In Proceedings of ACM Multimedia, pp. 171-180, Sept. 1998.

On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments

Jimmy H.P. Chim[†] Mark Green[‡] Rynson W.H. Lau[*] Hong Va Leong[†] Antonio Si[!]

[†] Department of Computing, The Hong Kong Polytechnic University, Hong Kong

[‡] Department of Computer Science, University of Alberta, Edmonton, Alberta, T6G 2H1, Canada

[*] Department of Computer Science, City University of Hong Kong, Hong Kong

[!] Sun Microsystems, 901 San Antonio Road, Palo Alto, CA 94303, U.S.A.

Abstract

Advances in networking technology and the establishment of the Information Superhighway have rendered the virtual library a concrete possibility. We are currently investigating user experience in walking through a large virtual environment in the context of Internet. This provides users with the ability to view various virtual objects from different distances and angles, using common web browsers. 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 techniques for multi-resolution object modeling in computer graphics are of great value here, since they are capable of simplifying the object models and therefore reducing the time to render them. Second, with the limited bandwidth constraint of the Internet, we need to reduce the response time by reducing the amount of data requested over the network. One alternative is to cache object models of high affinity. 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 resolution will be useful to provide at least a coarse view of the objects to a disconnected viewer for improved visual perception. In this paper, we propose a *multi-resolution caching mechanism* and investigate its effectiveness in supporting virtual walkthrough applications in the Internet environment. The caching mechanism is further complemented with several *object prefetching mechanisms* for predicting future accessed objects. The performance of our proposed mechanisms and their feasibilities are quantified via simulated experiments.

1 Introduction

Recent establishment of the World Wide Web infrastructure [2] has brought about a revolutionary change to the organization and presentation of information. The technology has allowed multi-media information to be accessed on line in a user-friendly manner across geographical boundaries. Of the numerous emerging multi-media applications, we envision a particular kind of application, the virtual walkthrough application [25], to be of interest.

In a virtual walkthrough application, a user, with access to the Internet, could explore a specific place of interest without having to travel physically. The place of interest will be modeled as a virtual world, containing a vast number of virtual objects. Sample applications of this sort include virtual museum, virtual library, virtual university, etc. [25].

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 world, information of the virtual objects located within a visible distance from the viewer will be conveyed to the client machine of the viewer. The information will then be processed and rendered into images to be viewed by the viewer. Peripheral devices such as head-mount displays or conventional color monitors could be employed to view the rendered images. As the viewer moves within the virtual world, the relative locations and orientations of the virtual 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 a virtual environment where virtual objects are static. The ultimate 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 research. First, virtual objects must be modeled in a compact form. This can reduce not only the amount of storage space needed, but also the time required to transfer the objects from the server to a client under the scare Internet bandwidth. 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. A mechanism that can reduce both the storage space and transmission overhead, but yet provide a reasonable compressing and decompressing efficiency is required. The recently developed multi-resolution object modeling techniques in computer graphics [17, 21] 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 both 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 [13]. A prefetching mechanism allows a client to predict objects that will likely be visible in the future and obtain the objects in advance. 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 world 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 the virtual world.

In this paper, we propose a storage caching and prefetching mechanism that allows virtual objects from a remote database server to be cached in a client's local storage at various resolutions. We term our mechanism, *multi-resolution caching mechanism*, and attempt to quantify the performance via simulated experiments. The rest of the paper is organized as follows. Section 2 presents a survey on relevant research. Section 3 presents an overview of our method and contrasts the differences of our approach with previous ones. In Section 4, we present our multi-resolution modeling technique. Section 5 presents the multi-resolution caching and prefetching mechanism. In Section 6, we present and discuss the performance of our mechanism via several simulated experiments. Finally, we conclude our paper with a discussion on possible future work.

2 Related Work

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 also has an advantage of persistence. When disconnected from the server, a client can still operate on the database objects available 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 [4, 12, 13]. This is primarily because the server's storage is also page-based. The overhead for transmitting one item or a page is similar. In general, a page-based mechanism requires a high degree of locality among the items within each page to be effective [10].

In a virtual walkthrough environment, 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 narrow bandwidth Internet is very high. Furthermore, we might not always need to render an object at its full resolution (see Section 5). Therefore, there will be situations that we need to transfer less than a page of information and there also exists situations that we need to transfer more than a page of information. A more dynamic granularity for caching is needed in this virtual walkthrough environment.

To simplify the architectures, most existing virtual walkthrough systems are replication-based approach [1, 3, 5, 14, 15, 28]. The geometry database is first transferred to the client machine (or the client site where a high speed network connects the client and the local geometry database server). In an Internet environment, this approach is possible only if either the size of the geometry database is small or the client can provide unbounded disk storage and wait for a possibly very long pre-loading time, However, a more realistic situation is that the available storage for caching and the available pre-loading time are limited [26]. A different approach to database replication by transmitting the object models on demand to the clients during the walkthrough [23, 29] is a better alternative in this Internet environment. In [34], a similar approach is adopted by transmitting program codes for constructing the object models at the client. These approaches shorten the pre-loading time.

2.1 Multi-Resolution Modeling

In a virtual walkthrough application, rendering a complex object at a client is expensive. We notice that from the perspective of a viewer in the virtual world, distant objects occupy smaller screen areas 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 the transmission delay and the storage required to hold the objects in a client's local storage. This could also reduce the rendering overhead. To optimize both the transmission delay and the rendering performance, we employ multi-resolution modeling techniques here to study the effect of caching objects in a client at various degrees of granularity. 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 [9, 18, 30, 35]. 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 [8]. Although this method is fast and simple, it has two major limitations. First, when the object's distance from the viewer crosses the threshold distance, there is a sudden change in model resolution and an objectionable visual discontinuity effect can be observed. In [35], Turk proposes to have a transition period during which a smooth interpolation between the two successive models is performed to produce models of intermediate resolutions. This method, however, further increases the computational cost during the transition period because of the need to process two models at the same time. Second, because 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 [17]. The method assumes that the object is composed of triangles, which is commonly used for representing 3D objects. It is based on two operators, *edge col-*

lapse for reducing model resolution, and *edge split* for increasing model resolution. Each object is modeled as an ordered list. The list begins with a minimal resolution model of the object, also represented in the form of a triangle model. Each subsequent record in the list stores information of an edge split. The structure of a progressive mesh is shown in Figure 1. 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 applied.

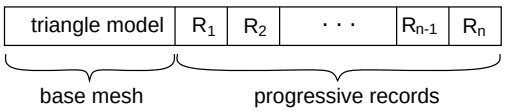

Figure 1: The structure of a progressive mesh.

To create the list, the method begins with the highest resolution model of the object. A triangle edge located in a locally flat region is selected, and collapsed into a single point. Such a collapse will result in the removal of two triangles from the model. The information related to this single edge collapse is packaged in a single record and placed at the end of the list. This edge collapsing operation is continued and the corresponding records are inserted in the list in a reverse order until the model is reduced to its minimal resolution. This minimal resolution model is then added to the beginning of the list to form the complete list. This list is referred to as the progressive mesh, and the minimal resolution model is referred to as the *base mesh*. Each of the records in the simplification list is known as a *progressive record*. We have recently developed a similar method [19, 21].

2.2 Replacement and Prefetching Techniques

In [11], 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 [4, 13, 33]. In [31], 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 [6] that prefetching could be very beneficial in improving the performance of database applications if the prefetching is performed intelligently.

3 Method Overview

In our walkthrough application within 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 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, $\check{\mathcal{R}}_o$. 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 will increase 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 and $\hat{\mathcal{L}}_o$ is the number of progressive records.

Each object and viewer is defined with a *scope*. We denote the viewer scope for viewer, V, by $\bigcirc_V$ and the object scope for object, o, by $\bigcirc_o$. Intuitively, an object scope of an object defines the area within which the object will be visible. It is roughly proportional to it 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 for transmitting and rendering small objects.

The interaction between a viewer and the virtual world is illustrated in Figure 2. 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 will be considered for rendering. 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 visible to the viewer if they are not obscured by another object. We refer to these objects as *visible objects*.

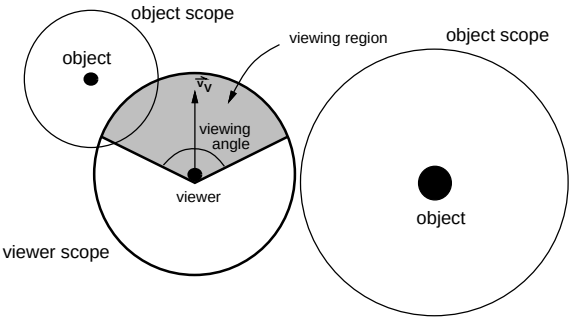

Figure 2: Organization of the virtual world.

If the two scopes overlap, we need to determine the appropriate resolution of the object for rendering. We note that the relationship between the rendered resolution of an object and the perceived image quality (*visual perception*) of the object by the viewer is not linear. We denote such a relationship in Figure 3. As shown in the diagram, each object will be rendered at least at its minimum resolution when it is rendered using its base mesh only. Further notice that each object is also associated with an *optimal resolution*. If the object is rendered at a resolution higher than its optimal resolution, the additional details will not be easily noticeable to the viewer. Rendering an object higher than its optimal resolution is therefore a waste of resources. By contrast, if the object is rendered at a resolution lower than this optimal resolution, the visual perception experienced by a viewer drops rapidly. It is therefore, preferable to render an object at its optimal resolution in order to strike a balance between rendering efficiency and visual perception. This optimal resolution of an object is defined as a percentage of the its maximum resolution, i.e., the number of progressive records, and 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 (see Section 4). We denote the optimal resolution of an object, o, by $\mathcal{R}_o$ and the corresponding resolution level by $\mathcal{L}_o$.

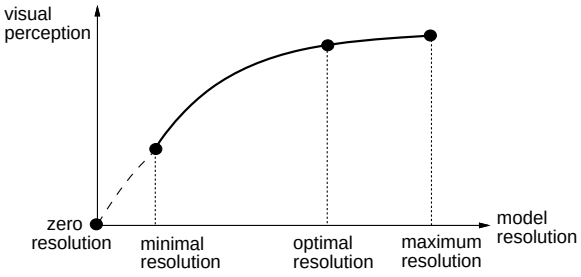

Figure 3: The effect of model resolution on visual perception of an object on the viewer.

During the walkthrough, we continuously determine those objects in the virtual environment 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. When transmitting the models from the server to a client, those for 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. The identification of victim objects is based on their probabilities of being viewed at higher resolution in the future, rather than adopting the conventional LRU scheme (see Section 5).

We also attempt to further improve the performance of the walkthrough application by having the server to prefetch objects which will most probably be accessed in the near future to the client. Given the current location and viewing direction of a viewer, the database server predicts 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 location will be transmitted at their optimal resolutions to the client as well.

The advantages of our method can be summarized as follows:

- In [29], discrete multi-resolution method is used for model transmission. Redundant information will have to be 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 resolution of the viewing device.
- Our caching mechanism differs from conventional caching mechanisms [4, 13, 33] in that objects could be cached at multiple degree of granularity. Replacement is also based on object access patterns rather than adopting the conventional LRU scheme.
- We further improve the performance of the walkthrough application by predicting the future movement of the viewer and prefetching objects in advance.

4 Multi-Resolution Modeling Technique

4.1 Object Scope

In order to render an image representing what the viewer is supposed to see during a walkthrough, the objects that are visible to the viewer at each frame need to be determined. To minimize the number of objects needed to be handled, it is proposed that the environment be divided into regions in [14, 15]. Objects that are visible to each of the regions are precomputed. The list of potentially visible objects is readily available during run-time by determining the region that the viewer is located. The limitation of this method, however, is that the computational cost of updating the lists during run-time can be high if the objects in the environment may freely move around.

A different approach is to consider the *area of interest* (AOI) of the viewer [12, 24, 29]. 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*. 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 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. The definition of the object scope is different. 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 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 are 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 [16].

In general, the size of an rendered object from a viewpoint might be significantly different from that for another viewpoint such as a long pillar object. Hence, the size of the scope for an object at different viewpoint might be different as well. However, defining a scope at such a fine granularity would result in very high computation overhead when identifying renderable objects. We, therefore, simplify the definition of a scope (viewer or object) to a circular region in our implementation. Each scope is centered at the center

of gravity of the object or viewer, and 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 $r_{\bigcirc_V}$.

4.2 The Optimal Resolution of an Object Model

The optimal resolution of an object model can be defined according to the *visual importance* of the object to a viewer. In [22], 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 of the object [27, 36]. 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 [20]. 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 4 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 θ_o ($-\pi \leq \theta_o \leq \pi$). The visual importance of o, i.e., I_o to a viewer can be defined with the following formula:

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

where K_o is a constant for adjusting the decrement rate of object o due to the increase in 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. Notice that $0 \leq I_o \leq 1$.

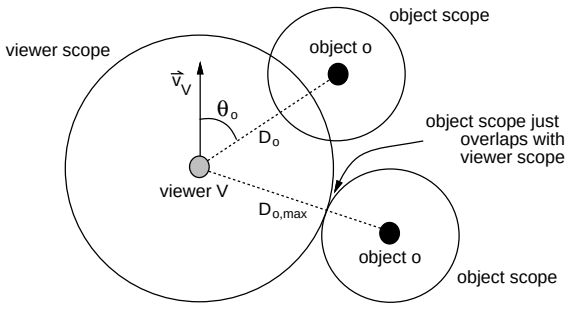

Figure 4: Visual importance of an object to a viewer in a virtual environment.

The optimal resolution of object o is defined as I_o fraction of the number of progressive records. Notice that when the object scope touches the perimeter of the viewer scope, D_o is equal to $D_{0,max}$ and I_o becomes 0. The object will be rendered using its base mesh only. As the object moves closer to the viewer or to the viewer's line of sight, the resolution level of the object increases.

5 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, 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. Notice that memory caching at a client can be transparently manipulated by the local operating system, in the form of virtual memory [32]. The only concern with us here is the utilization and management of the storage, i.e., local disk.

5.1 The Cache Model

When a viewer, V, moves within the virtual world, its client machine, C, will transmit the current viewing direction, $\vec{v}_V$, and the current location, loc_V, of V to the server. This message could be treated as a query to the geometry database server, requesting for all renderable objects, including the prefetched ones.

While the database server is processing the query, the client C will concurrently identify the cached objects that are renderable from its local storage cache. Notice that 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, containing a list of $\langle o, \mathcal{L}^*_o \rangle$ pairs. This list informs the database server about those renderable objects that are cached in C's storage and their maximum renderable resolution levels. Those renderable objects whose $\mathcal{L}^*_o \geq \mathcal{L}_o$ do not need to be transmitted to the client as the client is able to render at the optimal resolution, $\mathcal{R}_o$, from locally cached data. On the other hand, those renderable objects not cached in C or those which do not have sufficient records for the required resolution will have to be transmitted to the client. A *result list* of o in the form of $\langle o$, progressive mesh$\rangle$ will be transmitted from the server. The progressive mesh only contains enough progressive records to define the optimal resolution, and it might or might not include the base mesh, depending on whether the object is partially cached in the client.

Upon receiving the result list from the server, C might cache the objects in its local storage. If the storage is exhausted, a replacement scheme will be invoked to identify the victim objects to be discarded to accommodate the incoming objects. For each virtual object, o, an *access score* S_o indicating the prediction of its future access probability is determined. The higher is the access score, the higher is the probability that o will be accessed again soon. If a new object has an access score higher than the lowest access score of some currently cached objects, the storage space of some of the cached objects will be reclaimed and allocated to accommodate the new object.

5.2 Replacement Mechanism

We employ the *Most Required Movement* (MRM) replacement technique in defining the access score for each object. The scheme is based on the observation that the further an object is from the viewer, the lower the resolution it can be rendered and the longer it will take for the viewer to move to view the object in greater detail. Consequently, its probability of having it rendered at a higher resolution and hence

its value of being cached in the storage are lower. Similarly, the larger is the angle between an object and the viewer's line of sight, the lower is the resolution required and the longer it will take for a client to rotate to view the object directly in front. The probability of being rendered at a higher resolution and its value of being cached in the storage are also lower. Preliminary experiments have shown that such a replacement scheme outperforms traditional LRU replacement scheme [7]. We investigate the effectiveness of MRM in more detail here, with consideration 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 our scheme to have as few parameters to be adjusted as possible, we define the access score by a linear function of the distance between o and V, D_o, and the angular distance between o and $\vec{v}_V$, θ_o. The access score is thus defined using only one single adjustable parameter, ω ($0 \leq \omega \leq 1$):

$$S_{o,V} = \omega(1 - \frac{D_o - r_{\bigcirc_o}}{r_{\bigcirc_V}}) + (1-\omega)(1 - \frac{|\theta_o|}{\pi})$$

When the object with the lowest access score, o, is selected for replacement, we will not remove the whole object from the storage cache immediately. Rather, its extra resolution detail, $\mathcal{R}_o^*$, will be reduced to its optimal resolution, $\mathcal{R}_o$, 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. Again, all its extra progressive records will be removed. This process will be iterated until enough room is allocated for all new objects.

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 objects with the lowest access score will be removed from the storage cache, leaving only the base mesh of the object at its minimum resolution, $\check{\mathcal{R}}_o$. Again, this process will be iterated for every currently cached object until there is enough room for all new objects. Finally, the base mesh of the object with the lowest access score will be removed if there is still not enough room after removing all progressive records of all cached objects. This process will be iterated until enough room is allocated for all new objects.

This multi-resolution replacement scheme tries to keep a coarse resolution of an object in a client's storage as much as possible. This provides a 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 at a low resolution.

5.3 Prefetching Mechanism

To enable prefetching, the server maintains a separate profile for each viewer V, containing the set of historical *movement vectors*, $\{\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_{n-1}\}$. 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, $\vec{m}_n$, 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 renderable if V were at loc_{n+1}, in addition to the renderable 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 three schemes are depicted in Figure 5.

In the mean scheme, the next movement vector at loc_{n+1} is predicted as the average of the previous n movement vectors. The intuitive meaning of the mean scheme is depicted in Figure 5*a*, predicting the 4^{th} movement vector by averaging the previous three movement vectors. In the figure, we only illustrate the direction of the predicted movement vector. Let us denote the movement vector in the n^{th} step by $\vec{m}_n$ and the predicted movement vector for the next step by $\hat{m}_{n+1}$. The predicted vector will then be $\hat{m}_{n+1} = \frac{1}{n}\sum_{i=1}^{n}\vec{m}_i = \frac{(n-1)\hat{m}_n + \vec{m}_n}{n}$. It is not difficult to see that the intermediate vectors, and hence viewer locations, are not required in predicting the new vector; only a "running sum" suffices.

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 idea is indicated in Figure 5*b*, indicating a window of size $W = 2$. In general, with a window of size W, the new predicted movement vector $\hat{m}_{n+1} = \frac{(n-1)\hat{m}_n + \vec{m}_n}{n}$ when $n \leq W$. When $n > W$, the predicted vector is $\hat{m}_{n+1} = \hat{m}_n + \frac{1}{W}(\vec{m}_n - \vec{m}_{n-W})$. In this case, some of the intermediate movement vectors, and hence some of the viewer locations, should be maintained.

A problem for the window scheme is the amount of storage needed in maintaining the vectors or viewer locations within the windows. To avoid the need of a moving window, and to adapt quickly to changes in viewer moving patterns, our third scheme assigns 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 in the scheme 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. A high α will give similar weights to all the movements and predict future movements as a function of many movements, including the aged ones. This idea is indicated in Figure 5*c*, indicating the predicted moving direction with a high α. By contrast, Figure 5*d* indicates the predicted moving direction with a low α in which aged movements will fall off the scene quickly and the prediction is biased towards contributions from recent movements. The new predicted vector is $\hat{m}_{n+1} = \frac{1}{S_n}\sum_{i=1}^{n}\alpha^{n-i}\vec{m}_i$, 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 $(1-\alpha)\sum_{i=1}^{n}\alpha^{n-i}\vec{m}_i$. Incrementally, $\hat{m}_{n+1}$ can be approximated by $\alpha\hat{m}_n + (1-\alpha)\vec{m}_n$.

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 [31], 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. The rate of change, i.e., the first derivative of the access frequency function, must either oscillate between positive and negative, or it will gradually fall to zero (the ideal case). 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, 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

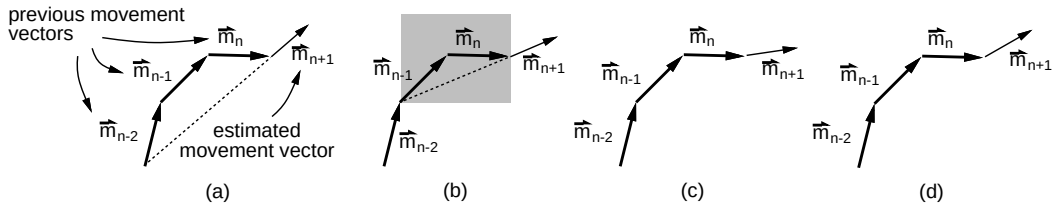

Figure 5: Predicting next moving direction: (a) mean, (b) window, (c) EWMA with high α, and (d) EWMA with low α.

circle. Thus, the first derivative may never fall to zero or cross zero and 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 *residual* in each prediction of the movement vector by $\vec{e}_n = \hat{m}_n - \vec{m}_n$. As a result, the adjusted prediction for the vector should be $\hat{m}_n - \vec{e}_n$. To cater for the problem of non-stationary changes in direction, we restrict our attention of the residual to the angle between $\hat{m}_n$ and $\vec{m}_n$. We will denote that angle 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 so that $\vec{m}_n$ can be predicted by rotating $\hat{m}_n$ through an angle of $-\phi_n$. This corresponds to a multiplication by $e^{-\imath\phi_n}$ in the complex plane. 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, such as 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 $\vec{m}_{n+1} = \hat{m}_{n+1}e^{-\imath\hat{\phi}_{n+1}}$ Notice that we only need to maintain a single running value for this angle argument. In next section, we will illustrate the improvement in prefetching performance brought about by this residual adjustment.

6 Results and Discussions

Since we have done some preliminary studies on comparing our MRM replacement scheme with traditional LRU scheme [7], our experiments presented here try to quantify the performance of the MRM caching scheme and that of various prefetching schemes under different situations. The parameters used in our simulation model are listed in Table 1.

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

Table 1: Parameters listing for simulated experiments.

In our simulation model, there are n virtual objects, all residing within the secondary storage at the database server. In the experiments presented below, we focus on an environment with one database server and a single client. The effect of the number of clients on the performance of caching and prefetching schemes will be reported in the future. The client and database server communicate via a network with a bandwidth of 2Mbps, modeling the Internet environment. The disk bandwidth is fixed at 40Mbps. The computation cost for determining scope intersection, optimal resolution, and access score are ignored for simplicity. The virtual world is regularly subdivided into 2000×2000 square units. The n virtual objects are distributed uniformly among the square units. Each unit therefore contains an average of $\frac{n}{4 \cdot 10^6}$ objects. The viewer is assumed to reside at the center of the viewer scope. The viewing angle which defines the viewing region of the viewer scope is set to be 120 degrees, i.e., $\frac{2\pi}{3}$.

Each virtual object in the database server 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 is assumed to have a size equal to 2KB. The actual sizes of the object models range between 660KB and 1,363KB, with a mean of 1,020KB.

Only storage cache at the client is modeled in our simulation. The size of the storage cache is equal to N% of that of the database. We experimented with different prefetching schemes, F_{disk}, including mean, Window, and EWMA. 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 by EWMA-R, and the EWMA scheme with residual adjustment disabled by EWMA-NR. We further denote a window scheme with window size, W, as Win-W.

We simulate three moving patterns, P, experienced by a viewer. The moving patterns are depicted in Figure 6. Each pattern contains a sequence of movement steps. The first pattern models a constant circular translation pattern (CP). The viewer moves circularly with a diameter of 715 square units, starting at coordinate $\langle 240, 1000 \rangle$ and ending at the same location. Each movement step includes a translation of 150 square units along the viewing direction, followed by rotating the viewing direction with an angle of $\frac{\pi}{15}$, i.e., 12 degrees. At every position, the viewer also rotates with an angle of $\pm\frac{\pi}{9}$, i.e., ±20 degrees. This models a situation where the viewer explores the virtual objects around him/her for every movement. The second pattern models the same pattern as the circular pattern except that the moving direction changes with an angle of $\frac{\pi}{36}$, i.e., 10 degrees, after every 4 movement steps. We call this the changing circular pattern (CCP). Finally, the last pattern models a random movement pattern (random walk or RW). Each movement step is either a translation of arbitrary length or a rotation of arbitrary angle.

We characterize the performance of the caching and replacement schemes with two sets of metrics. The first set measures the absolute performance and is characterized by *cache hit ratio* and *response time*. Cache hit ratio measures the percentage of bytes of the visible objects, i.e., those within the viewing region, that could be retrieved from the

Parameter	Experiment #1	Experiment #2	Experiment #3
n	5000	5000	1000, 5000, 10000, 20000
N	1%	0.5%, 0.6%, 0.7%, 0.8%, 0.9%, 1%	1%
ω	0.5	0.5	0.5
F_{disk}	No Prefetch, Mean, Window EWMA-NR, EWMA-R	No Prefetch, Mean, Window EWMA-NR, EWMA-R	No Prefetch, Mean, Window EWMA-NR, EWMA-R
W	1, 3, 5, 7	1, 3, 5, 7	1, 3, 5, 7
α	0.5	0.5	0.5
P	CP, CCP, RW	CP	CP

Table 2: Parameter values for the experiments.

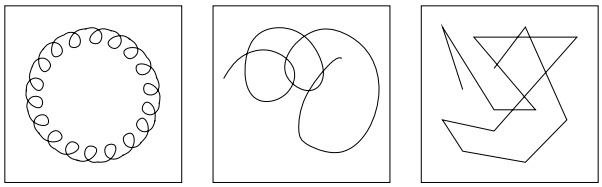

Figure 6: Moving patterns: (a) CP, (b) CCP, and (c) RW.

local storage cache of the client. Response time measures the average time (in seconds) that the viewer needs to wait from the moment the viewer makes a move to the moment when the records for the optimal resolutions of all visible objects are obtained at the client.

The second set of metrics is concerned with the image quality perceived by a viewer and is characterized by *visual perception* and *latency time*. Visual perception measures the degree (in percentage) of image quality experienced by the viewer. It is difficult to model visual perception as it requires user visual experiences within a virtual system. Currently, 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. This definition assumes a 100% visual perception when the cached model could provide the optimal resolution. 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. We are conducting experiments to collect user experiences via a prototype which will allow us to refine our visual perception definition. Along the same rationale, latency time measures the average time (in seconds) required to retrieve the base meshes of all visible objects, i.e., from the time 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. The standard deviations are found to be small.

Due to space limitation, we will only be presenting three experiments to quantify the performance of MRM and the prefetching techniques. The parameters settings are summarized in Table 2.

6.1 Experiment #1

In the first experiment, we would like to study the performance of the caching mechanism, with and without prefetching, on the various moving patterns. In this experiment, n is fixed at 5000 objects, resulting in a database size of approximately 5GB. The size of the storage cache, N, is fixed at 1% of n, leading to a cache size of 50MB. We experimented with all three moving patterns: CP, CCP, RW. We repeat our experiments with various prefetching schemes: Mean, Window, EWMA-NR, and EWMA-R, to be compared with the base case of no prefetching, No Prefetch. The measurements of the four metrics are depicted in Figure 7, which is organized as an array of graphs. The first row (Figures 7a and 7b) depicts the hit ratios and response times while the second row (Figures 7c and 7d) depicts the visual perception and latency measurements.

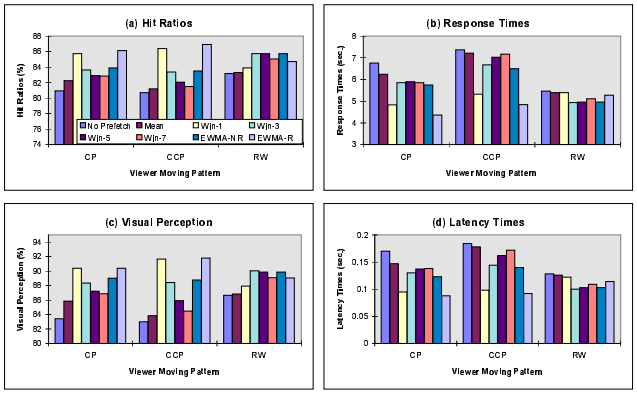

Figure 7: Performance of Experiment #1.

From Figure 7, we observe that even without prefetching, the caching mechanism performs reasonably well. It achieves a hit ratio ranging from 79% to 83% (Figure 7a). With prefetching, the hit ratios could be improved by up to 6%. This results in a decrease in response time when prefetching is enabled (Figure 7b). 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 ratio of the caching mechanism.

With a window-based prefetching scheme, a small window size will result in better performance under the CP and CCP moving patterns. This is mainly because under the CP and CCP moving patterns, the moving direction is always changing, very often with a constant angle. With a large window size, aged moving vectors will be contributing to the prediction of the next moving vector; this introduces some noise in the prediction. The performance is thus not as good as that with a small window size. By contrast, under the RW moving pattern, each movement step will bear a high degree of randomness. A small window does not necessarily capture enough knowledge to predict the next movement vector. Therefore, the performance from a small window

size is not as good as that from a large window size under RW.

EWMA exhibits a similar behavior as window-based scheme and EWMA-R performs better under the CP and CCP moving patterns. This is mainly because the angles of deviation under these two moving patterns exhibit a well-defined pattern (quite constant) and is thus predictable. Under a RW moving pattern, the angle deviation does not exhibit a clean pattern and the residual correction does not seem to yield any improvement.

6.2 Experiment #2

In the second experiment, we would like to study the effect of cache size on the performance of the caching and prefetching mechanisms. In this experiment, n is again fixed at 5000 objects. The moving pattern is fixed at CP. The size of the storage cache, N, ranges from 0.5% to 1% of n, leading to a cache size of 25MB to 50MB. Again, the experiments are repeated with different prefetching schemes. Figure 8 depicts the results which are arranged in the same way as Figure 7.

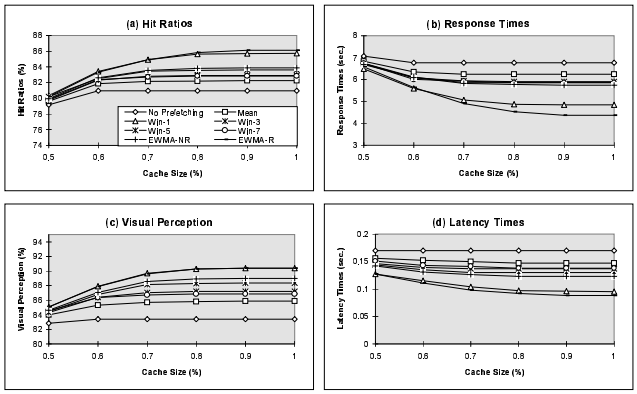

Figure 8: Performance of Experiment #2.

From Figure 8a, we observe a gradual increase in hit ratios when the cache size increases. This also accounts to a decrease in response times when the cache size increases as shown in Figure 8b. It is simply because a larger cache size is able to hold more object models; thus, the chance of hitting an object model in the local cache becomes higher. It is also promising that even without prefetching, we are able to achieve a hit ratio of 79%, with a cache size as small as 0.5% of the database.

Figures 8c and 8d depict the visual perception and latency time experienced by the viewer. They exhibit similar behavior as hit ratio and response time respectively. Notice that the latency time is always under a second. This could provide an almost continuous service to the viewer.

6.3 Experiment #3

In this last experiment, we would like to study the effect of database size on the performance of the caching and prefetching mechanisms. In this experiment, n ranges from 1000, 5000, 10000, and 20000 objects. This models environments with small, medium, and large databases respectively. The size of the storage cache, N, is fixed at 1% of n. The moving pattern is fixed at CP. The experiments are repeated with different prefetching schemes. Figure 9 depicts the results which again, are arranged in an identical manner as Figure 7.

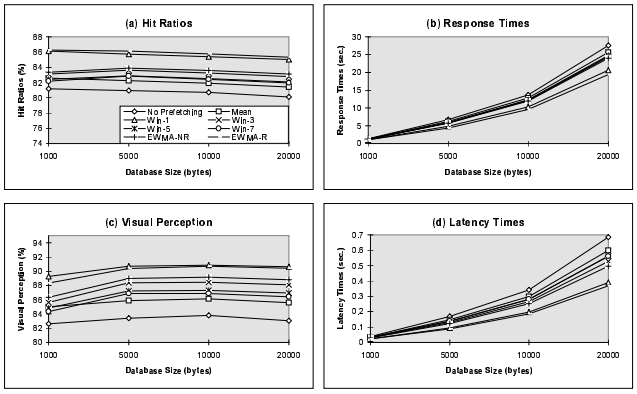

Figure 9: Performance of Experiment #3.

From Figure 9, we observe that the hit ratios and visual perception drop slightly as the database size increases (Figures 9a and 9c). As the database size increases, the expected number of object models that are required for each movement also increases accordingly. This is because the number of virtual objects located within a unit increases with the database size. Therefore, at any location, the likelihood that the viewer scope will overlap with an object scope becomes higher. This will decrease the chance of hitting an object model in the local cache. Since the size of the cache is fixed at 1% of the database size, the decrease in hit ratios is rather slight. The increase in response times and latency times (Figures 9c and 9d) is due to the decrease in hit ratios and could be explained similarly.

7 Conclusions

In this paper, we have described a virtual walkthrough application as an example of next-generation multi-media applications. We describe technical challenges that needed 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 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 future access objects. The prediction is based on the semantics of virtual walkthrough application. The various prefetching methods are investigated for performance and 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 developing a prototype on the Web environment to validate the actual performance with our simulated results.

Acknowledgements

We would like to thank the anonymous reviewers for their invaluable and constructive comments that help to improve this paper. This research is supported in part by the Hong Kong Research Grant Council under CERG grant number PolyU 5111/97E, succeeded by PolyU grant number 350/960, and by the Hong Kong Polytechnic University Central Grant numbers 351/217 and 351/495.

References

[1] J. Barrus, R. Waters, and D. Anderson. Locales: Supporting Large Multiuser Virtual Environments. *IEEE Computer Graphics abd Applications*, 29(11):50–57, 1996.

[2] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, 1994.

[3] 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.

[4] 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.

[5] 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.

[6] B. Y. L. Chan, A. Si, and H. V. Leong. Cache Management for Mobile Databases: Design and Evaluation. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 54–63, February 1998.

[7] 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.

[8] F. Crow. A More Flexible Image Generation Environment. In *ACM Computer Graphics (SIGGRAPH'82)*, pages 9–18, 1982.

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

[10] 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.

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

[12] 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.

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

[14] T. Funkhouser, C. Sequin, and S. Teller. Management of Large Amounts of Data in Interactive Building Walkthoughs. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics*, pages 11–20, 1992.

[15] T. Funkhouser, S. Teller, C. Sequin, and D. Khorramabadi. The UC Berkeley System for Interactive Visualization of Large Architectural Models. *Presence: Teleoperators and Virtual Environments*, 5(1), January 1996.

[16] 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.

[17] H. Hoppe. Progressive Meshes. In *ACM Computer Graphics (SIGGRAPH'96)*, pages 99–108, 1996.

[18] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh Optimization. In *ACM Computer Graphics (SIGGRAPH'93)*, volume **27**, pages 19–26, 1993.

[19] 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.

[20] 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.

[21] 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*, 7(1):22–35, February 1998.

[22] 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, March 1997.

[23] 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.

[24] 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.

[25] B. Mannoni. A Virtual Museum. *Communications of the ACM*, 40(9):61–62, 1997.

[26] 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 the International Conference on Extending Database Technology*, pages 323–336, 1994.

[27] 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.

[28] I. Pandzic, T. Capin, E. Lee, N. Thalmann, and D. Thalmann. A Flexible Architecture for Virtual Humans in Networked Collaborative Virtual Environments. *Eurographics'97*, **16**(3):177–188, 1997.

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

[30] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of Triangle Meshes. In *ACM Computer Graphics (SIGGRAPH'92)*, volume **26**, pages 65–70, July 1992.

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

[32] A. Silberschatz and P. Galvin. *Operating System Concepts*. Addison-Wesley, Reading, Massachusetts, 5th edition, 1998.

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

[34] 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.

[35] G. Turk. Re-tiling Polygonal Surfaces. In *ACM Computer Graphics (SIGGRAPH'92)*, volume **26**, pages 55–64, 1992.

[36] 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.