

## Multi-Resolution Model Transmission in Distributed Virtual Environments

Jimmy H.P. Chim<sup>†</sup>   Rynson W.H. Lau\*   Antonio Si<sup>‡</sup>   Hong Va Leong<sup>†</sup>  
 Danny To\*   Mark Green<sup>‡</sup>   Miu Ling Lam<sup>†</sup>

<sup>†</sup> Department of Computing, The Hong Kong Polytechnic University, Hong Kong

\* Department of Computer Science, City University of Hong Kong, Hong Kong

<sup>‡</sup> Sun Microsystems, 901 San Antonio Road, Palo Alto, CA 94303, U.S.A.

<sup>‡</sup> Department of Computer Science, University of Alberta, Edmonton, Alberta, Canada

## Abstract

Distributed virtual environments allow users at different geographical locations to share and interact within a common virtual environment via a local network or through the Internet. To deliver a good performance for such applications, we need to address several issues in different research disciplines. First, we must be able to model virtual objects effectively. The recently developed multi-resolution techniques for object modeling are of great value here, since they are capable of simplifying the object models and therefore reducing the time to render them. This may greatly reduce the demand for rendering performance on the client machines. Second, with the constraint of the limited bandwidth of the Internet, we need to reduce the response time by reducing the amount of data requested over the network. Caching of suitable object models of high affinity will reduce the amount of data requested over the network for a faster response time. Prefetching object models by predicting those which are likely to be used in the near future and downloading them in advance will lead to a similar improvement. Third, the Internet often suffers from disconnection. A caching mechanism that allows objects to be cached with at least their minimum resolutions will be useful to provide at least a coarse view of the objects to the viewer for improved visual perception. In this paper, we describe our implementation of a distributed walkthrough system. Two techniques are fundamental to our system, a *multi-resolution caching mechanism* and a set of *object prefetching mechanisms*. Towards the end of the paper, we quantify the performance of the proposed mechanisms.

## 1 Introduction

In a virtual walkthrough application, a user could explore a specific place of interest without having to travel physically. The place of interest is modeled as a virtual environment, containing a vast number of virtual objects. Sample applications of this sort include virtual museum, virtual library, virtual university, etc. [19]. Employing a standard client-server architecture, information of virtual objects, including

their locations, sizes, orientations, and shapes, will be maintained in a central database server. When a viewer (user) walks through a virtual environment, information of the virtual objects located within a visible distance from the viewer will be conveyed to the client machine of the viewer for rendering. As the viewer moves within the virtual environment, the relative locations and orientations of the objects may change with respect to the position of the viewer. Such changes of information should be reflected into the rendered images in a timely fashion. In general, the virtual objects could be dynamic as well, changing their locations and orientations within the virtual environment. However, in this paper, we only focus on virtual environments where objects are static. The goal is to provide a good performance of the application, both in terms of responsiveness and resolution, under the existing constraints of relatively low Internet bandwidth and the large memory demand of virtual objects.

We are addressing several issues in this application. First, virtual objects must be modeled in a compact form so as to reduce the amount of storage space needed and the amount of time required to transfer the objects from the server to a client. A compact modeling of virtual objects also has the benefit of fast retrieval from secondary storage, both at the server and at a client. However, over-compact modeling of virtual objects will increase the overhead in compressing and decompressing the objects. The recently developed multi-resolution methods for object modeling [13, 16] could be employed here. The techniques allow progressive transmission of objects with only minimal overheads.

Second, with the limited bandwidth of the Internet, we need to reduce the amount of data requested over the network for faster response time. This can be achieved by caching and prefetching mechanisms. A caching mechanism allows a client to utilize its memory and local storage to cache currently visible objects that are likely to be visible in the near future [11]. A prefetching mechanism allows a client to predict objects that will be visible in the future and obtain the objects in advance to improve response time. A good caching mechanism should retain objects with high affinity while a good prefetching mechanism should predict those objects which will most likely be used.

Third, the Internet often suffers from various degrees of disconnection. The local storage cache of a client can be used to provide partial information to support a certain degree of disconnected operation. For example, a viewer may still be able to see a coarse resolution of objects in the virtual environment if the minimal approximated models of the objects are cached. Even if only the coordinates of the virtual objects are cached, a viewer could still be aware of the

existence of the objects.

In this paper, we describe our implementation of a distributed walkthrough system. Two techniques are fundamental to our system, a caching technique called the *multi-resolution caching mechanism* and a prefetching technique composing of a set of *object prefetching mechanisms*. The rest of the paper is organized as follows. Section 2 presents a survey on relevant research. Section 3 presents the multi-resolution modeling technique, and Section 4 presents the multi-resolution caching and prefetching mechanisms. In Section 5, we discuss the implementation of our experimental prototype system. We quantify the performance of our caching and prefetching mechanisms with several experiments via simulation as well as on the prototype in Section 6. Finally, we conclude our paper with a discussion on possible future work.

## 2 Related Work

Several approaches have been proposed to distribute data from the server to the clients in distributed virtual reality applications. Most existing systems, such as DIVE [3], SIMNET [1], and VLNET [22], use a complete replication approach to distribute all geometry data to the clients before the start of the application. This approach assumes the use of a reliable high speed network and is therefore not suitable for use in the Internet environment. If the geometry database is large, distributing it via the Internet may result in a high pre-loading time. Another approach to distribute geometry data is to send them on demand to the clients [10, 23, 27] during the application. This approach employs a standard client-server architecture, in which a central server maintains a geometry database of the virtual environment and distributes data to clients when requested. Client-server architecture is very common in the Internet.

In a client-server database environment, to combat the network transmission latency, a multi-level caching mechanism could be established by caching database objects from the server in a client's local memory and/or local storage. A storage cache has an advantage of persistence. When disconnected from the server, a client can still operate on the database objects in its local storage.

In a conventional client-server database environment, data objects are usually transferred from the database server to a client on a per-page basis [2, 11]. This is primarily because the server's storage is also page-based. The overhead for transmitting one item or a page is similar. In a virtual walkthrough environment, virtual objects are represented using *object models* and are usually very complex and large in size, occupying possibly multiple pages. The overhead required to transfer an object model (or simply object) in its entirety via the narrow bandwidth Internet is very high. Furthermore, we might not always need to render an object at its full resolution (see Section 4). Hence, there will be situations that we need to transfer less than a page of information and there also exist situations that we need to transfer more than a page of information. A more dynamic granularity for caching is therefore needed in a virtual walkthrough environment.

If a client can provide unbounded disk storage and wait for a possibly very long pre-loading time, we could transmit all virtual objects in the environment to the client before starting the walkthrough, as in the complete replication approach [1, 3, 22]. However, a more realistic situation is that the available storage for caching and the available pre-loading time are limited [20]. A cache replacement policy

must be employed to retain only frequently accessed objects. Finally, to reduce the access and rendering latency, it is also beneficial to prefetch potentially visible objects in the client, if disk space is available.

### 2.1 Multi-Resolution Modeling

In a virtual walkthrough application, rendering a complex object at a client is expensive. From the perspective of a viewer in the virtual environment, distant objects appear smaller than nearby objects after projection. Most of the details of distant objects are actually not visible to the viewer. Hence, it is only necessary to represent an object at the resolution just high enough for the given viewing distance. This could reduce not only the rendering time, but also the transmission delay and the storage required at a client to hold the objects. Here, we employ multi-resolution modeling techniques to study the effect of caching objects in a client at various granularities. In brief, our multi-resolution mechanism caches and prefetches a nearby object at a higher resolution and a distant object at a lower one.

There are many methods developed for generating multi-resolution models [8, 14, 24]. However, most of them focus on the accuracy of the simplification, and hence, are slow. A popular method to overcome the performance limitation is called the *discrete multi-resolution method*. This method pre-generates a few key models of an object at different resolutions. During run-time, the object's distance from the viewer determines which model to use for rendering [7]. Although this method is fast and simple, it has one major limitation in our context. Since all the key models are independent of each other, the overall amount of information needed to represent a particular object is increased and is dependent on the total number of key models used. In a distributed environment, this will increase the network traffic and hence reduce the availability of objects.

A method referred to as *progressive meshes* was recently proposed for progressive transmission of multi-resolution object models [13]. The method is based on two operators, *edge collapse* for reducing model resolution, and *edge split* for increasing model resolution. Each object is modeled as an ordered list. The list begins with the minimal resolution model of the object, referred to as the *base mesh*. Each subsequent record in the list, referred to as the *progressive record*, stores information of an edge split. If we apply the information stored in each of the records to the object in order, the object will gradually increase in resolution until it reaches the maximum resolution, when all the records in the list are exhausted. Conversely, the method may begin with the highest resolution model of the object; if we apply the information stored in each of the records in reverse order, which is equivalent to an edge collapse operation, the object will gradually decrease in resolution until it reaches the minimum resolution. We have recently developed a similar method [15, 16].

### 2.2 Replacement and Prefetching Techniques

In [9], various cache replacement policies have been proposed and their suitabilities in a conventional database system have been examined. These policies are all page-based, due to the logical mapping made by the database or operating system to the physical storage. In general, the performance of individual replacement policies is sensitive to the characteristics of queries initiated and the application environment. A general conclusion on the performance of the replacement policies cannot be made. In practice, the

replacement policy is often approximated by the *Least Recently Used* (LRU) policy in conventional caching [2, 11, 26]. In [25], it was shown that LRU policy is not appropriate in a context where the objects accessed by a client might change over time. Rather, the semantics of data access is more important in defining the replacement policy. We, therefore, need to develop a more appropriate replacement policy based on the semantics of accesses in a walkthrough environment. It was also noticed in [4] that prefetching could be very beneficial in improving the performance of database applications if the prefetching is performed intelligently.

### 3 Multi-Resolution Modeling Technique

#### 3.1 Object Scope

In our method, 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  $o$  is actually reflecting a count of the total number of edges which can be collapsed from its maximum resolution. Each object will also have a base mesh at its minimum resolution,  $\mathcal{R}_o$ . We say that the base mesh  $\mathcal{R}_o$  of  $o$  represents  $o$  at resolution level 0, denoted as  $\hat{\mathcal{L}}_o$ , which has a value of 0. Each progressive record will increase the resolution level by 1. Therefore, the maximum resolution  $\hat{\mathcal{R}}_o$  represents  $o$  at the highest resolution level, denoted as  $\hat{\mathcal{L}}_o$ , when all progressive records are applied.

To minimize the amount of data needed to be handled, most existing methods consider only the *area of interest* (AOI) of the viewer [10, 18, 23]. 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 to be visible. Although these methods can quickly eliminate invisible objects, they do not consider the sizes of the objects. Hence, a mountain located just outside the AOI of the viewer may still be visible to the viewer, but is considered as invisible, while a tiny object such as a book located just inside the AOI of the viewer is unlikely to be visible to the viewer, but is considered for visibility. 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*. We denote the viewer scope for viewer,  $V$ , by  $\mathcal{O}_V$  and the object scope for object,  $o$ , by  $\mathcal{O}_o$ . A viewer scope is similar to AOI. It indicates the depth of sight of the viewer, i.e., how far the viewer can see. A viewer with a 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. An object scope indicates how far the object can be seen. A large object has a larger scope and a small object has a smaller scope. An object may be visible to a particular viewer only when its scope overlaps with the viewer's scope. When the two scopes overlap, the distance between the object and the viewer and the angle of the object from the viewer's viewing direction can be used to determine the optimal resolution of the object. Obviously, a viewer may also be considered as an object and assigned with 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 [12].

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.,  $\mathcal{O}_o$ , as  $r_{\mathcal{O}_o}$ , while the radius of the viewer scope for viewer  $V$ , i.e.,  $\mathcal{O}_V$ , as  $r_{\mathcal{O}_V}$ .

The interaction between a viewer and the virtual environment is illustrated in Figure 1. In addition to the viewer scope, each viewer,  $V$ , is also associated with 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 considered as visible to the viewer. Even though some objects may be located at the back of the viewer, the viewer may be able to see them within a very short time simply by turning around. As will be described later, these objects have the highest priority to be cached in the client's memory and we refer them as *cacheable objects*. Each viewer also has a viewing angle. This viewing angle defines the viewer's viewing region, which is a sub-space of the viewer scope. All cacheable objects within the viewing region are considered for rendering, and we refer them as *renderable objects*. Each renderable object will be rendered at its *optimal resolution*. 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 distance from the viewer and the angular distance of the object from the viewer's viewing direction, i.e., line of sight. If the object is rendered at a resolution higher than this optimal resolution, the additional details will not be easily 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 [5]. Such a perceived image quality is called the *visual perception*.

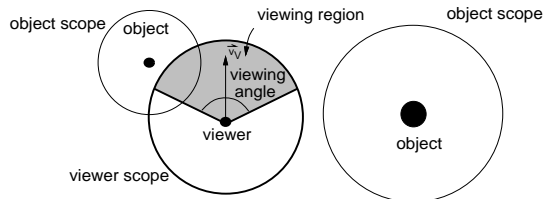


Figure 1: Objects-viewer interaction in a virtual environment.

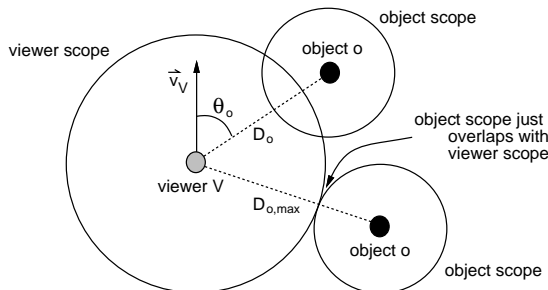
#### 3.2 The Optimal Resolution of an Object Model

The optimal resolution of an object model can be determined according to the *visual importance* of the object to a viewer. In [17], we have identified several factors that may affect the visual importance of an object. Here, we only consider two of those factors, which are relevant to the context here. 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 [21, 28]. Degradation of peripheral visual detail can improve rendering performance and reduce perceptual impact. Here, we assume that the viewer's line of sight is at the center of the screen.

Figure 2 depicts the visual importance of an object,  $o$ , to a viewer,  $V$ . In the figure,  $D_{o,V}$  indicates the current distance of the object from the viewer, while  $D_{o,V,max}$  is the distance between the object and the viewer when their scopes just overlap. Hereafter, we will consider in the context of viewer  $V$  and the subscript  $V$  can be dropped if the context is clear. 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  $\theta_o$  ( $-\pi \leq \theta_o \leq \pi$ ). The visual importance of  $o$  to a viewer can be defined with the following formula:

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

where  $K_o$  is a constant for adjusting the decrement rate of object  $o$  due to the increase in  $\theta_o$ . In our implementation, we do not want the line of sight factor to dominate the distance factor. Hence, we use a small value of  $K_o$ .



**Figure 2:** Visual importance of an object to a viewer in a virtual environment.

To incorporate this idea into the progressive multi-resolution method described in [13], the object importance,  $I_o$ , is used to determine the optimal resolution of the object model. During the walkthrough, we continuously determine those cachable objects. When the scope of an object,  $o$ , touches the perimeter of the viewer scope, the optimal resolution level of the object will be equal to its base mesh, which provides the minimal resolution of the object. As the object moves closer to the viewer or to the viewer’s line of sight, its optimal resolution increases. The object model of each cachable object at its optimal resolution will be transmitted to the client if it is not already cached in the local storage. When transmitting the models from the server to a client, models of objects within the viewing region are transmitted first, followed by the transmission of those outside the viewing region. 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 some object models that are not likely accessed in the near future in order to accommodate the new models, i.e., we try to decrease the resolution of some existing cached objects (see Section 4).

We also attempt to further improve the performance of the walkthrough application by having the server prefetch objects which will most probably be accessed in the future to the client. This is achieved by having the server predict the next location and viewing direction of the viewer based on his/her past movement profile. Models of objects whose scopes overlap with the viewer scope at the predicted loca-

tion will be transmitted at their optimal resolutions to the client as well (see Section 4).

Our method has several advantages over previous approaches. First, in [23], discrete multi-resolution method is used for model transmission. Redundant information will have to be sent through the network, since 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. Second, 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 resolution of the viewing device. Third, our caching mechanism differs from conventional caching mechanisms [2, 11, 26] in that objects could be cached at multiple degree of granularity. Replacement is also based on object access patterns rather than conventional LRU policy. Finally, the performance of the walkthrough application is further improved by predicting the future movement of the viewer and prefetching objects in advance.

## 4 Multi-Resolution Caching Mechanism

Multi-resolution modeling allows the database server to transmit an object model at the optimal resolution for rendering. This could save the scarce Internet bandwidth from transmitting details of an object too small to be visible to the viewer. To further reduce the dependency on the Internet to reduce transmission delay and to support disconnected operation, 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.

### 4.1 The Cache Model

When a viewer,  $V$ , moves within the virtual environment, the client machine,  $C$ , will transmit the current viewing direction,  $\vec{v}_V$ , and the current location,  $loc_V$ , of  $V$  to the server. This is a query to the database server for all cachable objects. Concurrently,  $C$  will identify the cachable objects among the cached objects stored in its local storage cache based on the list supplied by the server.

Each cached object,  $o$ , is associated with a resolution,  $\mathcal{R}_o^*$ , indicating the current highest possible resolution of the model available for rendering. This resolution level depends on the number of progressive records,  $\mathcal{L}_o^* \leq \hat{\mathcal{L}}_o$ , cached in  $C$ .  $C$  would then submit an *existent list* to the server, i.e., a list of  $\langle o, \mathcal{L}_o^* \rangle$  pairs about those cachable objects cached in  $C$ ’s storage. Cachable 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. On the other hand, those cachable objects not cached in  $C$  or those not at the required optimal resolution  $\mathcal{L}_o$  will have to be transmitted to  $C$ . A *result list* in the form of  $\langle o, \text{progressive mesh} \rangle$  pair is transmitted. Such a progressive mesh only contains enough progressive records to define the optimal resolution of the object.

Upon receiving the result list from the server,  $C$  might 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 object has a score higher than the lowest score of some currently cached objects, it will be cached.

## 4.2 Multi-Resolution Replacement Mechanism

We employ the *Most Required Movement* (MRM) replacement technique in defining the access score for each object. It is based on the observation that the farther an object is from the viewer, the lower the resolution it can be rendered since the longer it will take for the viewer to move to view the object in greater detail. Consequently, its value of being cached in the storage is lower. Similarly, the larger the angle between an object and the viewer’s line of sight is, the lower the required resolution is since the longer it will take for a client to rotate to view the object directly in front. Its value of being cached in the storage is also lower. Preliminary experiments have shown that such a replacement scheme outperforms traditional LRU replacement scheme [6]. In this paper, we investigate the effectiveness of MRM using a real prototype, taking into account of the viewer and object scope information.

There can be different formula to calculate the access score,  $S_{o,v}$ , for an object,  $o$ , with respect to a viewer,  $V$ . Since we would like to have as few parameters to be adjusted as possible, we have developed the following formula with only a single adjustable parameter  $\omega$  ( $0 \leq \omega \leq 1$ ). Using the notations in Figure 2,  $S_{o,v}$  is defined as:

$$S_{o,v} = \omega \left( 1 - \frac{D_o - r_{O_o}}{r_{O_v}} \right) + (1 - \omega) \left( 1 - \frac{|\theta_o|}{\pi} \right).$$

When the object with the lowest access score is selected for replacement, we will not remove the whole object from the storage cache immediately. Rather, its extra resolution detail will be reduced to its optimal resolution by removing all the extra progressive records. This will make room for the incoming objects. If there is still not enough room to accommodate the new objects, the object with the next lowest access score will be 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, all progressive records of the object with the lowest access score will be removed from the storage cache, leaving only the base mesh of the object at its minimum resolution. Again, this process will be iterated. Finally, the base mesh of the object with the lowest access score will be removed if there is still not enough room. This process will be iterated until enough room is allocated for all new objects.

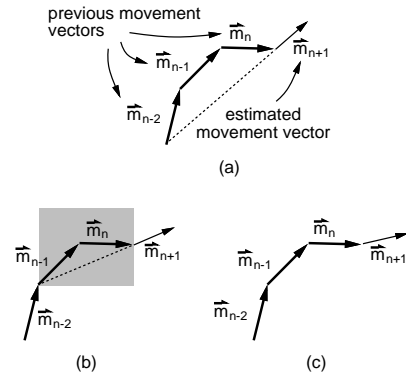
This multi-resolution replacement scheme tries its best to keep a coarse resolution of an object in a client’s storage. This provides a viewer with a much better visual perception since all or most of the cachable objects could be seen instantaneously, even though they may only be at a low resolution.

## 4.3 Prefetching Mechanism

To enable prefetching, the server maintains a separate profile for each viewer  $V$ , containing the set of historical *movement vectors*,  $\{\vec{m}_1, \vec{m}_2, \dots, \vec{m}_{n-1}\}$ . Each vector is calculated from the corresponding viewer’s location and orientation, containing a moving direction and a moving distance. When  $V$  moves to a new location,  $loc_n$ , with a new orientation, the  $n^{th}$  movement vector,  $\vec{v}_n$ , is calculated. The server attempts to predict the  $n+1^{th}$  movement vector,  $\vec{m}_{n+1}$ , of  $V$  and transmits objects that would be cachable if  $V$  were at  $loc_{n+1}$ , in addition to the cachable objects at  $loc_n$ . This

would save future requests to the server if the prefetched objects are indeed required by the client.

We propose three different schemes to predict the next location of the viewer: mean, window, and exponential weighted moving average (EWMA). The semantics of these schemes are depicted in Figure 3.



**Figure 3:** Prediction of 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 the previous  $n$  movement vectors, as depicted in Figure 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 3(b), showing a window of size  $W = 2$ .

A problem for the window scheme is the amount of storage needed in maintaining the movement vectors within the windows. To avoid the need of a moving window, and to adapt quickly to changes in viewer moving patterns, our EWMA scheme assigns a weight to each previous movement vector so that recent vectors have higher weights and the weights tail off as the vectors become aged. A parameter is the exponentially decreasing weight,  $\alpha$ . The most recent vector will receive a weight of 1; the previous vector will receive a weight of  $\alpha$ ; the next previous one will receive a weight of  $\alpha^2$ , and so on. This idea is depicted in Figure 3(c), indicating the predicted moving direction.

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 [25]. 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 predication is  $\vec{e}_n = \hat{m}_{n+1} - \vec{m}_n$ . We consider the angle between  $\hat{m}_{n+1}$  and  $\vec{m}_n$ , denoted as  $\phi_n = \arg(\hat{m}_{n+1}) - \arg(\vec{m}_n)$ ,

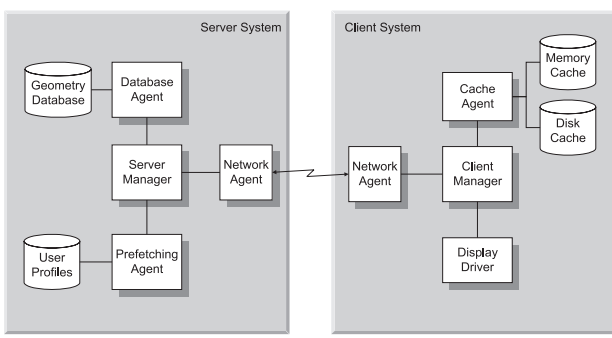


Figure 4: Architecture of the prototype.

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  $e^{-i\phi_n}$ . Since we do not really know  $\phi_{n+1}$  when we predict  $\hat{m}_{n+1}$ , we must try to predict  $\phi_{n+1}$  as well. There can be different ways of predicting  $\phi_{n+1}$  from the previous values of  $\phi_i$ , namely, mean, window and EWMA. Again, we propose to use EWMA to compute the prediction of  $\phi_i$  at each step as we compute  $\hat{e}_i$ . Thus,  $\hat{\phi}_{n+1} = \alpha\hat{\phi}_n + (1 - \alpha)\phi_n$ , and  $\vec{m}_{n+1} = \hat{m}_{n+1}e^{-i\hat{\phi}_{n+1}}$ .

## 5 Implementation

We have developed a prototype of the proposed distributed walkthrough system. The prototype is mainly implemented using Java, due to its platform independence nature, except for the Display Driver. The Display Driver is implemented using OpenInventor, which would utilize the underlying graphics capability of the client, if available, for better performance. This is also because when we started our implementation work of the system, Java3D was not available.

### 5.1 Architecture

As mentioned previously, our distributed walkthrough system is based on a client-server model. The architecture of our prototype is therefore divided into two main parts, the *Client System* and the *Server System*. The Client System consists of 4 main components as shown in Figure 4. The major functions of the components are as follows:

- *Client Manager*: It serves as the coordinator of all other components at the Client System. All viewer inputs, such as translation or rotation, are directed to and handled by the Client Manager.
- *Cache Agent*: It controls all the local caches, including the memory cache 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 will perform cache cleanup at the underlying cache(s) when necessary. The Cache Agent also maintains a *Score Table*, containing the access score (see Section 4) of each object in the local cache.
- *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.

- *Display Driver*: It accepts input from the viewer and generate output images to be displayed on the display screen.

The Server System also consists of 4 main components. Their major functions are as follows:

- *Server Manager*: It serves as the coordinator of all other components at the server side and handles all clients' requests.
- *Database Agent*: It maintains the database of the virtual environment, and sends object models in the form of progressive meshes or progressive records to a client, upon receiving the request for objects from the client. To reduce the overhead incurred for determining the set of cachable objects within a viewer scope, the Database Agent maintains a *Database Index Table*. The table is indexed based on the virtual coordinate and a radius, which identifies the set of cachable objects within the radius, centered at the coordinate.
- *Prefetching Agent*: It prefetches objects to clients based on some historical movement vectors of the client in the form of user profiles. These movement vectors are used to predict the next position of the client. A number of prefetching methods are implemented including mean, window, and EWMA.
- *Network Agent*: It handles all the communications between a client and the server, and maintains the connection between them once a connection has been established. It also handles all the object requests by the client.

Figure 5 illustrates sample sessions with the prototype. Figure 5a shows a scene in the virtual environment. Figure 5b illustrates the structure of the meshes for rendering. Note that objects farther away are represented by meshes with lower resolution. Figures 5c and 5d represent the scene and mesh structure when the viewer moves forward, with an increase in the resolution of the cow and the trees.

### 5.2 Client-Server Interaction

A connection-oriented protocol is used in the prototype. The connection between the client and the server would be maintained once it has been established to prevent the extra overhead incurred in further communications between the client and the server. For connectionless protocol such as HTTP, a connection between the client and server has to be established every time before any data transmission. This would increase the overhead and thus the response time to the viewer.

When a client wants to connect to a server, the client sends a **NEW** command, specifying the size of its scope and the viewing angle. If the server accepts the client's connection, the server replies with the assigned ClientID and other parameters of the virtual environment to the client. As the viewer moves or rotates within the virtual environment, the client sends an **ACK** command to the server, specifying the client's updated location and viewing direction. The server replies with an object ID list, specifying all cachable objects that are within the new scope of the client and the optimal resolution required for each object. If prefetching is used, the IDs of the predicted objects would also be included in the list. Upon receiving the object ID list, the client looks up its cache via the Cache Agent to identify the cached and

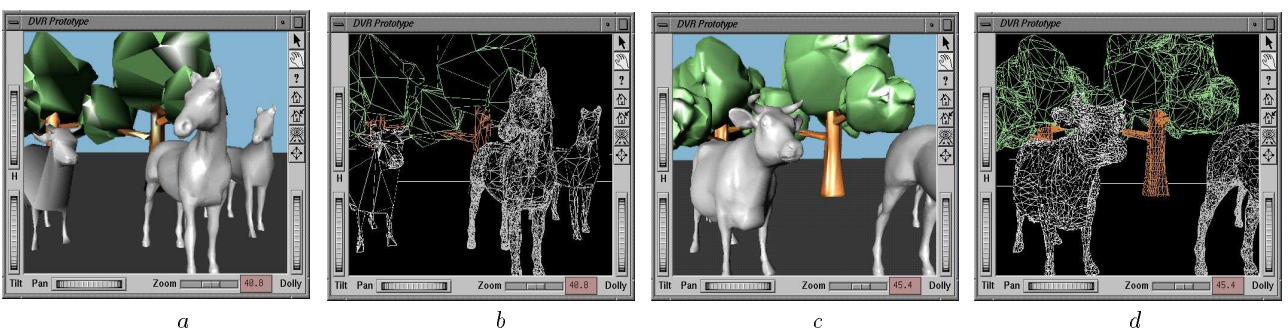


Figure 5: Sample sessions with prototype.

uncached information. An REQ command will be sent to the server, requesting progressive records and/or progressive meshes of the cachable objects from the server. When the client stops the walkthrough, a DONE command is sent to the server, closing the connection.

### 5.3 Progressive Mesh Transmission

To transmit a progressive mesh 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 a small amount. As an edge is inserted or split in a local mesh, vertices might need to be divided as well. This idea is illustrated in Figure 6. In Figure 6, as an edge identified by the two vertices,  $V_{parent}$  and  $V_{child}$ , is inserted into a local mesh, some of the immediate neighboring vertices of  $V_{parent}$  are divided as well. The new vertex  $V_{child}$  becomes an immediate neighboring vertex of  $V_{parent}$  while some of the former neighbors of  $V_{parent}$  become the immediate neighbors of  $V_{child}$ . To reduce the amount of information needed to be maintained in each progressive record for such a vertex division, a linked list for each vertex of an object model is maintained at the client. The linked list points to all immediate neighboring vertices for each vertex, ordered in a clockwise direction.

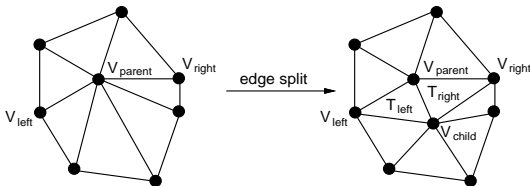


Figure 6: A progressive record stores information for an edge split.

The progressive record for the edge split operation shown in Figure 6 stores the  $(x, y, z)$  position of  $V_{child}$ , i.e., the vertex to be inserted, and the ID of the parent vertex, i.e.,  $V_{parent}$ .  $V_{parent}$  will then join with  $V_{child}$  to form the inserted edge. We also need to transmit the IDs of vertices,  $V_{left}$  and  $V_{right}$ . They help to identify the locations where the two new triangles are to be inserted to. The two triangles are defined as  $T_{left} = \langle V_{parent}, V_{child}, V_{left} \rangle$  and  $T_{right} = \langle V_{child}, V_{parent}, V_{right} \rangle$ . The two vertices also help to divide all the immediate neighboring vertices into two groups.

The data structure for our implementation of the progressive mesh is shown in Figure 7.

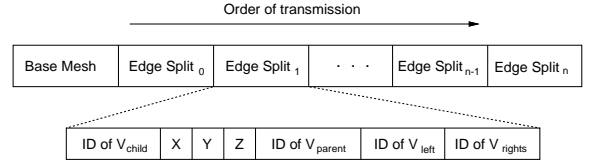


Figure 7: The data structure of the progressive mesh for transmission.

## 6 Results and Discussions

We have conducted some preliminary experiments to quantify the performance of MRM replacement scheme and the effectiveness of various prefetching schemes via simulation as well as on the prototype. The purpose of simulation is a proof of concept, allowing us to experiment the behavior of the mechanisms under diverse situations easily. The prototype provides a study under a real situation. We will first present a simulated experiment to illustrate the general behavior of the caching and prefetching schemes, followed by a more detail analysis of the performance of the mechanisms using our prototype.

We characterize the performance of the caching and replacement schemes with two metrics: *cache hit ratio* and *visual perception*. Cache hit ratio measures the percentage of bytes of the renderable objects, i.e., those within the viewing region, that could be retrieved from the local storage cache of the client. A high hit ratio is important to reduce reliance on network and to provide service during disconnection. Visual perception measures the relative degree (in percentage) of image quality experienced by a viewer just after the move. The visual perception of a cached renderable object,  $o$ , is modeled as a cubic function:  $1 - \left(\frac{B_o - B_o^*}{B_o}\right)^3$ , 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. This definition of visual perception is based on the fact that when a viewer makes a move in the virtual environment, s/he would experience a high visual perception if all renderable objects could be seen instantaneously (from cache), even at a coarse resolution. By contrast, a viewer would experience a low perception if s/he needs to wait for a long time before all renderable objects could be observed. A visual perception of 100% is assumed when the the cached model could provide the optimal resolution.

## 6.1 Experimental Environment

The same set of parameters are used in our experiments under simulation and prototype, as listed in Table 1. In our experimental environment, there are  $n$  virtual objects in the database server. We focus on a single server and a single client in our study here. The effect of the number of clients on the performance of caching and prefetching schemes will be reported in the future.

Notation	Description
$n$	Number of virtual objects
$N$	Size of storage cache (percentage of database)
$\omega$	Parameter for determining access score (fixed at 0.5)
$F_{disk}$	Prefetching scheme for storage cache
$W$	Window size
$\alpha$	Exponentially decreasing weight (fixed at 0.5)
$P$	Moving patterns of the viewer

Table 1: Parameters listing for experiments.

The virtual environment is modeled as a  $2000 \times 2000$  square units in size. The  $n$  virtual objects are distributed uniformly among the square units, each containing an average of  $\frac{n}{4000000}$  objects. The viewer is assumed to reside at the center of the viewer scope. The viewing angle is set to be 120 degrees, i.e.,  $\frac{2\pi}{3}$ . The radius of the viewer scope is set to 10 units. Only storage cache at the client is considered in our experiments. The cache size is equal to  $N\%$  of the database size. We experimented with different prefetching schemes, including no prefetching (No Prefetch), Mean, Window, and EWMA. No Prefetch forms a base case for comparison. We experiment with four window size,  $W$ , ranging from 1, 3, 5, and 7. We denote Window with window size,  $W$ , as Win- $W$ . We refer to EWMA with residual adjustment enabled by EWMA-R, and EWMA with residual adjustment disabled by EWMA-NR.

We experiment with three moving patterns,  $P$ , of a viewer, as depicted in Figure 8. Each pattern contains a sequence of movement steps. The first pattern models a constant circular translation pattern (CP). The viewer moves circularly starting and ending at the same location. Each movement step includes a translation of 150 units along the viewing direction, followed by rotating the viewing direction through an angle of  $\frac{\pi}{15}$ , i.e., 12 degrees. At every position, the viewer rotates by  $\pm\frac{\pi}{9}$ , i.e.,  $\pm 20$  degrees. This models a situation where a viewer explores the virtual objects around him/her for every movement. The second pattern, called changing circular pattern (CCP), models the same pattern as CP except that the moving direction changes with an angle of  $\frac{\pi}{36}$ , i.e., 10 degrees, after every 4 movement steps. Finally, in the random moving pattern (random walk or RW), each movement step is either a translation of arbitrary length or a rotation of arbitrary angle.

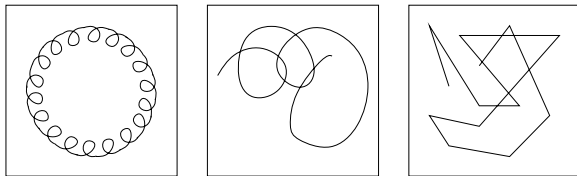


Figure 8: Moving patterns: (a) CP, (b) CCP, and (c) RW.

## 6.2 Experiments from Simulation

The purpose of our simulation is to study the performance of the caching mechanism, with and without prefetching, on various moving patterns. In our simulation model, there are 5000 virtual objects. Each object is modeled by a progressive mesh, containing a base mesh and a list of progressive records. The number of progressive records associated with each object model follows a normal distribution with a mean of 25,000 records and a standard deviation of 2,500 records. Each progressive record has a size of 40 bytes while each base mesh has a size of 2KB. The database is approximately 5GB and the size of the storage cache is fixed at 1% of the database.

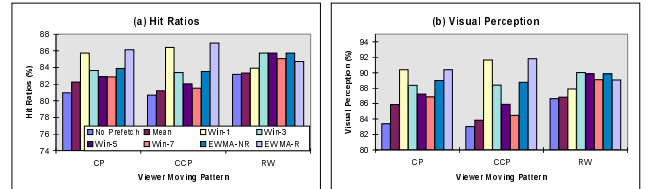


Figure 9: Performance from simulation.

The measurements of the metrics are depicted in Figure 9. We observe that even without prefetching, the caching mechanism performs reasonably well, achieving a hit ratio ranging from 79% to 83% (Figure 9a). With prefetching, the hit ratios could be improved by up to 6%. We observe that Mean is not very effective in predicting future movements, performing similar to the base case, i.e., No Prefetch. Both Window and EWMA perform equally well in improving the hit ratios of the caching mechanism.

With respect to Window, a small window size results in better performance under the CP and CCP moving patterns. Under CP and CCP, the moving direction is always changing, very often with a constant angle. With a large window size, aged moving vectors will contribute to the prediction of the moving vector, introducing some noise in the prediction. By contrast, under the RW moving pattern, each movement step bears a high degree of randomness. The small window does not capture enough information to predict the next movement vector. Therefore, the performance with a small window size is not as good as that with a large window size under RW.

EWMA exhibits a similar behavior. EWMA-R performs better under the CP and CCP moving patterns. This is mainly because the angle deviation under these two moving patterns exhibit a well defined pattern and is thus predictable. Under the RW moving pattern, the angle deviation does not exhibit a clear pattern and the residual correction does not seem to yield any improvement.

## 6.3 Experiments from Prototype

In our prototype experiments, the server runs on an Ultra-Sparc 2 station with 128MB RAM. The client runs on an SGI Indigo 2 station with 64MB RAM. We study the behavior of the caching and prefetching mechanisms under a real system and compare the relative performance against the simulation. Due to space limitation, we only present two representative sets of experiments in this paper.

### 6.3.1 Experiment #1

Our first set of experiments resembles our simulated experiment presented in Section 6.2. However, since running ex-



periments on a prototype is very time-consuming, we reduce the number of objects,  $n$ , to 500 here. The average size of each object is also reduced to 200KB. All other parameters remain the same. We hope to be able to compare the general behavior of the mechanism under a real system with simulated behavior with this adjustment.

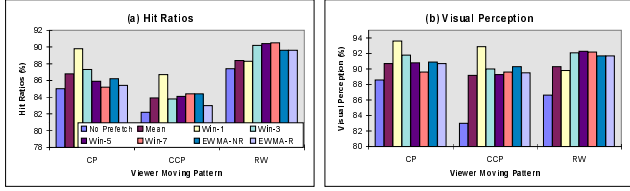


Figure 10: Performance from Experiment #1.

The measurements of the metrics are depicted in Figure 10. For the RW moving pattern, the general behavior of the performance from the prototype is similar to that from the simulation. The only difference is a slight increase in hit ratios and visual perception by a few percents in the experiment, across all prefetching schemes.

For other moving patterns, the improvement in hit ratios from EWMA seems to be smaller than those brought about by simulation. This is perhaps due to the object distribution in the experimental environment. We are still in the process of testing different object distributions with repetition, and will report our findings in the future. The impact on visual perception is similar to hit ratios, but at a smaller scale.

### 6.3.2 Experiment #2

In our second experiment, we study the effect of cache size on the performance of the caching and prefetching mechanisms. To obtain a better understanding on the effect of cache size, we further measure the average response and latency times. Response time refers to the amount of time spent from the moment a client initiates a query for renderable objects to the moment when the optimal resolutions of all renderable objects are available. Latency time refers to the amount of time spent between the initiation of a query to the time the base meshes of renderable objects are available at the client. It measures the observable delay experienced by a viewer when the viewer makes a move.

In this experiment,  $n$  is again fixed at 500 objects. The moving pattern is fixed at CP. The size of the storage cache,  $N$ , ranges from 0% to 2% of the database. Other parameters remain the same. Figure 11 depicts the results, with the second row showing the response and latency times of the experiments. With a cache size of only 0.5% of that of the database, it is able to reduce response and latency times of the application to a quarter and a half respectively, even without prefetching.

We observe an increase in hit ratios and visual perception when the cache size increases as shown in Figure 11. It is simply because a large cache size is able to hold more object models; thus, the chance of hitting an object model in the local cache becomes higher. The improvement in both hit ratio and visual perception from  $N = 0.5\%$  to  $N = 1\%$  is very significant. However, the improvement seems to level off when cache size increases beyond 2%. EWMA is also performing more satisfactory, yielding similar performance as in the simulation.

With respect to response and latency times, they are not as stable as hit ratio and visual perception, due to their heavy dependency on the available network bandwidth when

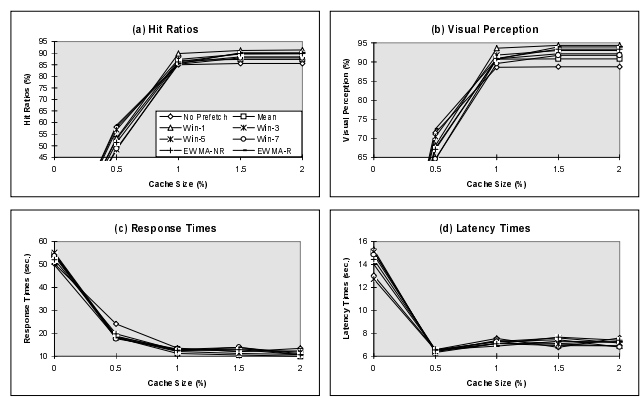


Figure 11: Performance from Experiment #2.

the prototype is running. However, a general observation can still be drawn about their relative performance. With caching, latency time is usually in the order of a few seconds. Compared with other prefetching schemes, the EWMA schemes generally result in a smaller access latency. We also observe that prefetching leads to a small improvement in latency. The response time is about two to four times of the latency, i.e., from 10 to 20 seconds with a cache size of 1% for all movement patterns, as in Experiment #1, and higher with a smaller cache size, as depicted in Figure 11c. However, when compared with no caching, caching alone could improve the response and latency times of the walkthrough application by quite a few times (see Figures 11c and 11d). Prefetching also leads to improvement in response times. Finally, with an increasing cache size, improvement to response and latency times is also observed.

## 7 Conclusions

In this paper, we have described our implementation of a virtual walkthrough system. We describe technical challenges that need to be addressed in order to improve the performance of such kind of applications. As one alternative to improve the performance, we propose a caching mechanism that employs the local storage of a client machine to hold remote objects residing at the database server. The caching mechanism is further complemented by a prefetching mechanism to predict objects accessed in future. The prediction is based on the semantics of virtual walkthrough application. The various prefetching methods are investigated for performance and are shown to be effective.

We are currently conducting more experiments to study the performance of the caching and prefetching mechanism under different situations. In particular, we are studying the effect of multiple clients on the performance of the caching mechanism. We are also investigating the situation when objects are dynamic, i.e., an object can move within the virtual environment. This further complicates our caching mechanism as the updated location of each dynamic object needs to be reflected in the object model cached in each client in a consistent manner.

## Acknowledgements

This research is supported in part by the PolyU grant numbers 350/960 and 351/217.

## References

- [1] J. Calvin, A. Dicken, B. Gaines, P. Metzger, D. Miller, and D. Owen. The SIMNET Virtual World Architecture. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 450–455, 1993.
- [2] 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.
- [3] C. Carlsson and O. Hagsand. DIVE - a Multi-User Virtual Reality System. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 394–400, 1993.
- [4] B. Y. L. Chan, A. Si, and H. V. Leong. Cache Management for Mobile Databases: Design and Evaluation. In *Proceedings of IEEE International Conference on Data Engineering*, pages 54–63, February 1998.
- [5] J. Chim, M. Green, R.W.H. 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.
- [6] J. Chim, R.W.H. Lau, H.V. Leong, and A. Si. Multi-resolution Cache Management in Digital Virtual Library. In *Proceedings of IEEE Advances in Digital Libraries Conference*, pages 66–75, April 1998.
- [7] F. Crow. A More Flexible Image Generation Environment. In *ACM Computer Graphics (SIGGRAPH'82)*, pages 9–18, July 1982.
- [8] M. DeHaemer and M. Zyda. Simplification of Objects Rendered by Polygonal Approximations. *Computers & Graphics*, **15**(2):175–184, 1991.
- [9] W. Effelsberg and T. Haerder. Principles of Database Buffer Management. *ACM Transactions on Database Systems*, pages 560–595, December 1984.
- [10] 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.
- [11] 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.
- [12] C. Greenhalgh and S. Benford. MASSIVE: a Distributed Virtual Reality System Incorporating Spatial Trading. In *Proceedings of the International Conference on Distributed Computing System*, pages 27–34, 1995.
- [13] H. Hoppe. Progressive Meshes. In *ACM Computer Graphics (SIGGRAPH'96)*, pages 99–108, August 1996.
- [14] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh Optimization. In *ACM Computer Graphics (SIGGRAPH'93)*, volume **27**, pages 19–26, August 1993.
- [15] V. İşler, R.W.H. Lau, and M. Green. Real-Time Multi-Resolution Modeling for Complex Virtual Environments. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pages 11–20, July 1996.
- [16] 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.
- [17] R.W.H. Lau, D. To, and M. Green. An Adaptive Multi-Resolution Modeling Technique Based on Viewing and Animation Parameters. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 20–27, 1997.
- [18] 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 IEEE Virtual Reality Annual International Symposium*, pages 2–10, March 1995.
- [19] B. Mannoni. A Virtual Museum. *Communications of the ACM*, **40**(9):61–62, 1997.
- [20] 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.
- [21] T. Ohshima, H. Yamamoto, and H. Tamura. Gaze-Directed Adaptive Rendering for Interacting with Virtual Space. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 103–110, July 1996.
- [22] I.S. Pandzic, T.K. Capin, E. Lee, N.M. Thalmann, and D. Thalmann. A Flexible Architecture for Virtual Humans in Networked Collaborative Virtual Environments. *Eurographics*, **16**(3):177–188, 1997.
- [23] D. Schmalstieg and M. Gervautz. Demand-Driven Geometry Transmission for Distributed Virtual Environments. In *Proceedings of Eurographics '96*, pages 421–432, 1996.
- [24] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of Triangle Meshes. In *ACM Computer Graphics (SIGGRAPH'92)*, volume **26**, pages 65–70, July 1992.
- [25] A. Si and H. V. Leong. Adaptive Caching and Refreshing in Mobile Databases. *Personal Technologies*, **1**(3):156–170, September 1997.
- [26] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 1996.
- [27] G. Singh, 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.
- [28] 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.