio under all moving patterns than MRM. When the cache size is small (1MB), the cache cannot hold all the visible objects and thus, replacement is needed to reclaim storage space for incoming objects. As described in Experiment #1 and #2 , LRU would discard objects within the viewing region due to the access order under Vx. These objects may in fact have a high probability of being accessed soon as the viewer is moving towards the viewing direction. By contrast, MRM always picks up those objects beyond the viewing region for replacement. As a result, LRU results in a lower hit ratio and visual perception than MRM. It, therefore, causes an increase in both the response time and latency.

We also observe that MRM results in a higher hit ratio with a smaller adjustment parameter, ω. With ω equal to 0.5 and 0.1, the hit ratio can be improved up to 72.7% and 97.6% respectively. Actually, the adjustment parameter, ω, controls the weighting

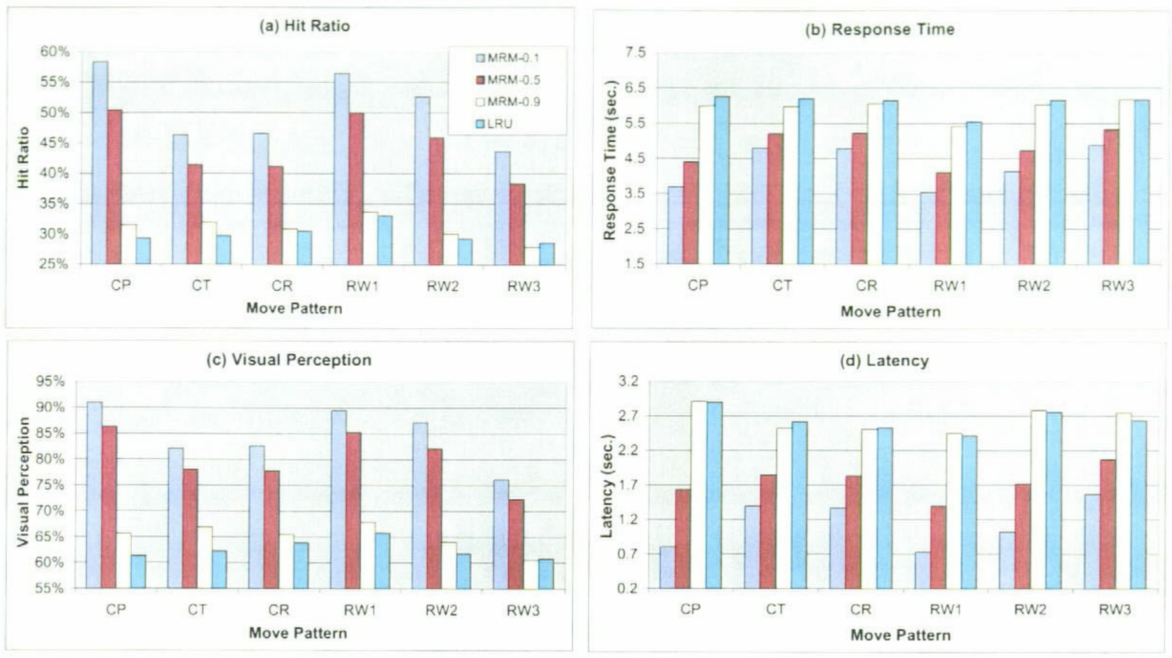


Figure 5.7: Performance measurements of Experiment #3.

between the distance component and the angular component in determination of access score in MRM. The higher the ω value, the more weighting of the distance component and the less weighting of the angular component. Thus, objects that are closer to the viewer will mostly be kept. By contrast, the lower the ω value, the more weighting of the angular component and the less weighting of the distance component. Thus, objects that are closer to the viewing direction of the viewer will mostly be kept. As these objects will have a higher probability to be accessed in the future, a lower ω value results in a higher hit ratio.

We further notice that the hit ratio for the moving pattern CP is higher than that for CT and CR under MRM-0.1 and MRM-0.5. It is because under CP, the viewer only moves along the viewing direction. Thus, objects along the viewing direction that are being kept under MRM-0.1 and MRM-0.5 will most likely be accessed again. However, under the moving patterns of CT and CR, the viewer explores nearby objects by rotating around before making the next movement and thus, objects being kept under MRM with a low ω may not be accessed in the future under these two moving patterns. This results in a better performance for MRM-0.1 and MRM-0.5 under CP than those under CT and CR. Similarly, under the moving patterns of RW, the probability of changing the viewing direction is increased with the σ in defining the moving pattern. The probability of turning is increased from RW1, RW2, to RW3. This accounts for the reduction of cache hit ratio and visual perception from RW1 to RW3.

From this experiment, we can conclude that MRM is more effective than LRU when the cache size is small. MRM results in a higher cache hit ratio and visual perception and in turns, reduces the response time and latency of the DVE system. In addition, MRM with a smaller weighting factor, ω, performs better.

5.5.4 Experiment #4

In this experiment, we would like to study the effect of cache size on the performance of various replacement policies. With N being fixed at 5000 objects and moving pattern being fixed at CP, we experiment the cache size ranging from 0MB to 5MB, i.e. 0.0% to 0.284%. The replacement policies are LRU and MRM and the adjustment parameter, ω, for MRM is fixed at 0.1, 0.5 and 0.9. Similar to the previous experiments, we only show the metrics for different policies under Vx. The results of this experiment are depicted in Figure 5.8.

From Figure 5.8(a), we observe that the cache hit ratio for all four replacement policies increases from 0% up to 83.4% as the cache size increases. Thus, the response time and latency reduces. The cache hit ratio achieves a maximum value when the cache becomes saturated. Further cache enlargement does not help improving the performance. It is because when the cache is large enough to hold all objects within

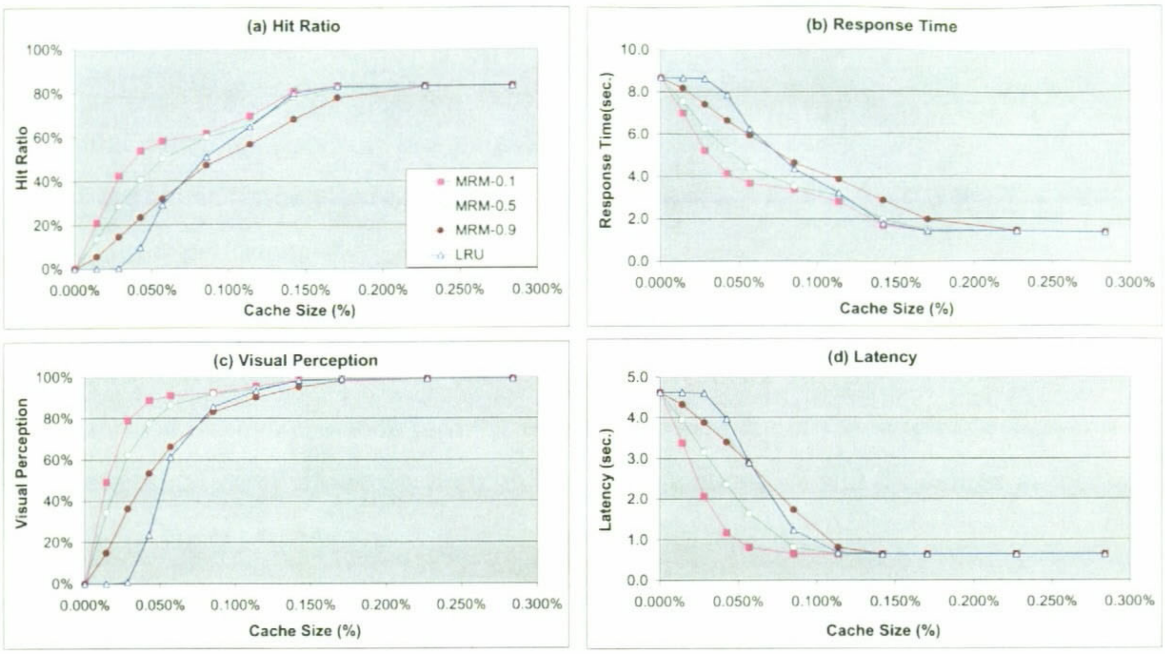


Figure 5.8: Performance measurements of Experiment #4.

the viewing region, no replacement of these objects is required.

We also observe that the cache hit ratio of LRU remains 0% for cache size from 0 to 0.5MB (0.028%) and the hit ratio increases from 0% to 83.4% for cache size from 0.5MB (0.028%) to 3MB (0.170%). It shows that LRU is ineffective when the cache size is small. It is because LRU would discard objects within the viewing region due to the access order. In case a very small cache is used, LRU replaces almost all visible objects and the cache becomes useless. This accounts for the 0% of hit ratio and visual perception.

We notice that MRM-0.1 and MRM-0.5 achieve a higher hit ratio than LRU for all cache sizes. When cache size is set to 0.75MB (0.043%), the hit ratio of MRM-0.1 and MRM-0.5 can be improved by 44.7% and 31.6% respectively. It is interesting that MRM-0.9 achieves a higher hit ratio than LRU for a cache size of up to 1.5MB (0.085%), but it achieves a lower hit ratio for cache size from 1.5MB up to 4MB

(0.227%). As mentioned before, the lower the ω value, the more weighting of the angular component and the less weighting of the distance component. Thus, objects that are closer to the viewing direction of the viewer are mostly kept and this results in a better performance.

This experiment further shows that LRU is not as effective as MRM in this context and MRM with a small ω value performs the best. From the results in Experiment #3 and #4, we can conclude that 0.1 is the optimal value of the weighting factor, ω, in determination of the access score in *MRM*. Therefore, we will fix *omega* as 0.1 in the following experiments.

5.5.5 Experiment #5

In the fifth experiment, we would like to examine the effect of database size on the performance of caching mechanism. In this experiment, N ranges from 2500 objects, 5000 objects, 10000 objects, to 20000 objects, leading to a database size of 0.68GB, 1.72GB, 3.44GB and 6.88GB respectively. In addition, the moving pattern, P, is CP and the transmission pattern is Vx. The replacement policy is MRM-0.1 and the cache size ranges from 0% to 0.284%. The results of this experiment are depicted in Figure 5.9.

From both Figure 5.9(a) and Figure 5.9(c), we observe that the hit ratio and visual perception are almost invariant when accessing database with different sizes. Among various database sizes, the hit ratio increases from 0% to a maximum of 83% and the visual perception increases from 0% to 99% when the cache size increases.

We notice from Figure 5.9(c) and Figure 5.9(d) that the response time and latency increase as the database size increases. When the cache size is large enough, the caching mechanism can successfully reduce the response time to one fifth and reduce the latency to one sixth, as compared to the base case of no caching. We found that

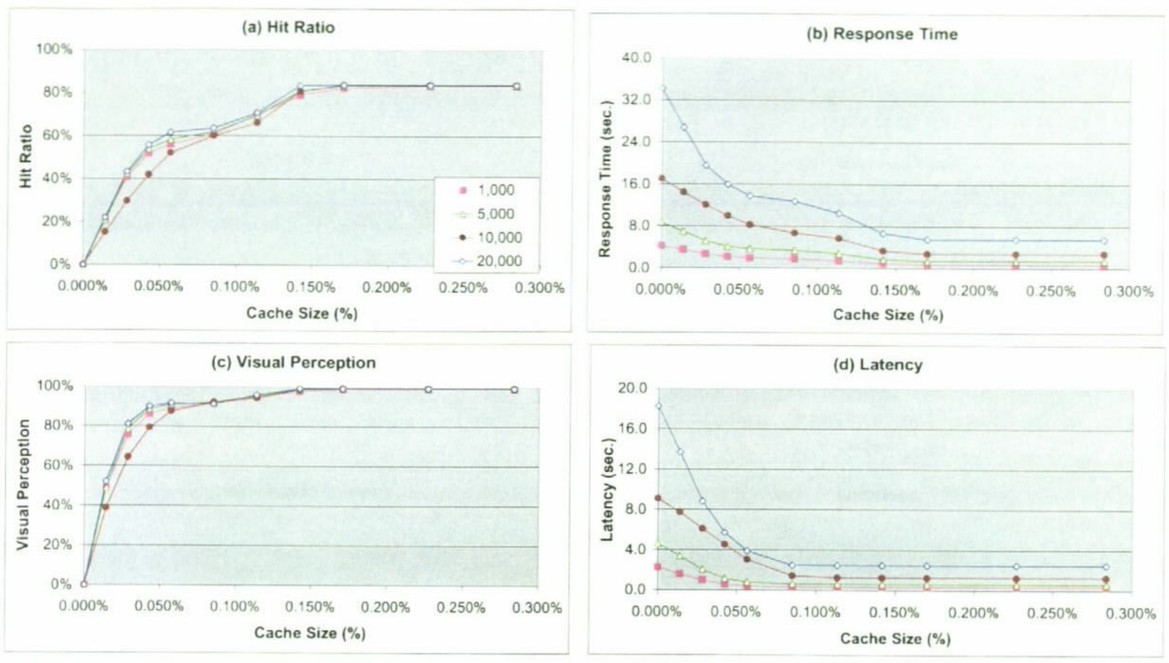


Figure 5.9: Performance measurements of Experiment #5.

the response time and latency are directly proportional to the database size. As the database size increases, the density of objects contained in each unit of the database also increases. Thus, more objects have to be transmitted to the client, resulting in an increase in both the response time and latency.

This experiment shows that MRM, like other caching mechanisms [44], is relatively invariant to the database size and it is effective in reducing the response time and latency of the DVE system.

5.5.6 Experiment #6

In this experiment, we would like to study the effect of cache size on the performance of caching mechanism under different moving patterns. With N being fixed at 5000 objects, we experiment cache size ranging from 0 to 5MB, i.e. 0.0% to 0.284%. The replacement policy is MRM-0.1 and the transmission pattern is Vx. The results of this experiment are depicted in Figure 5.10.

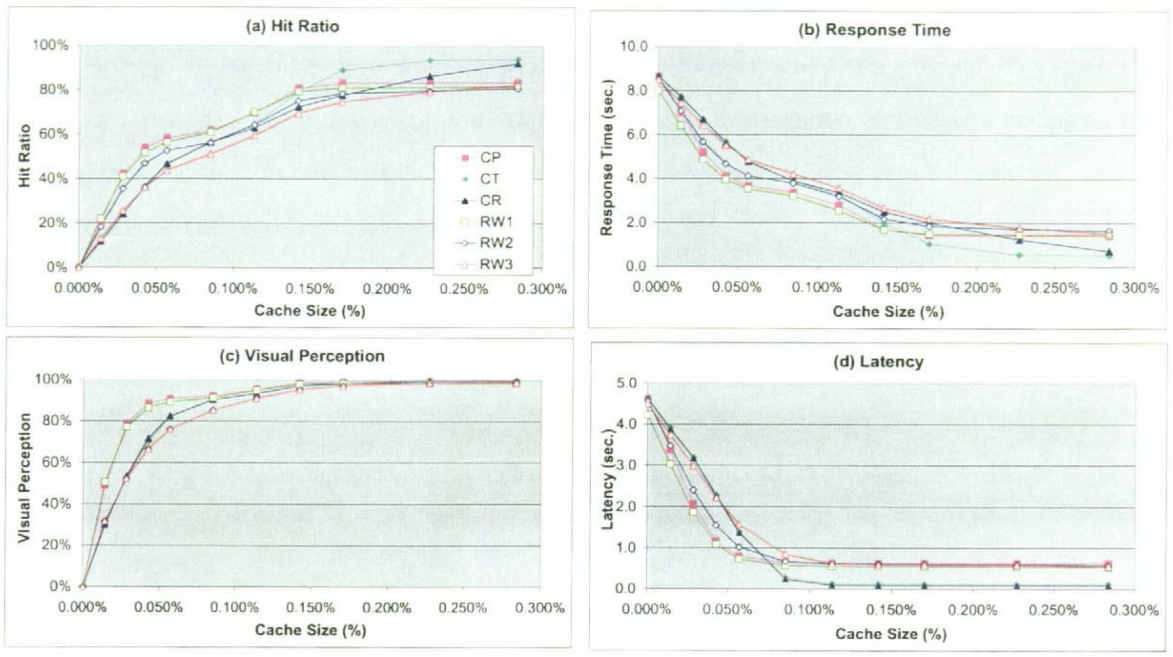


Figure 5.10: Performance measurements of Experiment #6.

As shown in Figure 5.10, we observe that all the four metrics exhibit a similar trend as the cache size increases and they are almost invariant to the different moving patterns. As the cache size increases, more objects are kept in the local storage, thus resulting in a rise in cache hit ratio and visual perception, as well as a drop in response time and latency.

From Figure 5.10(a) and Figure 5.10(c), we observe that the moving pattern RW3 achieves the lowest hit ratio and visual perception. It is because under RW3, a viewer has a high probability of changing its viewing direction. Since MRM retains objects within the viewing region, it is not very effective in improving the performance under this moving pattern. This also accounts for the highest response time (as shown in Figure 5.10(b)) of RW3 among the six moving patterns. As shown in Figure 5.10(d), we notice that the moving patterns CT and CR achieve the lowest latency of 0.127 second and 0.091 second respectively when the cache size is greater than 2.5MB (0.142%). Under these two moving patterns, the viewer will turn around to explore objects around before making the next translational movement. When the cache size is large enough, the base mesh of objects within the viewer scope can be retained and thus, no re-transmission of the base mesh is needed.

This experiment shows that MRM is relatively invariant to different moving patterns and is effective in reducing the response time and latency of the DVE system.

5.6 Experiments on Prefetching Scheme

Parameter	Values for each experiment				
Experiment	#7	#8	#9	#10	#11
N	5000				2500, 5000, 10000, 20000
n	0.284%		0.114%, 0.142%, 0.170%, 0.199%, 0.227%, 0.255%, 0.284%, 0.312%, 0.341% 0.397%, 0.454%, 0.568%		0.284%
R_{disk}	MRM				
ω	0.1				
F_{disk}	EWMA-NR, EWMA-R	No Prefetch, Mean, Window, EWMA-NR, EWMA-R			
W	N/A	1, 3, 5, 7			
α	0.1, 0.5, 0.9	0.5			
P	CP, CT, CR, RW1, RW2, RW3		CP	RW3	CP
T	Vx				

Table 5.3: Parameter values for simulation experiments #7 to #11.

After studying the performance of the proposed *multi-resolution caching mechanism*, we would like to examine performance of the various prefetching scheme as described in Section 4.6. We have selected a representative set of experiments and presented in the following subsections. The parameter setting of the following four experiments is summarized in Table 5.3.

5.6.1 Experiment #7

In the first experiment on prefetching, we would like to determine the optimal value of the exponentially decreasing weight, α, for use in the EWMA prefetching schemes. In this experiment, N is fixed at 5000 object, resulting in a database size of approximately 1.72GB. The exponentially decreasing weight, α, includes 0.1, 0.5, and 0.9. The cache size is fixed at 5MB (0.284%). We experiment all the six moving patterns: CP, CT, CR, RW1, RW2 and RW3. The measurements of the four metrics are depicted in Figure 5.11.

From Figure 5.11(a) and Figure 5.11(c), we observe that EWMA-R results in slightly higher hit ratio and visual perception than EWMA-NR with the same exponentially decreasing weight, α, under most of the moving patterns. Also, EWMA schemes with α 0.1 and 0.5 produces similar cache hit ratio and visual perception which are generally higher than than those with α 0.9. With a higher *alpha* value, there is a higher weighting of the aged moving vectors in predicting the next moving vector and this introduces more noise in the prediction. The performance is thus not as good as those with a smaller α under most moving patterns. However, we notice that EWMA-R schemes with α 0.1 and 0.9 result in a lower hit ratio and visual perception than that with α 0.5 under RW3. It is because the viewer has a very high degree of randomness in changing his/her viewing direction under RW3. Too little or too much weighting of the aged movement vectors is not effective in predicting the error introduced by the non-stationary changes in viewing direction.

From this experiment, we can conclude that the optimal value of the exponen-

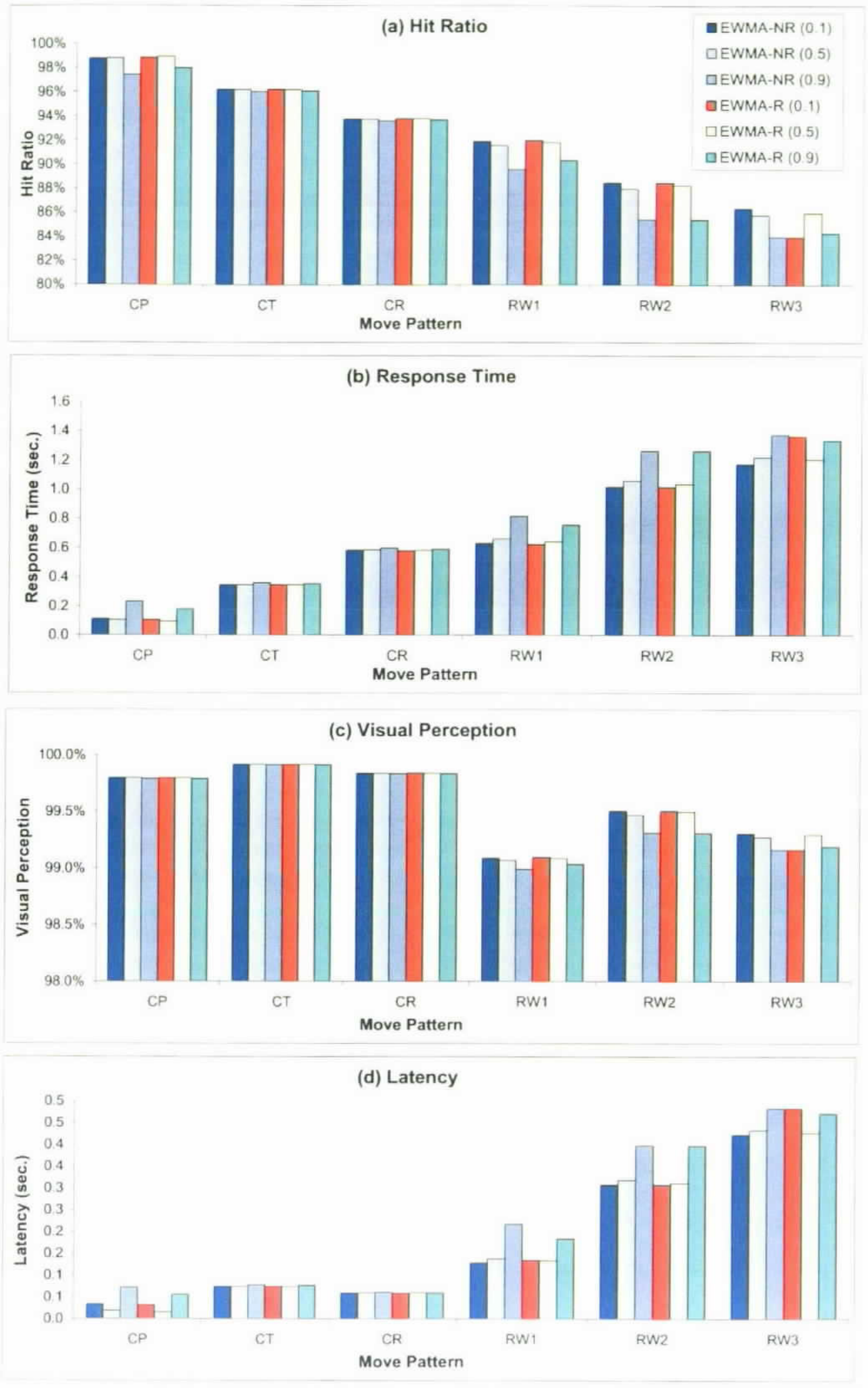


Figure 5.11: Performance of Experiment #7.

tially decreasing weight, α, for achieving the best performance of the caching and prefetching mechanisms under EWMA schemes is 0.5. Thus, we will fix the value of α as 0.5 in the following experiments.

5.6.2 Experiment #8

In this experiment, we would like to study the performance of our proposed caching mechanism, with or without prefetching, on various moving patterns. In this experiment, N is fixed at 5000 object, resulting in a database size of approximately 1.72GB. The cache size is fixed at 5MB (0.284%). We experiment all the six moving patterns: CP, CT, CR, RW1, RW2 and RW3. We repeat the experiments with various prefetching schemes: Mean, Window, EWMA-NR and EWMA-R, to be compared with the base case in which no prefetching is used, i.e. No Prefetch. The measurements of the four metrics are depicted in Figure 5.12.

From Figure 5.12, we observe that even without prefetching, the caching mechanism performs very well. It achieves a hit ratio ranging from 81% to 94% (Figure 5.12(a)) among the various moving patterns. With prefetching, the hit ratio could be improved by up to 16%. This results in a decrease in response time (Figure 5.12(b)). We observe that Mean is not effective in future movement prediction. Both Window and EWMA perform equally well.

Under the moving patterns of CT and CR, caching without prefetching can achieve a relatively high hit ratio of 96% and 94% respectively, and prefetching can only achieve an improvement of 2%. All the prefetching schemes perform equally well under these two moving patterns. For the other moving patterns, a window-based prefetching scheme with a smaller window size, W, results in a better performance than that with a larger window size. This is mainly because under these moving patterns, the moving direction is always changing. With a large window size, aged

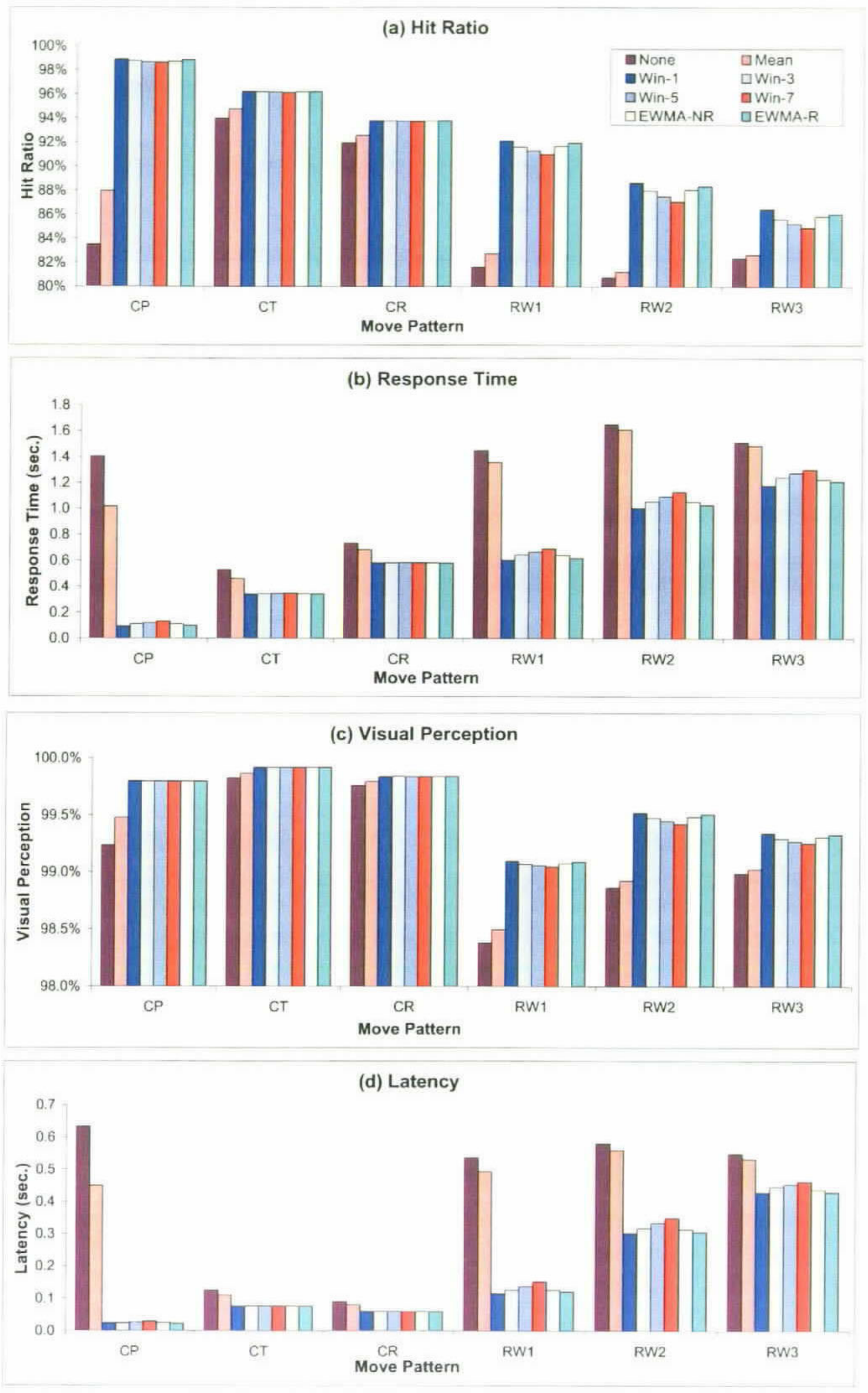


Figure 5.12: Performance of Experiment #8.

moving vectors contribute to the prediction of the next moving vector and this introduces noise in the prediction. The performance is thus not as good as that with a small window size. EWMA exhibits a similar behavior as window-based scheme. EWMA-R performs better than EWMA-NR because EWMA-R can successfully predict and correct the errors incurred due to the "non-stationary" changes of viewing direction.

From this experiment, we can conclude that prefetching is very useful in improving the performance of the caching mechanism. A window-based prefetching scheme with a small W and EWMA-R performs better than the other prefetching schemes.

5.6.3 Experiment #9

In this experiment, we would like to examine the effect of various prefetching schemes at different cache sizes. Again, N is fixed at 5000 objects and the cache size ranges from 2MB to 10MB (0.114% to 0.568%). The moving pattern is fixed at CP and transmission pattern is Vx. We repeat our experiments with various prefetching schemes and the base case of no prefetching. The results of the experiment are depicted in Figure 5.13.

From Figure 5.13(a), we observe that without prefetching, the hit ratio results in a maximum value of 83.5% when the cache size is greater than 3MB (0.17%) and further increase in cache size does not improve the caching performance. When the cache size increases, both the objects within the viewing region and the prefetched objects can be retained in the cache and no replacement is carried out. Thus, the cache must be large enough in order to make the prefetching mechanism effective. We also notice that Mean can achieve a maximum hit ratio of 88.0% and all the other prefetching schemes can achieve a maximum hit ratio of 99.0% when the cache size is greater than 5MB (0.284%). This shows that all the prefetching schemes, except

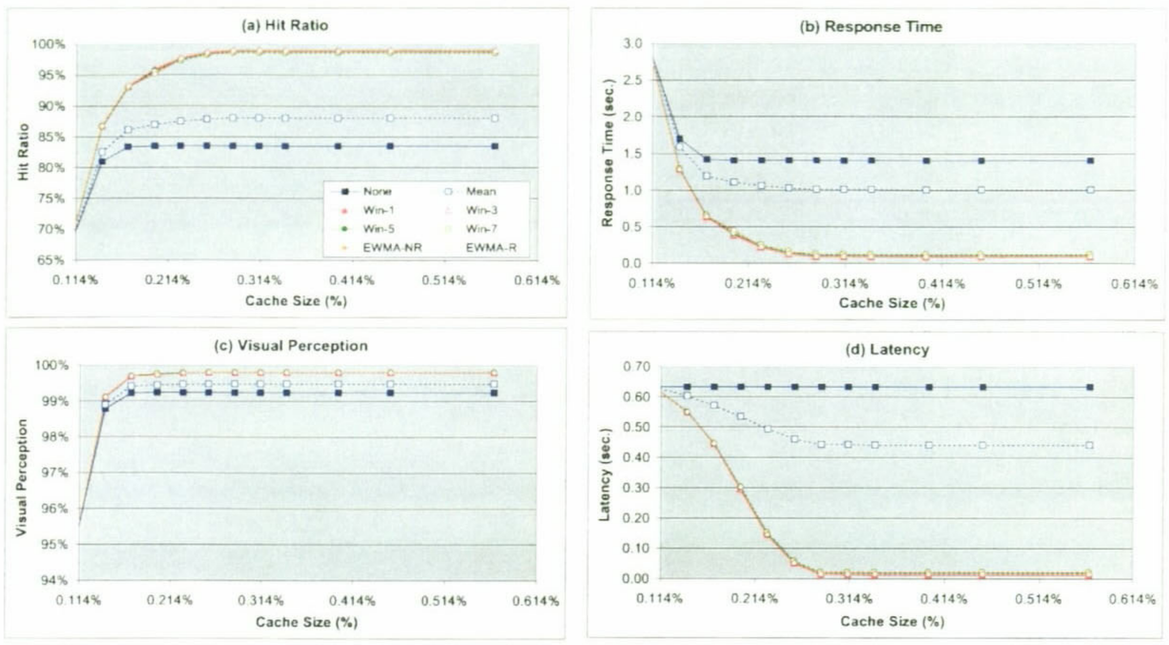


Figure 5.13: Performance of Experiment #9.

Mean, are capable of predicting the next movement vector of the viewer accurately. This accounts for the significant reduction in response time and latency as depicted in Figure 5.13(b) and Figure 5.13(d).

From this experiment, we can conclude that the cache must be large enough in order to make the prefetching mechanism more effective.

5.6.4 Experiment #10

In this experiment, we would like to further examine the effect of the various prefetching schemes at different cache sizes. The moving pattern is fixed at RW3 and all the other parameters are the same as those in Experiment #10. The results of this experiment are depicted in Figure 5.14

We observe from Figure 5.14(a) that the cache hit ratio increases as the cache size

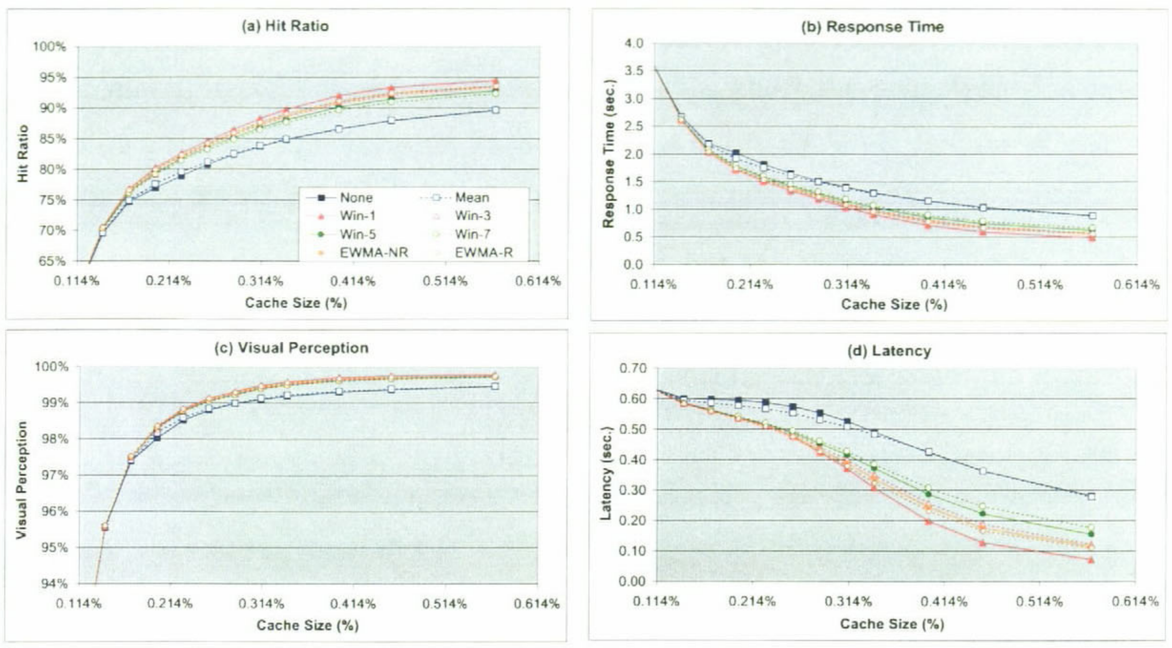


Figure 5.14: Performance of Experiment #10.

increases. Even without prefetching, the caching mechanism can achieve a relatively high hit ratio of 89.8% when the cache size is 10MB (0.568%). When prefetching is enabled, the caching mechanism can achieve an improvement of up to 5%. This accounts for the reduction of response time and latency when the cache size increases. Under the moving pattern of RW3, a viewer has a high probability of turning around and objects being accessed in the past have a high probability of being accessed in the future. Thus, when the cache is large enough, those objects having been accessed in the previous movement steps are retained longer. This results in an increase in the hit ratio and visual perception.

Since the viewer turns around frequently, Mean is unable to cope with the changing moving pattern and can only have a small improvement on the caching mechanism as compared with the base case of no prefetching. Among the other prefetching schemes, Win-1 achieves the best performance because with a small window size, aged moving vectors do not contribute to and thus affect the prediction. EWMA exhibits a similar behavior as the Window schemes. Also, EWMA-R performs slightly better than EWMA-NR.

From this experiment, we can also conclude that the cache size must be large enough in order to make the prefetching mechanism effective. Win-1 and EWMA-R result in a better performance than the other prefetching schemes under the moving pattern of RW3.

5.6.5 Experiment #11

In this experiment, we would like to study the effect of database size on the performance of caching and prefetching mechanisms. In this experiment, N ranges from 2500 objects, 5000 objects, 10000 objects, to 20000 objects, leading to a database size of 0.68GB, 1.72GB, 3.44GB and 6.88GB respectively. The cache size is 0.284%. The moving pattern is CP and the replacement policy is MRM-0.1. The experiments are repeated with different prefetching schemes. The result of the experiment is depicted in Figure 5.15.

From Figure 5.15(a) and Figure 5.15(c), we observe that the hit ratio and visual perception for various prefetching schemes are almost invariant to the different database sizes because the cache size is defined as a percentage of the database size. The increase in database size also increases the actual storage for caching. We also observe that except Mean, all the other prefetching schemes can achieve an improvement of approximately 15% in hit ratio as compared to the base case of no prefetching. There is a slight increase in visual perception as the database size increases. It is because the increase in database size results in an increase in the object density. Therefore, at any location, the likelihood that the viewer scope overlaps with the object scope becomes higher. This increases the chance of hitting an object in the

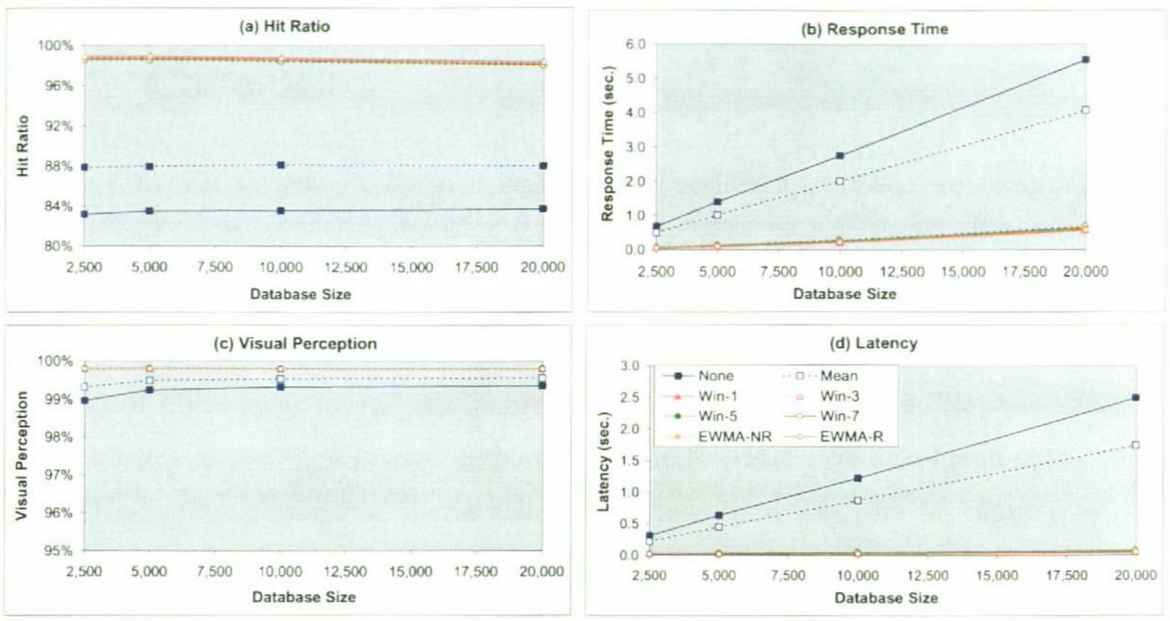


Figure 5.15: Performance of Experiment #11.

local cache as the viewer moves.

We notice from Figure 5.15(*b*) and Figure 5.15(*d*) that the response time and latency is directly proportional to the database size. As the database size increases, more objects are visible to the viewer and thus more objects will be transmitted across the network. It is promising that except the Mean scheme, all the other prefetching schemes can achieve a significant improvement on reducing the response time and latency by up to 12 and 42 times respectively as compared to the base case.

From this experiment, we can conclude that all the prefetching schemes, except the Mean scheme, are very effective in reducing the response time and latency of the system.

5.7 Conclusion

In this chapter, we have studied and evaluated the performance of our proposed *multi-resolution caching mechanism* and *user-profiling based prefetching mechanism* through a set of simulated experiments. We have shown that the conventional LRU replacement policy is ineffective in the context of DVE across the Internet. By contrast, our proposed MRM replacement policy can significantly reduce both the response time and latency of the DVE system by improving the hit ratio. We also found that the Vx transmission pattern results in a lower response time than the Cx counterpart. In addition, MRM with a smaller adjustment parameter, ω, performs better than that with a larger one and the optimal value of ω is 0.1. MRM, like other caching mechanisms, is relatively invariant to the database size being accessed.

We have shown that the various prefetching schemes as described in Section 4.6 are complemented with the caching mechanism and are useful in further reducing the response time and latency of the DVE system. Even without prefetching, the caching mechanism alone can achieve a relatively high hit ratio and prefetching can further improve the performance. We also found that the cache size must be large enough in order to make prefetching effective. Among the various prefetching schemes, Win-1 and EWMA-NR perform the best in general. The Mean scheme performs the worst as compared to other schemes and is unable to reduce the response time of the system by much. In addition, we have determined that the optimal value of the exponentially decreasing weight, α, in the EWMA prefetching schemes is 0.5.

Chapter 6

System Prototype

6.1 Introduction

A system prototype has been implemented for evaluating the performance of our proposed *Multi-resolution Caching Mechanism* [12]. The prototype is implemented using Java, due to its platform independence nature. We aim at developing a DVE system for commonly available personal computer systems, without special graphics accelerators or extremely fast CPUs as found in high-end graphics workstations. The system should allow users to navigate through different VE maintained by various remote servers via the Internet.

There are several challenges that have to be tackled in order to make the prototype workable. A desirable DVE system should provide users with smooth and real-time rendering of the VE. The system should be responsive such that the images must be updated within a short period of time upon receiving input from the user. The system should also connect to remote server, send request to it and receive data from it with minimal overhead.

6.2 System Architecture

The DVE system is based on a client-sever model. The architecture is therefore divided into two main parts, the *Client System* and the *Server System*. The Client System and the Server System consist of 4 and 5 main subsystems respectively as shown in Figure 6.1. The major functions of the subsystems are as follows:

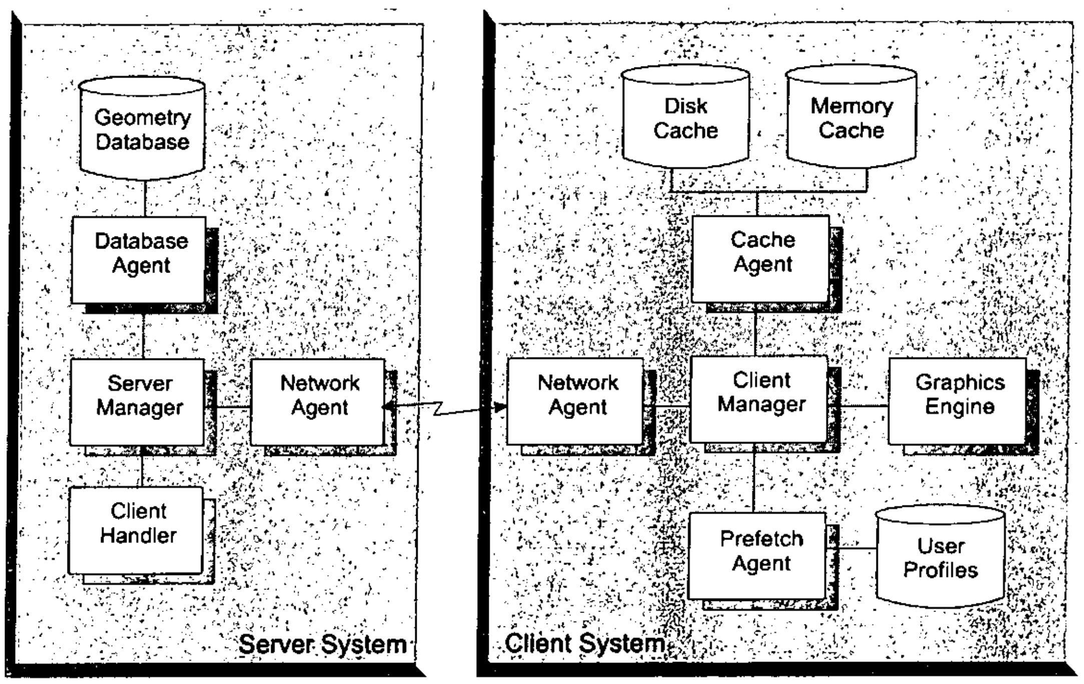


Figure 6.1: Architecture of the prototype.

Client System

- *Client Manager*: It serves as the coordinator of all other subsystems at the Client System. All inputs from the user, such as translation or rotation, are directed to and handled by the Client Manager. It also maintains the VE index received from the server.

- *Cache Agent*: It controls all the local caches, including the memory and/or the storage cache at the Client System. Whenever the client receives data from the server, the data would be cached via the Cache Agent. The agent performs cache cleanup at the underlying cache(s) when necessary. Each object in the local cache is associated with an access score, dictating the replacement order. The process is implemented as a separate thread for cache management.

- *Network Agent*: It handles all the communications between the client and the server, and maintains the connection between the client and the server once a connection has been established between them. It also handles all the object requests. The process is implemented as a separate thread under the Client Manager in order to reduce the response time.

- *Prefetch Agent*: It maintains some historical movement vectors of each viewer in the form of user profiles. It then predicts the next position of the viewer based on the profiles and the Client System prefetches data according the predicted position. A number of prefetching schemes are implemented, including Mean, Window, and EWMA.

- *Graphics Engine*: It renders the VE into images for display and also accepts input from the user. The rendered images are be updated in a timely fashion.

Server System

- *Server Manager*: It serves as the coordinator of all other subsystems at the server side.

- *Database Agent*: It maintains the database of the virtual environment. Each virtual object in the database is in the form of progressive mesh containing a base mesh and a list of progressive records. The database agent allows fast retrieval of data from secondary storage.

- *Client Handler*: It receives requests from each client, processes the requests and sends the requested data back to the client. It is implemented as a separate thread for each connected client in order to reduce the response time.

- *Network Agent*: It handles all the communications between clients and the server and maintains the connection between each client and the server once a connection has been established between them. When a client requests connection, it creates a separate Client Handler process to service client requests.

6.3 System Design

6.3.1 Client System

The class diagram of the Client System is shown in Figure 6.2. The Client System is divided into 5 subsystems, which includes Client Manager, Cache Agent, Network Agent, Prefetch Agent and Graphics Engine (see Figure 6.1).

Client Manager

- *ClientManager* is the main coordinator of the Client System and is responsible for controlling all the other subsystems of the client system. It handles inputs from the user.

- *VEIndex* contains the received VE index from the server and is responsible for determining the set of objects within the viewer scope.

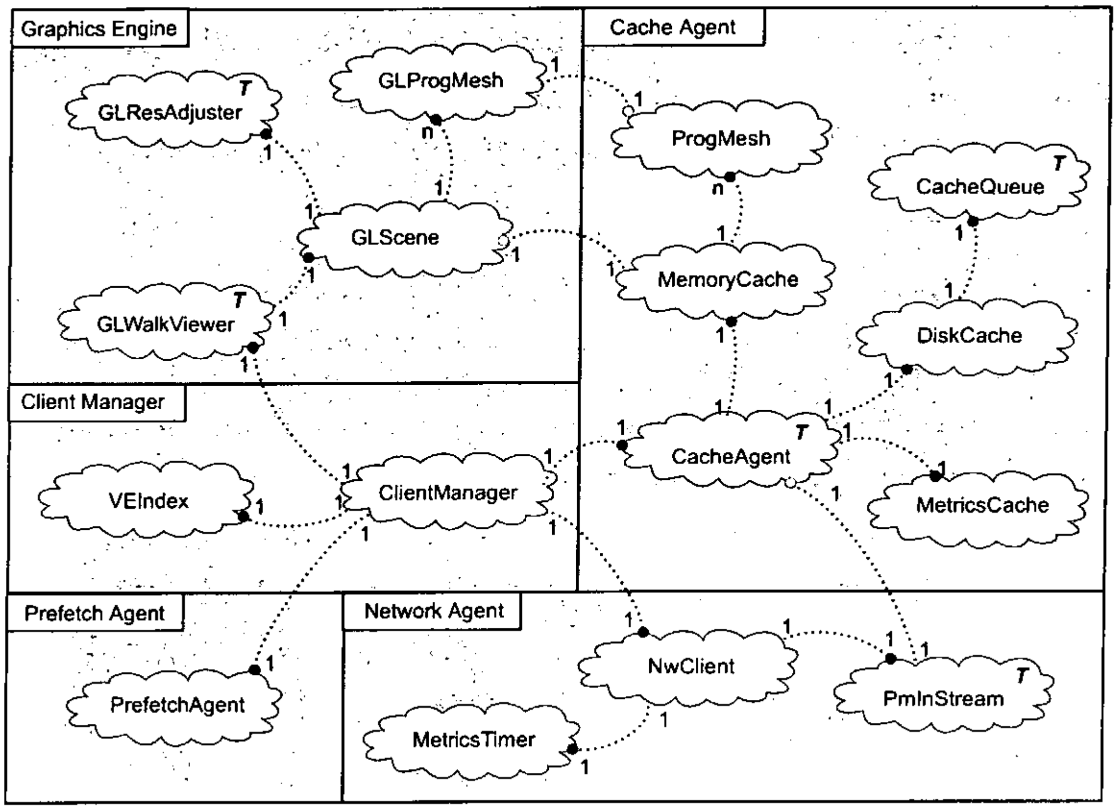


Figure 6.2: Class diagram of the Client System.

Cache Agent

- *CacheAgent* is a thread for processing data received from the server. It contains a shared buffer which is implemented as a synchronized FIFO queue. It retrieves data from the buffer and put the data into the memory and/or disk cache. It is also responsible for building the request list and prefetch list when requesting and prefetching data from the server.
- *MetricsCache* is responsible for calculating the hit ratio and visual perception whenever a request is generated.

- *DiskCache* is the disk cache for maintaining the received data in the secondary storage. It contains the *CacheQueue* for writing data to the secondary storage. When data is put into the disk cache, it checks whether the data should be cached or not. If the data is allowed to be cached, it puts the data into the *CacheQueue*.

- *CacheQueue* is a thread for writing data to the secondary storage. It prevents the prototype from being blocked by the time-consuming I/O processing.

- *MemoryCache* is the memory cache for controlling the content of *GLScene* for rendering. When a new *ProgMesh* is introduced, it creates an instance of *GL-ProgMesh* and adds it to the *GLScene*. When the cache storage space is exhausted, it selects a replacement victim using our proposed MRM replacement policy. The victim is then removed from the cache and rendering scene.

- *ProgMesh* represents a virtual object in the form of progressive mesh. It contains a base mesh and a list of progressive records.

Network Agent

- *NwClient* is responsible for establishing connection between the clients and servers and maintaining the connections between them once established. It is also responsible for sending requests to server for missing data.

- *PmInStream* is a thread for reading data from the server. It checks the incoming data stream regularly. When data is available, it reads the data and converts the incoming binary data stream into various data types according to the received header. The data is then transferred to the *CacheAgent*. The thread is suspended for a period of time whenever no data is available from the data stream.

- *MetricsTimer* is responsible for measuring the latency and response time of the system.

Prefetch Agent

- *PrefetchAgent* predicts the next position of the viewer based on the historical movement vectors. The predicted position is then used to prefetch data from the server utilizing the otherwise idle network. A number of prefetching schemes are implemented including Mean, Window, and EWMA.

Graphics Engine

- *GLWalkViewer* is responsible for generating the images of the virtual environment for display. It sends all the visible objects to the graphics hardware for rendering. It also accepts input from the user, which may either be mouse events or keyboard events, and processes the events accordingly.

- *GLScene* maintains the content of the virtual environment for rendering. During rendering, the content is traversed and the set of visible objects is determined through bounding-box test.

- *GLResAdjuster* is a thread for adjusting the resolution of each of the objects in the *GLScene* with respect to the position and orientation of the viewer. A priority queue is implemented for resolution adjustment. This can improve the response time of the system by updating the resolution of visible objects in front of the viewer prior to the invisible objects.

- *GLProgMesh* represents the polygon model of each virtual object for rendering. It contains a vertex array and a polygon array and is capable of adjusting the resolution of the model using the progressive record list.

6.3.2 Server System

The class diagram of the Server System is shown in Figure 6.3. The Server System is divided into 4 main subsystems, which includes Server Manager, Database Agent, Client Handler and Network Agent (see Figure 6.1).

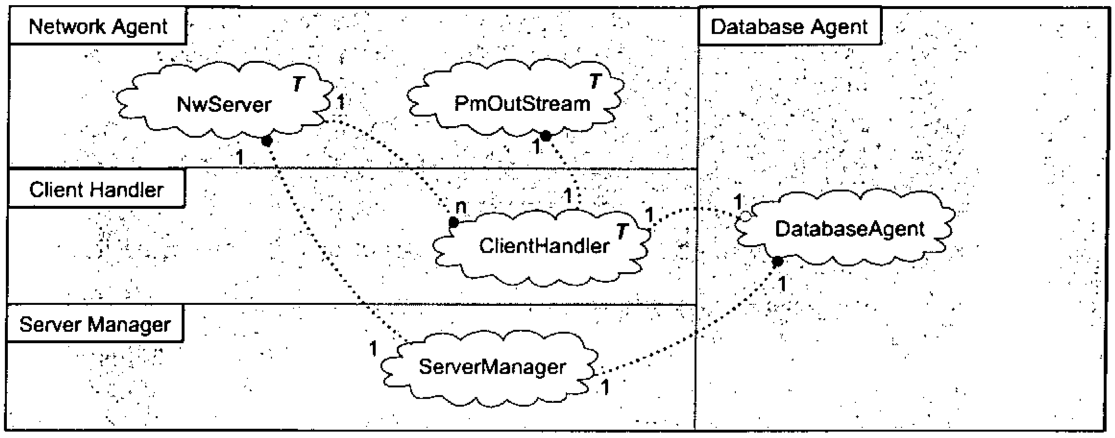


Figure 6.3: Class diagram of the Server System.

Server Manager

- *ServerManager* is the central coordinator of all subsystems at the Server System.

Database Agent

- *DatabaseAgent* maintains the database of the virtual environment and retrieves the requested data from the secondary storage.

Network Agent

- *NwServer* is a thread which listens to a well-known port and accepts connection requests from different clients. Once the connection is established, it creates a new instance of *ClientHandler* to serve future requests of the client.
- *PmOutStream* sends the requested data back to the client and is implemented as a separate thread in order to reduce the response time of the system.

Client Handler

- *ClientHandler* is also a thread which accepts requests from its connected clients, processes the requests and requests data from *DatabaseAgent*. The results are passed to the *PmOutStream*.

6.4 Implementation Details

6.4.1 Multi-resolution Modeling

Model Generation

The generation of multi-resolution modeling is divided into 2 pre-processing stages, *Generation of simplification list* and *Generation of progressive mesh*, as shown in Figure 6.4.

In [43], several visual importance factors are identified. For simplicity, we only employ static visual importance factors for constructing a simplification list of the original input model. The simplification list stores the sequence of edge collapse operations according to the static visual importance of each vertex of the original input model.

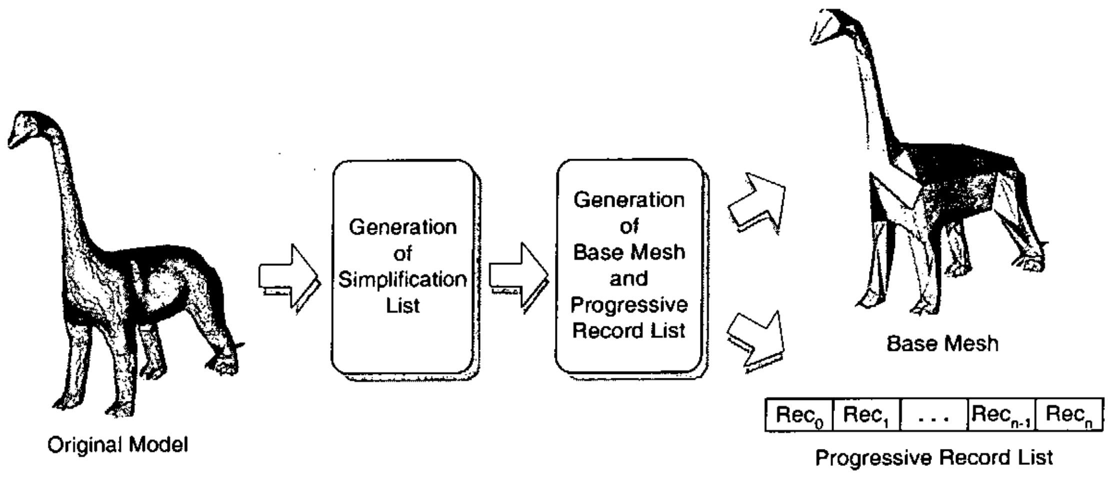

Original Model
Generation of Simplification List
Generation of Base Mesh and Progressive Record List
Base Mesh
Rec_0 Rec_1 ... Rec_{n-1} Rec_n
Progressive Record List

Figure 6.4: Generation of a multi-resolution model.

After generating the simplification list, we construct a base mesh and a list of progressive records from the simplification list. Since we try to preserve the feature edges of the original model when generating the simplification list, the edge collapse sequence in the simplification list is different from the vertex sequence of the original model. As a result, we rearrange the simplification list such that the newly introduced vertex and polygons for each refinement are always the last elements of the arrays. This can prevent insertion of vertices and polygon indices in the middle of the arrays for efficient memory utilization.

Model Transmission

To transmit a multi-resolution model to the client, the base mesh is transmitted first as a single unit. The client reconstructs the minimal resolution model of the object as it receives the base mesh. Each subsequent progressive record is transmitted in order. Each record stores information for splitting an edge of the object model, thereby increasing the resolution of the object model by one level. As an edge is inserted or split in a local mesh, a vertex needs to be divided as well. This idea is illustrated in Figure 6.5. In Figure 6.5, as an edge identified by two vertices, V_{parent} and V_{child}, is inserted into a local mesh. Some neighboring triangles also need to be divided.

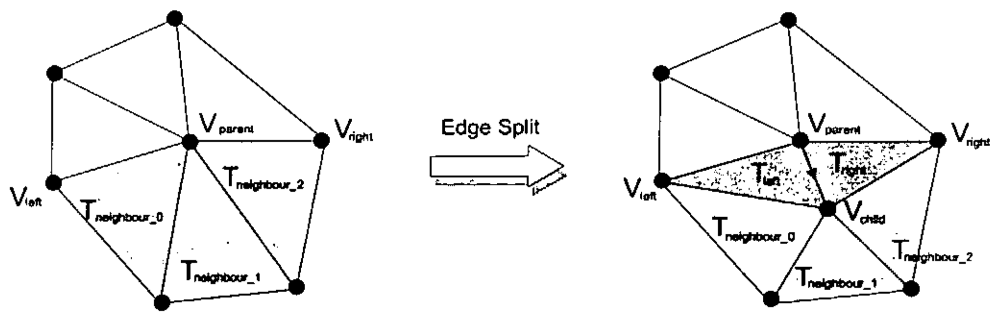


Figure 6.5: A progressive record stores information for an edge split.

The progressive record for the edge split operation shown in Figure 6.5 contains the $\langle x, y, z\rangle$ position of V_{child}, i.e. the vertex to be inserted, and the ID of the parent vertex, i.e. V_{parent}. V_{parent} then joins with V_{child} to form the inserted edge. We also need to transmit the IDs of two vertices,V_{left} and V_{right}. They help to identify the locations where the two new triangles are to be inserted. The two triangles are defined as $T_{left} = \langle V_{parent}, V_{child}, V_{left}\rangle$ and $T_{right} = \langle V_{child}, V_{parent}, V_{right}\rangle$. In order to reduce the time for retriangulating the original model after the edge split, we also need to transmit the IDs of neighboring triangles, i.e. $T_{neighbour_n}$, for all triangles which are connected to V_{parent}. By modifying V_{parent} to V_{child} in these triangles, the resolution modification is completed. The data structure for our implementation of the multi-resolution modeling is shown in Figure 6.6.

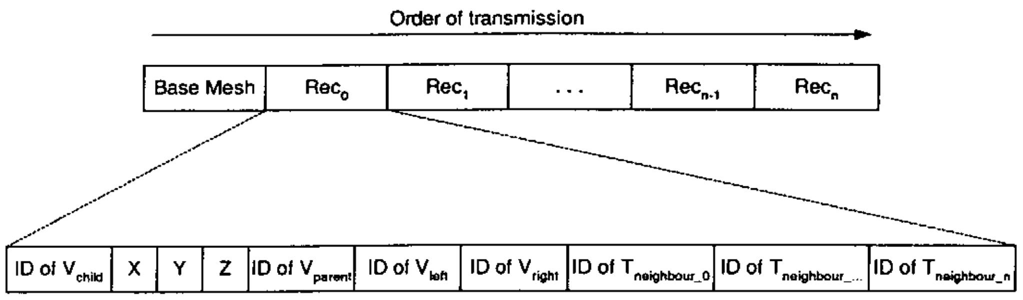


Figure 6.6: The data structure of a multi-resolution model for transmission.

6.4.2 Graphics Library

A graphics library is needed for rendering and visualizing the virtual environment. *Open Inventor for Java* is used for rendering in the first implementation of the prototype. However, we found that the library has some problems and is unsuitable for use in our prototype system. We then chose *GL4Java* in our second implementation of the prototype system. The details of the two libraries will be discussed as follows.

Open Inventor for Java

Open Inventor for Java is developed by *TGS* as a set of wrapper classes for the graphics library *Open Inventor for C++* [80]. It provides a Java interface to all the classes of *Open Inventor* and allows fast rendering by utilizing local graphics hardware at the client machine. In order to render the content of the virtual environment, it is necessary to construct a three dimensional scene using the graphics library. The library provides a component for displaying the content and allows users to navigate through the VE.

The first implementation of the prototype used *Open Inventor for Java* as the graphics library for rendering. After the implementation, the prototype crashed intermittently. Concurrent data accesses of the graphics library caused memory access violation. As data was constantly received from the server, the scene of the virtual environment was constantly updated. As there was no synchronization primitive being implemented in the library and the library was protected, it was impossible to implement custom synchronization primitives using this library. This made the library unsuitable for developing DVE system.

GL4Java

GL4Java is a relatively new and non-commercial graphics library developed by *Jausoft*. It is a set of wrapper classes for the graphics library *OpenGL*. Similar to *Open Inventor for Java*, it provides a Java interface to all the methods of *OpenGL* and allows fast rendering by utilizing graphics hardware at the client machine.

The second implementation of the prototype system used *GL4Java* as the graphics library. The performance of *GL4Java* was better than *Open Inventor for Java*. In addition, *GL4Java* permits custom synchronization primitive to be implemented. However, programming is at a lower level.

6.5 Experiments for Simulation Validation

We have developed a prototype system using *Java* for evaluating the proposed caching mechanism. In order to validate the prototype system against the simulation model as described in Chapter 5, we have conducted some experiments and compared the prototype results against the simulated experimental results. In the prototype experiments, the server runs on an UltraSparc 5 station with 512MB RAM. The client runs on a 450MHz Pentium II PC with 256MB RAM. The parameter settings of the prototype experiments are the same as the simulated experiments as described in Chapter 5. The parameter settings of the two prototype experiments are summarized in Table 6.1.

Parameter	Values for each experiment	
Experiment	#A	#B
N	5000	
n	0.057%	0%, 0.014%, 0.028%, 0.043% 0.057%, 0.071%, 0.085%, 0.114% 0.142%, 0.170%, 0.227%, 0.284%,
R_{disk}	MRM	
ω	0.1	
F_{disk}	No Prefetch	
W	N/A	
α	N/A	
P	CP, CT, CR, RW1, RW2, RW3	CP
T	Vx	

Table 6.1: Parameter values for the prototype experiments.

6.5.1 Experiment #A

In the first experiment, we would like to study the effect of different moving patterns on the performance of caching mechanism. This experiment is aimed at validating Experiment #3 in Chapter 5. With N being fixed at 5000 objects (1.72GB) and n fixed at 1MB, i.e., 0.057% of N, we experiment with the six moving patterns. The replacement policy is MRM with adjustment parameter, ω, 0.1. We only present the metrics for different policies under the Vx transmission pattern that is shown to perform better in the previous experiment. The results are depicted in Figure 6.7.

We observe from Figure 6.7(a) and Figure 6.7(c) that the prototype system and the simulated experiment result in the same hit ratio and visual perception. The caching mechanism in both the prototype system and the simulation model can achieve the same caching performance under various moving patterns.

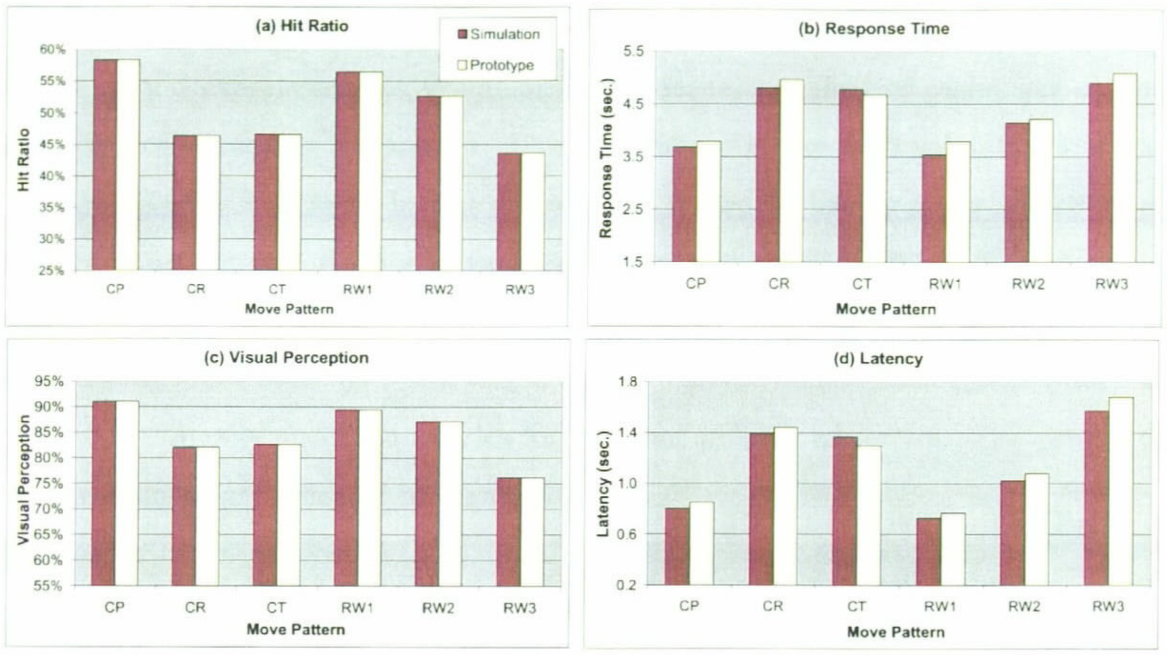


Figure 6.7: Performance measurements of Experiment #A.

We also observe from Figure 6.7(*b*) and Figure 6.7(*d*) that except CR, the prototype system results in a slightly higher response time and latency under the other moving patterns as compared to the simulation model. This is because the network traffic is relatively unpredictable and the network bandwidth available for the prototype experiments may vary under different conditions. In addition, the increase in both the response time and latency is due to the processing overhead in the prototype system, which is not modeled in the simulation.

We can conclude that the simulation model can correctly measure the performance of the caching mechanism in terms of cache hit ratio and visual perception. This can be validated through the experiments on the prototype system. The response time and latency from the prototype system is slightly higher than that of the simulation model due to the processing overhead.

6.5.2 Experiment #B

In the second experiment, we would like to examine the effect of cache size on the performance of caching mechanism. This experiment is aimed at validating the Experiment #4 in Chapter 5. In this experiment, we would like to study the effect of cache size on the performance of various replacement policies. With N being fixed at 5000 objects and moving pattern being fixed at CP, we experiment with the cache size ranging from 0MB to 5MB, i.e., 0.0% to 0.284%. The replacement policy is MRM-0.1. We only show the metrics for different policies under Vx. The results of this experiment are depicted in Figure 6.8.

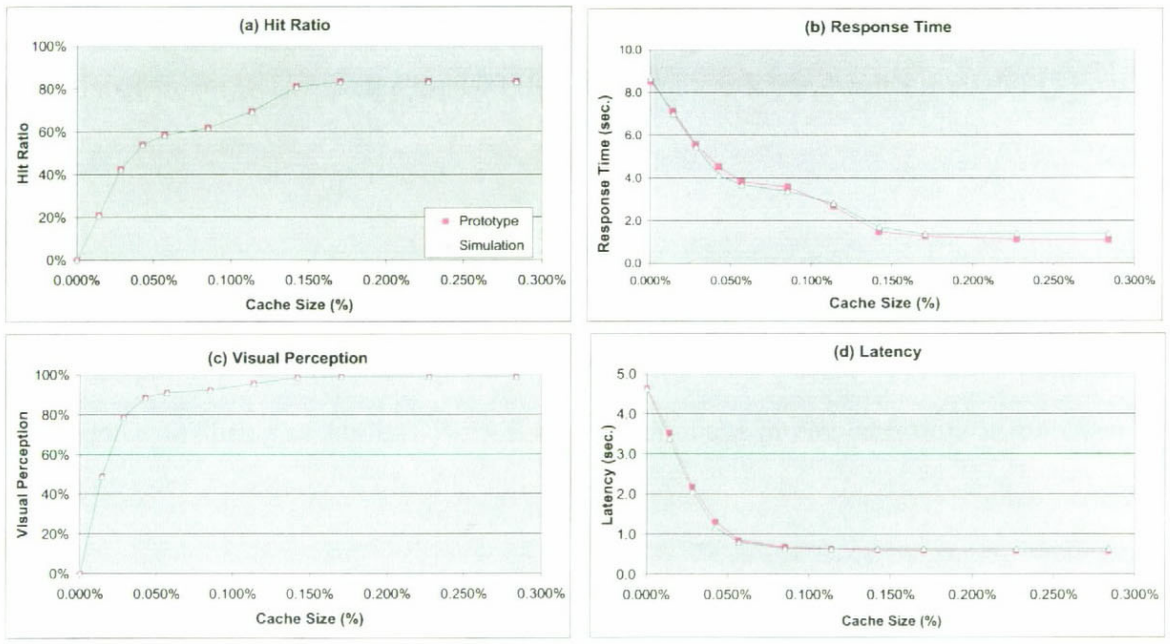


Figure 6.8: Performance measurements of Experiment #B.

From both Figure 6.8(a) and Figure 6.8(c), we observe that the prototype system and the simulated experiment produce the same hit ratio and visual perception again. This shows that the caching mechanism in both the prototype system and the simulation model can achieve the same caching performance with different cache size. The hit ratio increases from 0 to a maximum of 83% and the visual perception increases from 0% to 99% when the cache size increases.

We notice from Figure 6.8(c) and Figure 6.8(d) that the response time and latency decrease as the cache size increases. Both the prototype system and the simulation model show a similar trend. It is interesting that the response time and latency of the prototype system slightly deviates from those of the simulation model. It is because the network bandwidth availability for prototype experiments changed over time and could not be kept in a constant value.

We can conclude that the simulation model can correctly measure the hit ratio and visual perception. The response time and latency of the prototype system are slightly deviated from the simulation model because the network bandwidth availability may change over time.

6.6 Prototype Demonstration

We would like to demonstrate the implemented prototype system by showing different screen shots during execution. We will show both of the implementations using *Open Inventor for Java* and *GL4Java*.

Implementation using Open Inventor for Java

Figure 6.9 shows the execution of the prototype system in wireframe rendering and Figure 6.10 shows the execution in shaded rendering. In this implementation, the user interface is based on the *AWT*.

Implementation using GL4Java

Figure 6.11 shows the execution of the prototype system in wireframe rendering and Figure 6.12 shows the execution in shaded rendering. The user interface in this implementation is based on *Swing.*

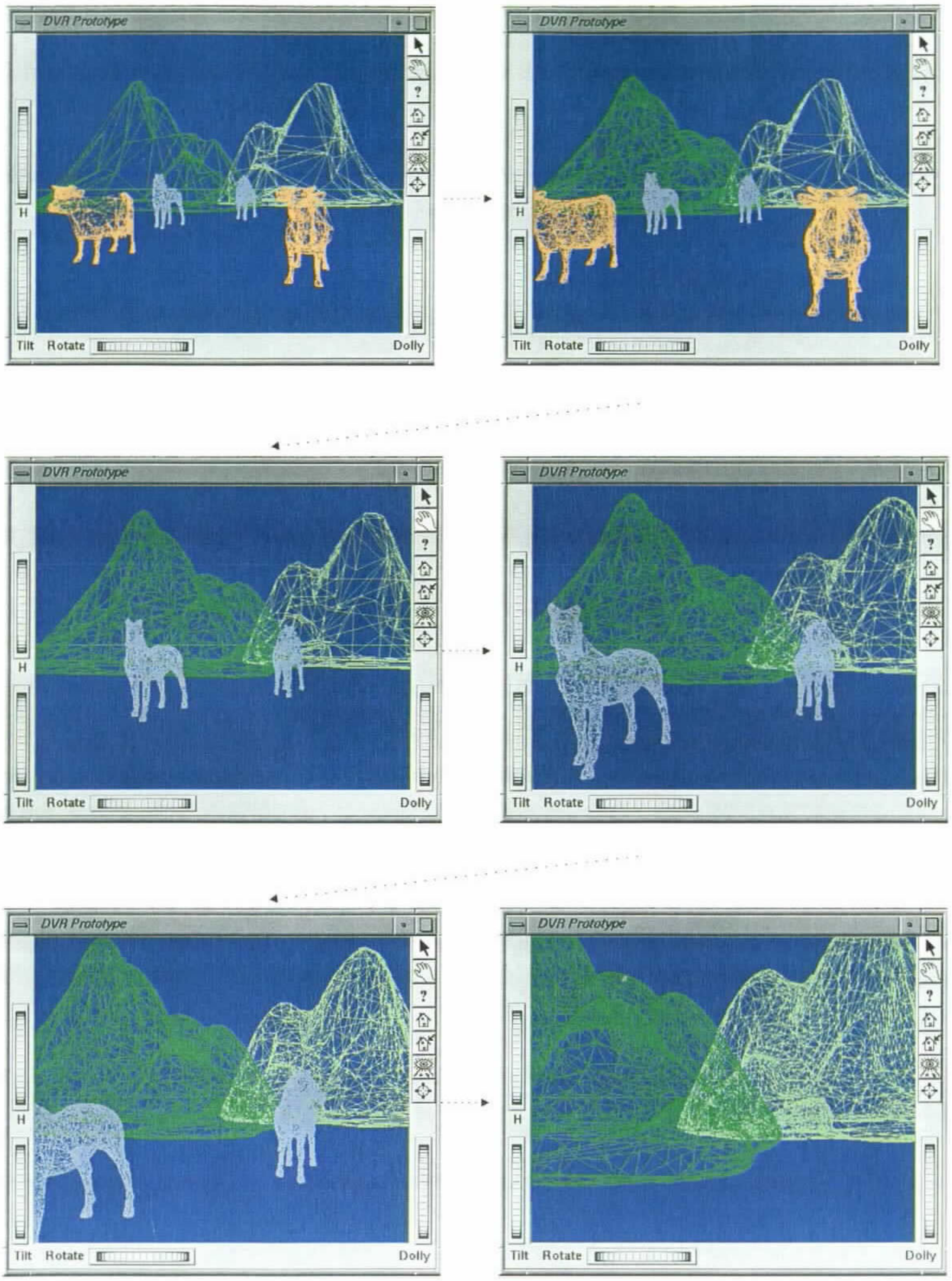


Figure 6.9: Demonstration of the first prototype implementation in wireframe rendering.

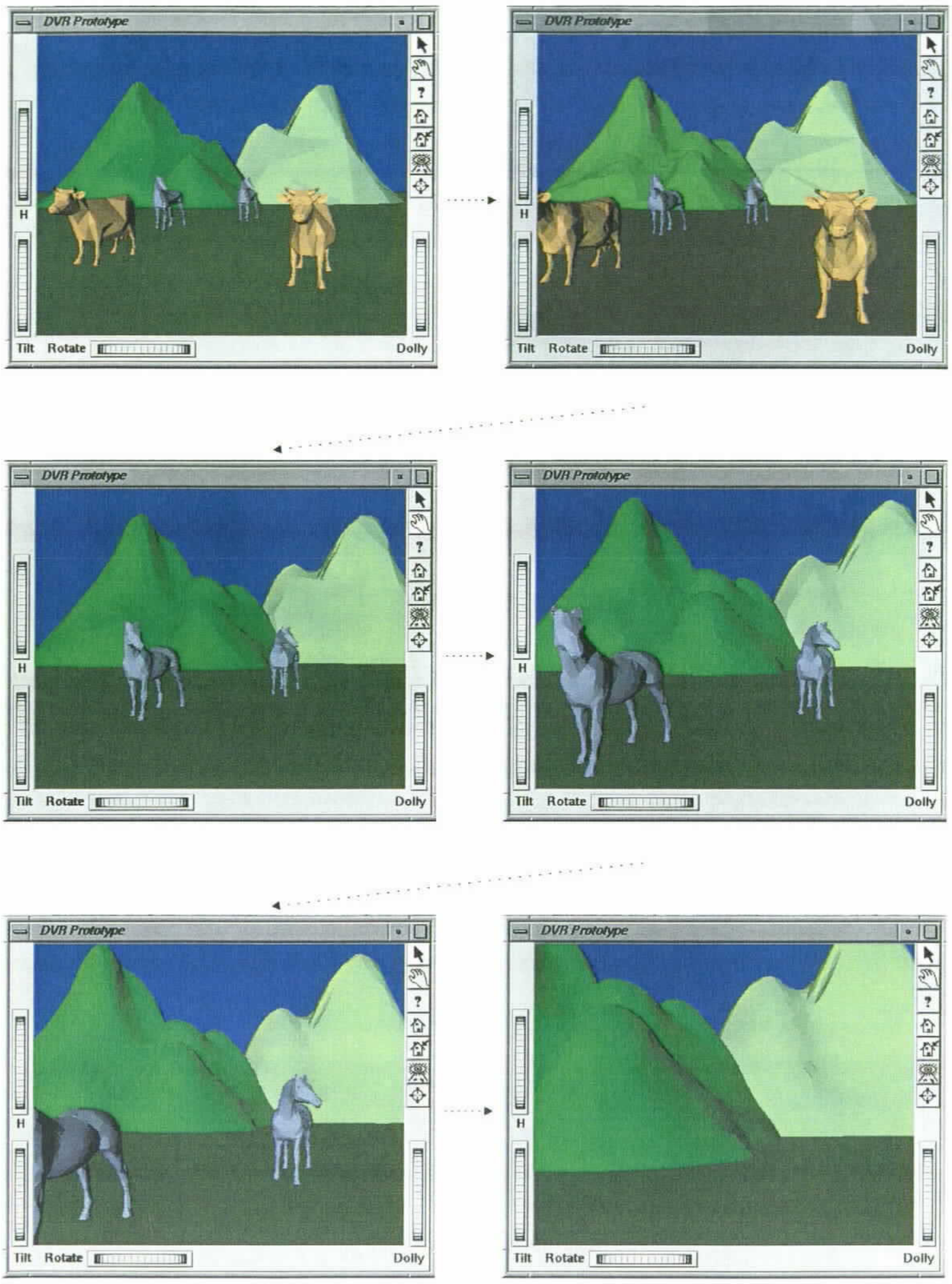


Figure 6.10: Demonstration of the first prototype implementation in shaded rendering.

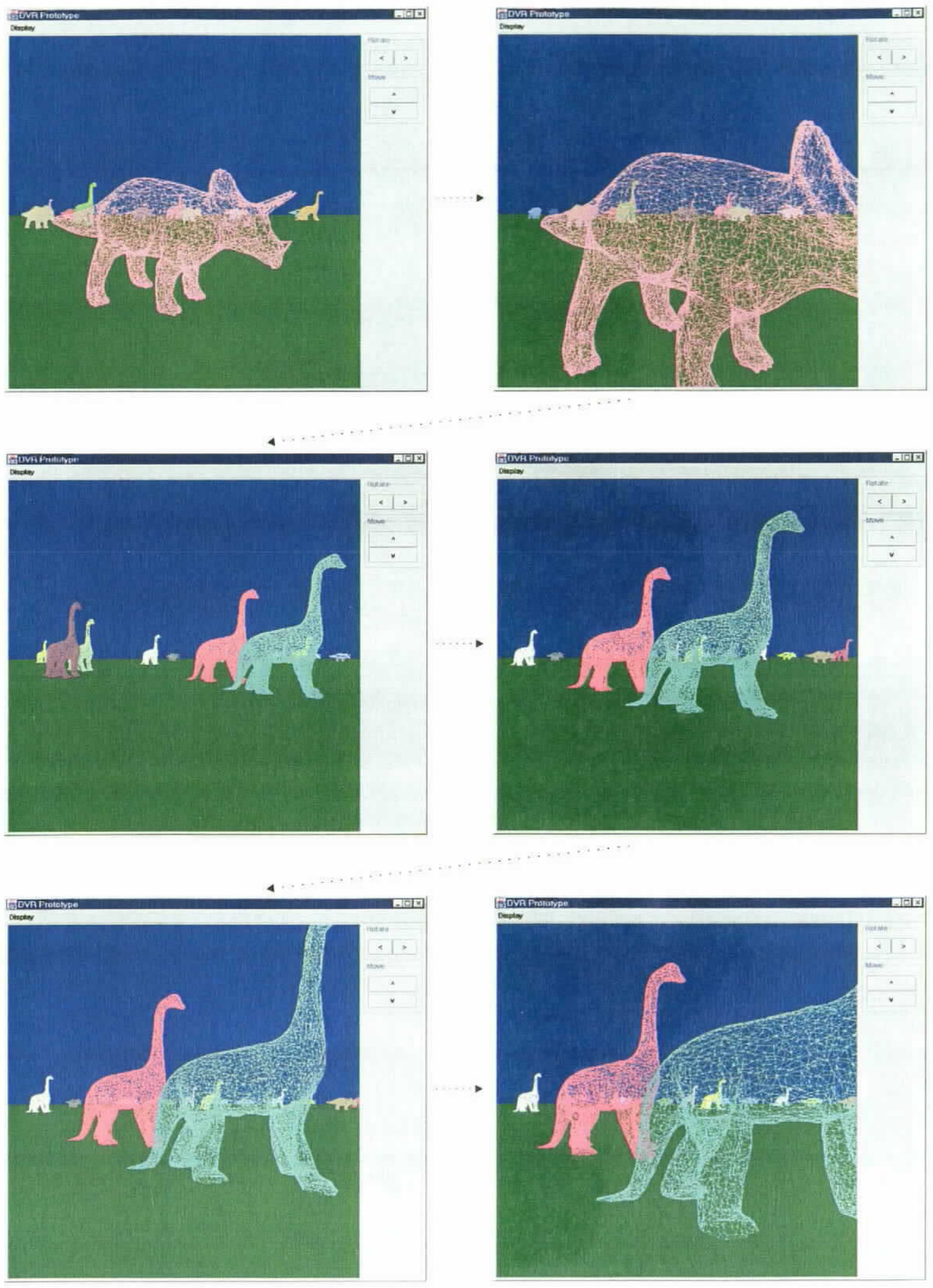


Figure 6.11: Demonstration of the second prototype implementation in wireframe rendering.

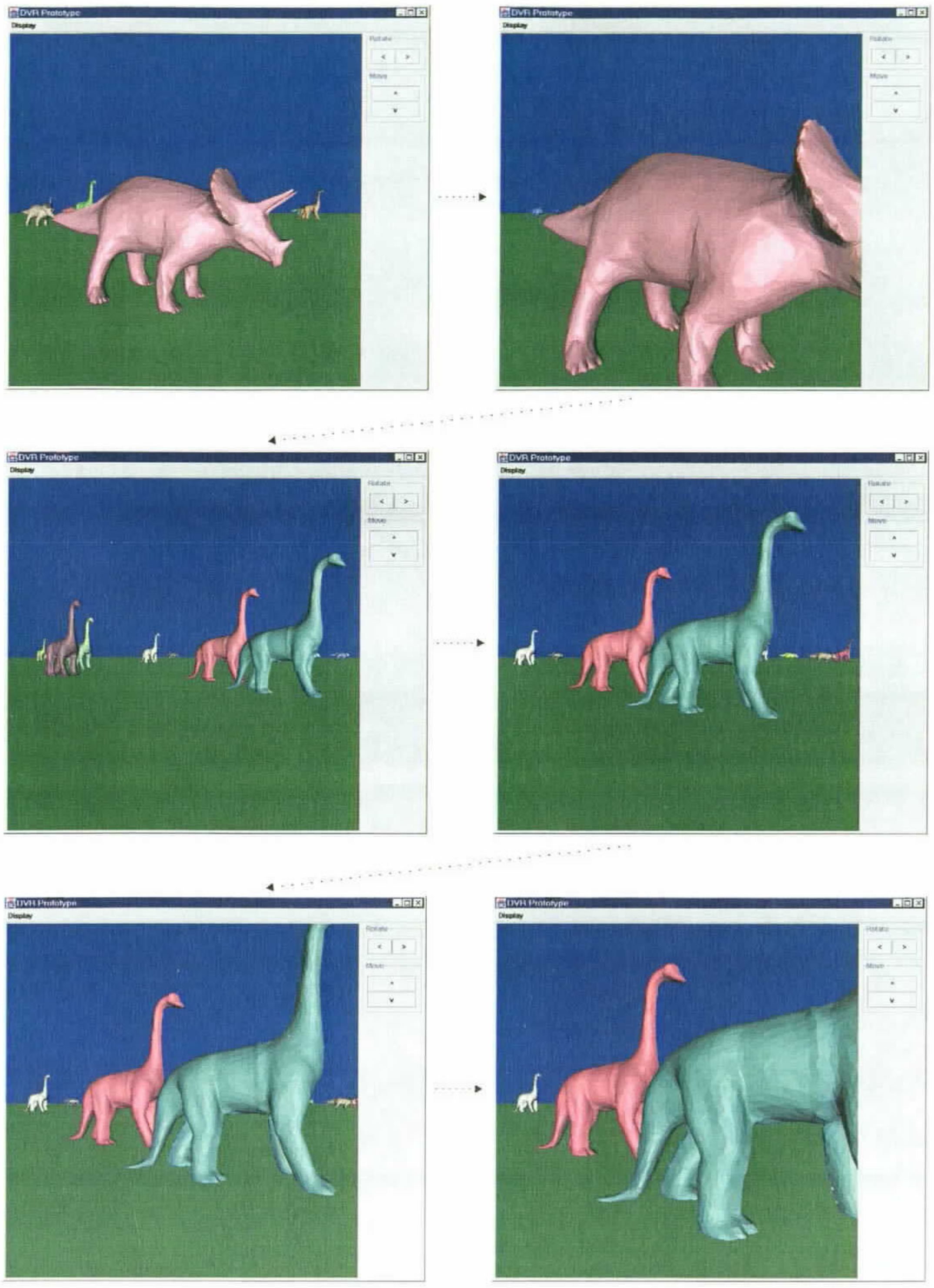


Figure 6.12: Demonstration of the second prototype implementation in shaded rendering.

Chapter 7

Conclusion

7.1 Contribution of this Thesis

In this thesis, we aim at developing a DVE system across the Internet. To overcome the problem of large network bandwidth requirement of the system, we have proposed the use of a *multi-resolution caching mechanism* and a *user-profiling based prefetching mechanism* to effectively cache and prefetch virtual objects of a VE at client machines. We have employed *multi-resolution modeling* for progressive transmission of virtual objects across the network and have defined the optimal resolution for rendering individual objects within a VE with respect to the viewer. We have proposed *object scope* and *viewer scope* for reducing the amount of data to be processed by client machines.

A realistic VE often consists of a large number of detailed geometric objects and results in a tremendous VE database containing gigabytes of data. As the Internet is notorious for limited bandwidth and instability, it takes an extremely long period of time for transmitting the content of a VE across the Internet. Caching data at the local storage of a client machine is commonly used to reduce the overhead in transmitting data across the network. With the constraint of limited storage space for caching, a replacement policy is needed for retaining valuable objects for possible future use. We have proposed the MRM replacement policy which exploits spatial locality among the cached items for effective cache management. We have demonstrated that MRM out-performs the conventional LRU replacement policy in the context of DVE across the Internet.

Caching is commonly complemented with prefetching for improving the performance of a caching mechanism if the prefetching is performed intelligently. We have proposed the use of a *user-profiling based prefetching mechanism* for improved prefetching accuracy and for better caching performance. A separate profile capturing the characteristics of individual viewer's walk pattern is maintained at the client machine. By predicting the future movement of the viewer based on the profile, it is possible to download data in advance and fully utilize the otherwise idle network.

In order to minimize the network bandwidth consumption, we have employed the *multi-resolution modeling* which allows progressive transmission of objects across the network. We have defined the *optimal resolution* as the best resolution for rendering individual objects within the VE. It is defined as a function of the size, distance and angle of an object with respect to a viewer. We have also proposed the concept of *object scope* and *viewer scope* for identifying a portion of the VE to be viewed by the user. This can significantly eliminate a large portion of the VE and prevent too much data from being transmitted across the network.

We have evaluated and quantified the performance of our proposed caching and prefetching mechanisms through a detailed simulation study. We have conducted numerous experiments and presented the results of a representative set of simulated experiments. We found that our proposed mechanisms do improve the system by significantly reducing the response time and latency, thus demonstrating that the mechanisms are effective.

We have implemented a system prototype using Java for demonstrating the feasibility of our proposed mechanisms. The prototype system is fast and responsive. It is capable of navigating a remote VE and download the visible virtual objects from the database server on demand. We have conducted a set of prototype experiments and the experimental results show the suitability of the proposed mechanisms in supporting a DVE system across the Internet. Furthermore, the prototype experimental results can validate the simulation results.

7.2 Future Work

Several important issues related to supporting a DVE system across the Internet remain unexplored in the thesis. To complete this thesis, we will discuss a few such issues and outline suggestions for future research directions.

Dynamic environment: In this thesis, we focus on static environments in which the virtual objects will not move within the VE. A more realistic and interesting environment should contain dynamic objects as well. A dynamic object will change its position and/or orientation which may either be controlled by a human user or by a computer algorithm autonomously. It is obvious that the number of dynamic objects will be relatively small because many virtual objects, such as mountains and buildings are static. An extension of our caching and prefetching mechanisms is still necessary in order to support a VE with dynamic objects. This will make our work more sound and complete.

Texturing: By mapping texture images onto virtual objects, the geometric complexity of a VE can be reduced. Texturing, in addition, can make a VE more realistic and believable. *Wavelet* texturing function is the most appropriate method for use with multi-resolution modeling due to its progressive nature. Our caching mechanism can easily be adapted to support texturing.

View-dependent Refinement and Transmission: For simplicity, the multi-resolution modeling technique in our method only allows view-independent refinement and transmission of geometric objects. View-dependent refinement permits resolution modification of a portion of the object selectively based on the view point of the viewer. By transmitting and refining only the portion of objects which are visible to the user, the network bandwidth consumption can further be reduced.

Geometry Compression: Compression techniques, using delta encoding or quantization, are useful in reducing the network bandwidth consumption. Although this method will increase the workload at both the server and the client, it is still beneficial because the processing power of computers is increasing rapidly.

Multiple servers: Currently, there is only one server maintaining the VE database and servicing requests from clients. In order to support a larger VE and more clients, it is possible to employ multiple servers maintaining the VE. A load-balancing algorithm is needed to even out the workload of individual servers. It is also necessary to handle the data consistency issue in updating the replicated data.

Bibliography

[1] S. Acharya, M. Franklin, and S. Zdonik. Prefetching from a broadcast disk. In *Proc. of the Int'l Conf. on Data Engineering*, pages 276–285, 1996.

[2] J. Ahn, M. Lee, and H. Lee. Doovie: An architecture for networked virtual environment systems. In *Proceedings of Computer Graphics International*, pages 113–119, 1997.

[3] B. Blau, C.E. Hughes, J.M. Moshell, and C. Lisle. Networked virtual environments. In *Proceedings of SIGGRAPH Symposium on Interactive 3D Graphics*, pages 157–160, 1992.

[4] W. Bricken and G. CoCo. The veos project. *Presence: Teleoperators and Virtual Environments*, 3(2):111–129, 1994.

[5] J. Calvin, A. Dicken, B. Gaines, P. Metzger, D. Miller, and D. Owen. The SIMNET Virtual World Architecture. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 450–455, 1993.

[6] M. Carey, M. Franklin, M. Livny, and E. Shekita. Data Caching Tradeoffs in Client-Server DBMS Architectures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 357–366, 1991.

[7] C. Carlsson and O. Hagsand. DIVE - a Multi-User Virtual Reality System. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 394–400, 1993.

[8] B. Y. L. Chan, H. V. Leong, A. Si, and K. F. Wong. MODEC: a multi-granularity mobile object-oriented database caching mechanism, prototype and performance. *In Journal of Distributed and Parallel Database*, 7(3):343–372, July 1999.

[9] S. Chen. Quick time vr - an image-based approach to virtual environment navigation. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, pages 29–38, 1995.

[10] J. Chim, M. Green, R. Lau, H.V. Leong, and A. Si. On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments. In *Proceedings of ACM Multimedia*, September 1998.

[11] J. Chim, R. Lau, H.V. Leong, and A. Si. Multi-resolution Cache Management in Digital Virtual Library. In *Proceedings of the IEEE Advances in Digital Libraries Conference*, pages 66–75, April 1998.

[12] J. Chim, R. Lau, A. Si, H.V. Leong, D. To, M. Green, and M.L. Lam. Multi-resolution model transmission in distributed virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 25–34, November 1998.

[13] J. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of ACM*, 19(10):547–554, 1976.

[14] F. Crow. A More Flexible Image Generation Environment. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, pages 9–18, July 1982.

[15] K.M. Curewitz, P. Krishnan, and J.S. Vitter. Practical prefetching via data compression. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 257–266, 1993.

[16] S. Dar, M.J. Franklin, B.T. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc. of the 22nd VLDB Conf.*, pages 330–341, 1996.

[17] M. DeHaemer and M. Zyda. Simplification of Objects Rendered by Polygonal Approximations. *Computers & Graphics*, **15**(2):175–184, 1991.

[18] D. DeWitt and D. Maier. A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems. In *Proceedings of International Conference on Very Large Databases*, pages 107–121, 1990.

[19] M. Eck, T. DeRose, T. Dunchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, pages 173–182, 1995.

[20] W. Effelsberg and T. Haerder. Principles of Database Buffer Management. *ACM Transactions on Database Systems*, pages 560–595, December 1984.

[21] J. Falby, M. Zyda, D. Pratt, and R. Mackey. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers & Graphics*, **17**(1):65–69, 1993.

[22] M. Franklin, M. Carey, and M. Livny. Global Memory Management in Client-Server DBMS Architectures. In *Proceedings of International Conference on Very Large Databases*, pages 596–609, 1992.

[23] M.J. Franklin. *Client Data Caching*. Kluwer Academic Publishers, 1996.

[24] T.A. Funkhouser. Ring: A client-server system for multi-user virtual environments. In *1995 SIGGRAPH Symposium on Interactive 3D Graphics*, pages 85–92, 1995.

[25] T.A. Funkhouser. Network toplogies for scalable multi-user virtual environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 222–229, 1999.

[26] T.A. Funkhouser, C.H. Sequin, and S.J. Teller. Management of large amounts of data in interactive building walkthoughs. In *Proceedings of SIGGRAPH Symposium on Interactive 3D Graphics*, pages 11–20, 1992.

[27] T.A. Furness and W Barfield. *Introduction to Virtual Environments and Advanced Interface Design*, chapter 1. Oxford University Press Inc, 1995.

[28] C. Gray and D. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proc. Int'l Symp. Operating System Principles*, pages 202–210, 1989.

[29] C. Greenhalgh and S. Benford. Massive: a distributed virtual reality system incorporating spatial trading. In *15th Int'l Conference on Distributed Computing System*, pages 27–34, 1995.

[30] O. Hagsand. Interactive multiuser ves in the dive system. *IEEE Multimedia*, 3(1):30–39, 1996.

[31] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel based object simplification. In *Proceedings of IEEE Visualization*, pages 296–303, 1995.

[32] P. Hinker and C. Hansen. Geometric optimization. In *Proceedings of IEEE Visualization*, pages 189–195, October 1993.

[33] H. Hoppe. Progressive Meshes. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, pages 99–108, August 1996.

[34] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh Optimization. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, volume **27**, pages 19–26, August 1993.

[35] V. İşler, R.W.H. Lau, and M. Green. Real-Time Multi-Resolution Modeling for Complex Virtual Environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 11–20, July 1996.

[36] IEEE1278, Mar. 1993.

[37] R. Jacob. Chapter 7: Eye tracking in advanced interface design. In *Virtual Environments and Advanced Interface Design, W. Barfield and T. Furness (Eds.)*, pages 258–288. Oxford University Press, 1995.

[38] A.D. Kalvin and R.H. Taylor. Superfaces: Polygonal mesh simplification with bounded errors. *IEEE Computer Graphics and Applications*, 16(3):64–77, 1996.

[39] R. Kazman. Making waves: On the design architectures for low-end distributed virtual environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 443–449, 1993.

[40] J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. In *Proc. Int'l Symp. Operating System Principles*, pages 213–225, 1991.

[41] D.F. Kotz and C.S. Ellis. Practical prefetching techniques for parallel file systems. In *Proc. of First Parallel and Distributed Information Systems*, pages 182–189, 1991.

[42] R.W.H. Lau, M. Green, D. To, and J. Wong. Real-Time Continuous Multi-Resolution Method for Models of Arbitrary Topology. *Presence: Teleoperators and Virtual Environments*, pages 22–35, February 1998.

[43] R.W.H. Lau, D. To, and M. Green. An Adaptive Multi-Resolution Modeling Technique Based on Viewing and Animation Parameters. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 20–27, 1997.

[44] H.V. Leong and A. Si. A database caching over the air-storage. *The Computer Journal*, 40(7):401–415, 1997.

[45] E. Levy and A. Silbershatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys*, 22(4):321–374, Dec. 1990.

[46] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, Nov. 1989.

[47] C. Liu and P. Cao. Maintaining strong cache consistency in the World-Wide Web. In *Proc. Int'l Conf. Distributed Computing Systems*, pages 12–21, 1997.

[48] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, pages 163–169, 1987.

[49] K. Low and T Tan. Model simplication using vertex-clustering. In *Proceedings of SIGGRAPH Symposium on Interactive 3D Graphics*, pages 75–82, April 1997.

[50] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygon environments. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, pages 199–208, August 1997.

[51] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz. NPSNET: A Network Software Architecture for Large-Scale Virtual Environments. *Presence: Teleoperators and Virtual Environments*, **3**(4):265–287, 1994.

[52] M. Macedonia, M. Zyda, D. Pratt, P. Brutzman, and P. Barham. Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 2–10, March 1995.

[53] P. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of SIGGRAPH Symposium on Interactive 3D Graphics*, pages 95–102, 1995.

[54] S. Makridakis, S.C. Wheelwright, and V.E. McGee. *Forecasting Methods and Applications.* John Wiley & Sons, 1983.

[55] D.C. Miller and J.A. Thorpe. Simnet: the advent of simulator networking. *Proceedings of the IEEE*, 83(8):1114–1123, 1995.

[56] C. Min, M. Chen, and N. Roussopoulos. The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching. In *Proceedings of International Conference on Extending Database Technology*, pages 323–336, 1994.

[57] D. Musser and A. Saini. *STL Tutorial and Reference Guide.* Addison-Wesley, 1996.

[58] B. Nitzberg and V. Lo. Distributed shared memory: A survey of issues and algorithms. *IEEE Computer*, 24(8):52–60, Aug. 1991.

[59] T. Ohshima, H. Yamamoto, and H. Tamura. Gaze-Directed Adaptive Rendering for Interacting with Virtual Space. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 103–110, July 1996.

[60] M. Palmer and S. Zdonik. Fido: A cache that learns to fetch. In *Proc. of Int'l Conf. on Very Large Database*, pages 255–264, 1990.

[61] M. Reddy and G.P. Fletcher. An adaptive mechanism for web browser cache management. *IEEE Internet Computing*, 2(1):78–81, 1998.

[62] J. Rossignac and J. Borrel. Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, June 1993.

[63] K. Salem. Adaptive prefetching for disk buffers. Technical Report TR-91-46, The Center of Excellence in Space Data and Information Sciences, 1990.

[64] G. Schaufler. Exploiting frame to frame coherence in a virtual reality system. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 95–102, 1996.

[65] G. Schaufler and W. Sturzlinger. A three-dimensional image cache for virtual reality. In *Proceedings of EUROGRAPHICS*, pages 227–236, 1996.

[66] D. Schmalstieg and M. Gervautz. Demand-Driven Geometry Transmission for Distributed Virtual Environments. In *Proceedings of Eurographics '96*, pages 421–432, 1996.

[67] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of Triangle Meshes. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, volume **26**, pages 65–70, July 1992.

[68] H. Schwetman. *CSIM Reference Manual (Revision 15).* Microelectronics and Computer Technology Corportaion, 1991.

[69] J. Shade, D. Lischiski, D. Salesin, T. Derose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, pages 75–82, 96.

[70] C. Shaw and M. Green. The mr toolkit peers package and experiment. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 463–469, 1993.

[71] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled simulation in virtual reality with the mr toolkit. *ACM Transactions on Information systems*, 11(3):287–317, July 1993.

[72] A. Si and H. V. Leong. Adaptive Caching and Refreshing in Mobile Databases. *Personal Technologies*, 1(3):156–170, September 1997.

[73] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts.* McGraw-Hill, 1996.

[74] A.P. Sistla, O. Wolfson, and Y. Huang. Miniziation of communication cost through caching in mobile environments. *IEEE Trans. on Parallel and Distributed Systems*, 9(4):378–390, Apr. 1998.

[75] D. Snowdon and A. West. Aviary: Design issues for future larg-scale virtual vironemnts. *Presence: Teleoperators and Virtual Environments*, 3(4):288–308, 1994.

[76] G. Sungh, L. Serra, W. Png, and H. Ng. BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence: Teleoperators and Virtual Environments*, 3(1):19–34, 1994.

[77] J. Torborg and J. Kajiya. Talisman: Commondity realtime 3d graphics for the pc". In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, pages 353–364, 1996.

[78] G. Turk. Re-tiling Polygonal Surfaces. In *Proceedings of ACM Computer Graphics (SIGGRAPH)*, volume **26**, pages 55–64, July 1992.

[79] B. Watson, N. Walker, and L. Hodges. Effectiveness of Spatial Level of Detail Degradation in the Periphery of Head-Mounted Displays. In *ACM CHI'96*, pages 227–228, April 1996.

[80] J. Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor.* Addision-Wesley, 1994.