CyberWalk: A Web-Based Distributed Virtual Walkthrough Environment

Jimmy Chim, Rynson W. H. Lau, *Member, IEEE*, Hong Va Leong, and Antonio Si, *Member, IEEE*

***Abstract*—A distributed virtual walkthrough environment allows users connected to the geometry server to walk through a specific place of interest, without having to travel physically. This place of interest may be a virtual museum, virtual library or virtual university. There are two basic approaches to distribute the virtual environment from the geometry server to the clients, complete replication and on-demand transmission. Although the on-demand transmission approach saves waiting time and optimizes network usage, many technical issues need to be addressed in order for the system to be interactive.**

CyberWalk is a web-based distributed virtual walkthrough system developed based on the on-demand transmission approach. It achieves the necessary performance with a *multiresolution caching mechanism*. First, it reduces the model transmission and rendering times by employing a progressive multiresolution modeling technique. Second, it reduces the Internet response time by providing a caching and prefetching mechanism. Third, it allows a client to continue to operate, at least partially, when the Internet is disconnected. The caching mechanism of CyberWalk tries to maintain at least a minimum resolution of the object models in order to provide at least a coarse view of the objects to the viewer. All these features allow CyberWalk to provide sufficient interactivity to the user for virtual walkthrough over the Internet environment.

In this paper, we demonstrate the design and implementation of CyberWalk. We investigate the effectiveness of the multiresolution caching mechanism of CyberWalk in supporting virtual walkthrough applications in the Internet environment through numerous experiments, both on the simulation system and on the prototype system.

***Index Terms*—Distributed virtual environments, model prefetching, multiresolution caching, multiresolution modeling, virtual walkthrough.**

I. Introduction

IN A VIRTUAL walkthrough application, a viewer could explore a specific place of interest without having to travel physically. The place of interest is usually modeled as a virtual environment, containing a vast number of virtual objects. Sample applications of this sort include virtual museum, virtual library, and virtual university [18]. CyberWalk is a distributed virtual walkthrough system, employing a standard client–server architecture. Information of virtual objects, including their locations and geometric shapes, is maintained in a central database server. A viewer using a client machine connected to the Internet may access and visit the virtual environment provided by the server. When the viewer walks through the environment, information of virtual objects located within a visible distance from the viewer will be conveyed to the client machine for rendering. In general, virtual objects could be dynamic, 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 of CyberWalk is to provide good performance, 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 research issues in CyberWalk. First, virtual objects must be modeled in a compact form so as to reduce the storage space needed and the 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 overheads in compressing and decompressing the objects. CyberWalk employs the progressive multiresolution technique for object modeling [12], [15]. The technique allows progressive transmission of object models 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. CyberWalk addresses this problem by employing 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 [10]. A prefetching mechanism allows a client to predict objects that are likely to 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 needed in the future.

Third, the Internet often suffers from various degrees of disconnection. The local 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 their existence.

Manuscript received December 1, 2000; revised March 18, 2002. This work was supported in part by an SRG grant from the City University of Hong Kong (Project 7001071), and the Hong Kong Polytechnic University Central Grant (Grant G-T040). The associate editor coordinating the review of thi spaper and approving it for publication was Dr. Chung-Sheng Li.

J. Chim was with the Department of Computing, the Hong Kong Polytechnic University, Hong Kong. He is now with the School of Visual Arts, New York, NY 10010-3994 USA.

R. W. H. Lau is with the Departments of Computer Science and Computing Engineering and Information Technology, City University of Hong Kong, Kowloon, Hong Kong (e-mail: rynson@cs.cityu.edu.hk).

H. V. Leong is with Department of Computing, The Hong Kong Polytechnic University, Hong Kong.

A. Si is with Oracle Corporation, Redwood Shores, CA 94065 USA.

Digital Object Identifier 10.1109/TMM.2003.819094

The rest of the paper is organized as follows. Section II presents a survey on relevant research. Section III presents the design of CyberWalk in detail. Section IV discusses the implementation of our experimental prototype system. Section V quantifies the performance of our caching and prefetching mechanisms with several experiments on CyberWalk. Finally, Section VI concludes our paper with a discussion on possible future work.

II. Related Work

In this section, we summarize existing work in three different areas which are fundamental to our work. First, we briefly look at the strengths and limitations of existing distributed virtual walkthrough systems. Next, we summarize related multiresolution methods for object modeling. Finally, we point out the rationale for employing replacement and prefetching techniques in CyberWalk.

A. Distributed Virtual Walkthrough Systems

Two main approaches have been proposed to distribute virtual objects from the server to the clients in distributed virtual reality applications [25]. Most systems, such as DIVE [4], SIMNET [2], and VLNET [20], employ a *complete replication* approach to distribute all geometry data to the clients before the start of the application. Since the geometry database is usually large in size, this approach assumes the use of a high speed network in order to reduce preloading time and may not be suitable for CyberWalk. A few systems employ the *on-demand transmission* approach to distribute geometry data to the clients [9], [21], [24] at runtime. When the virtual environment is large, a viewer would likely only visit a small section of it. This approach requires only the visible region of the environment to be transmitted to the client and can thus reduce startup time and network traffic. It, however, introduces a number of problems. In particular, we need to fetch the visible objects from the server in advance so that they can be available in the client when they are needed, in order for the walkthrough application to be interactive. In [27], this interactive problem was addressed as a scheduling problem, with emphasis on deadline and admission control of clients to sustain certain level of quality of service. However, the research work did not consider the client cache and, more importantly, did not make use of the multiple granularity of the object models, which constitute two major sources of performance improvement in CyberWalk. We address this interactive problem as well as related problems, such as continued viewing service in momentary network congestion and client disconnection, with a number of solutions as will be described in Section III.

In a conventional client-server database environment, data objects are usually transferred from the database server to a client on a per-page basis [3], [10]. This is primarily because the server's storage is also page-based. The overheads for transmitting one item or a page are similar. In a distributed virtual environment, virtual objects are represented using *object models*. These models are usually large in size, occupying possibly multiple pages. The cost to transfer them in their entirety via the Internet is very high. Furthermore, we might not always need to render an object at its full resolution. Hence, there will be situations that we need to transfer less than a page of information and situations that we need to transfer more than a page of information. A more dynamic granularity for caching is therefore needed in a distributed virtual environment. To combat frequent network disconnections of the Internet, a storage cache, which has the advantage of persistence, may be established. When disconnected from the server, a client can still operate on the cached database objects in its local storage.

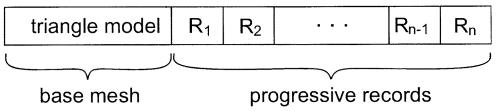


Fig. 1. Structure of a progressive mesh.

B. Multiresolution Modeling

In a distributed environment, rendering a complex object at a client is computationally expensive. From the perspective of a viewer in the environment, distant objects appear smaller than nearby objects after perspective projection. Most details of a distant object are actually not visible to the viewer. Hence, it is only necessary to represent the object at a resolution high enough for a given viewing distance. This could reduce not only the rendering time, but also the transmission delay and the storage space required at the client. CyberWalk employs multiresolution modeling techniques for caching and prefetching objects in a client at various granularities, with nearby objects at higher resolution and distant objects at lower one.

A method referred to as *progressive meshes* was recently proposed to encode object models for progressive transmission [12]. The method is based on two operations, *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 a minimal resolution model of the object, referred to as the *base mesh*. Each subsequent record in the list, referred to as a *progressive record*, stores information of an edge split. The structure of a progressive mesh is shown in Fig. 1. If we apply each progressive record to the base mesh in order, the object model will gradually increase in resolution until it reaches the maximum. Conversely, the method may begin with the highest resolution model. If we apply each of the records in reverse order, which is equivalent to an edge collapse operation, the object model will gradually decrease in resolution until it reaches the resolution of the base mesh. We have recently developed a similar method independently [13], [15].

C. Replacement and Prefetching Techniques

If a client can provide unbounded disk storage and wait for a possibly very long preloading time, we could transmit all virtual objects in the environment to the client before starting the walkthrough. This approach is adopted by some existing distributed virtual walkthrough systems [2], [4], [20]. However, a more realistic situation is that the available cache storage and preloading time are limited. Furthermore, preloading a large section of a database would saturate the network with unnecessary traffic, depriving other clients of their service. To avoid

this bandwidth over-utilization, CyberWalk employs cache replacement policy to retain only frequently accessed objects in the cache and prefetching mechanisms to prefetch only potentially visible objects in order to reduce access and rendering latency.

In [8], various cache replacement policies have been proposed and their suitabilities in a conventional database system have been examined. Policies such as Least Recently Used (LRU) and Least Reference Density (LRD) are being used widely. These policies are derived from their counterpart in operating systems. In the context of databases, the Most Recently Used (MRU) policy is also occasionally adopted to cater for cyclic data access behavior. 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. A general conclusion on the performance of the replacement policies cannot be made. In practice, replacement policy is often approximated by the LRU policy in conventional caching [3], [10], [23]. In [22], we show that LRU policy is not appropriate in a context where 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. This has been shown to be effective in our preliminary work [7]. Furthermore, we have also noticed in [5] that prefetching could be very beneficial in improving the performance of database applications if the prefetching is performed intelligently. In [14], prefetching has been adopted to reduce the expected access time to a sequence of web page accesses. Experimental results have demonstrated the benefits brought about by prefetching.

III. Design of CyberWalk

In CyberWalk, each virtual object is stored in the database server at its maximum resolution in the form of a progressive mesh. The use of such a multiresolution method allows the database server to transmit an object model at a resolution just high enough for rendering. This could save scarce Internet bandwidth from transmitting extra details of distant objects. The resolution of an object model at a particular point in time is determined by its distance from the viewer and its angular distance from the viewer's line of sight. However, it would be expensive to determine the resolution of every object within the virtual world whenever the viewer makes a movement. In practice, the number of objects that are visible to a viewer is limited, depending on the depth of sight of the viewer. Hence, we could further minimize the number of objects needed to be handled in each frame by associating the concept of depth of sight with each viewer. Objects beyond the depth of sight of a viewer will not need to be transmitted from the server to the client, nor do they need to be rendered. In CyberWalk, the depth of sight of a viewer is modeled by the idea of *object scope* and *viewer scope*. To further reduce the dependency on the Internet, to lower the transmission delay, and to support disconnected operation, we have incorporated a caching mechanism to retain objects of high affinity and a prefetching mechanism to predict those that will likely be accessed in the near future.

CyberWalk possesses several advantages over previous approaches. First, previous approaches use an LoD method [21] for model transmission. Redundant information will be sent over 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, the depth of sight of the viewer, and the resolution of the viewing device, allowing the object models to be rendered at the lowest cost. Third, our caching mechanism differs from conventional caching mechanisms [3], [10], [23] in that objects could be cached at multiple granularities. Cache replacement is also based on object access patterns rather than on the 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.

A. Object Scope

To minimize the amount of data needed to be handled, most existing methods consider only the *area of interest* (AOI) of a viewer [9], [17], [21]. If an object falls inside the AOI of a 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 object size. Hence, a mountain located just outside the AOI of a viewer may still be visible to the viewer, but is considered as invisible, while a tiny object such as an insect located just inside the AOI of a viewer is unlikely to be visible to the viewer, but is considered for visibility. The former situation may result in a sudden appearance of large objects, 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 *viewer scope* and *object scope*. We denote the viewer scope for viewer V by $\bigcirc_V$ and the object scope for object o by $\bigcirc_o$. A viewer scope is similar to AOI. It indicates the depth of sight of a viewer, i.e., how far the viewer can see. A viewer with a good eyesight 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 an object can be seen. A large object has a larger scope and a small object has a smaller scope. In general, 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 in the virtual environment. This approach is somewhat similar to the one proposed by [11]. In CyberWalk, we define a scope as a circular region. Therefore, each scope (object or viewer) is characterized by a radius.

An object may be visible to a particular viewer only when its scope overlaps with the viewer scope. When the two scopes

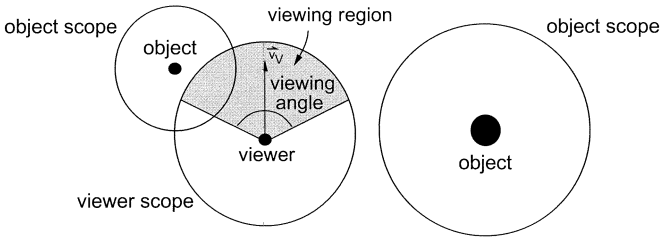


Fig. 2. Interaction between the viewer and the objects in the virtual environment.

overlap, the object's distance from the viewer and the object's angular distance from the viewer's viewing direction are used to determine the resolution, called the *optimal resolution*, at which the object should be rendered. If an 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 an 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 [6]. Such perceived image quality is called *visual perception*. The interaction between a viewer and the virtual environment is illustrated in Fig. 2. In addition to the viewer scope, each viewer V is also associated with a *viewing direction* $\vec{v}_V$, a location loc_V, and a viewing region. The viewing direction defines the viewer's line of sight. Given the location of a viewer, all virtual objects whose object scopes intersect with the viewer scope are potentially visible to the viewer. These objects will be associated with the highest priority for caching in the client's memory and we refer to them as *cachable objects*. Even though some objects may be located behind the viewer, the viewer should be able to see them within a very short moment simply by a brief rotation. The viewing region is a sub-space of the viewer scope and is captured by a *viewing angle*. All cachable objects within the viewing region are considered for rendering at their optimal resolution, and we refer to them as *renderable objects*.

B. The Optimal Resolution

The optimal resolution of an object model can be determined according to the *visual importance* of the object to the viewer. In [16], we have identified several factors that may affect the visual importance of an object. Those factors can all be considered here. However, for the sake of clarity, we only consider two of those factors, which are relevant to the context of CyberWalk. 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 one is the line of sight factor. 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 [19], [26]. Degradation of peripheral visual details can improve rendering performance and reduce perceptual impact. In our implementation, we assume that the viewer's line of sight is at the center of the screen.

Fig. 3 depicts visual importance I_o of object o to viewer V. D_o is the current distance between o and V, while $D_{o,\max}$ is the distance between them when their scopes just overlap. Since a scope is defined as a circular region, $D_{o,\max}$ is the sum of the radii of the two scopes. The angular distance of o from the viewer's line of sight, $\vec{v}_V$, is denoted by θ_o, where $-\pi \le \theta_o \le \pi$. I_o can be defined as

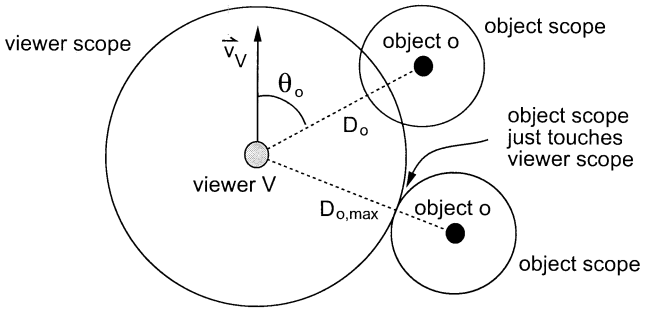


Fig. 3. Visual importance of an object to a viewer.

$$I_o = \left(\frac{D_{o,\max} - D_o}{D_{o,\max}}\right)^2 e^{-K_o|\theta_o|},\ 0 \le D_o \le D_{o,\max}. \quad (1)$$

The first term models the effect of the distance between o and V on I_o. Higher D_o results in a lower I_o. The second term models the effect of the angular distance of o from the viewer's line of sight on I_o. Higher θ_o results in a lower I_o. Since this relationship is not a linear one, a constant K_o is introduced for adjusting the decrement rate of the model resolution due to the increase in θ_o.

The value of K_o in (1) must be chosen carefully because it will affect the performance of visual importance estimation and hence, caching and prefetching performance. We determine here the value of K_o for an optimal visual importance distribution. Equation (1) can be rewritten as follows:

$$I_o = (1-r)^2 e^{-K_o|\theta_o|},\ \text{where}\ r = \frac{D_o}{D_{o,\max}}. \quad (2)$$

Let $\hat{\theta}_v$ be the angular cutoff point where the visual importance would be assumed to be effectively reduced to zero, Φ_v be the volume of I_o within $\hat{\theta}_v$, and Φ_o be the volume of I_o beyond $\hat{\theta}_v$. We now try to find K_o such that Φ_v is maximized:

$$\boxed{\max_{K_0}\left(\int\int \Phi_v dx - \int\int \Phi_o dx\right)} \quad (3)$$

$$\int\int \Phi_v dx = \int_0^1 \int_0^{\hat{\theta}_v} r*(1-r)^2 e^{-k_o|\theta_0|} dr\, d\theta_o = -\frac{1}{12K_o}(e^{-k_o|\hat{\theta}_v|} - 1) \quad (4)$$

$$\int\int \Phi_o dx = \int_0^1 \int_{\hat{\theta}_v}^{\pi} r*(1-r)^2 e^{-k_o|\theta_0|} dr\, d\theta_o = -\frac{1}{12K_o}(e^{-K_o\pi} e^{-k_o|\hat{\theta}_v|}). \quad (5)$$

Substitute (4) and (5) into (3), and perform a differentiation to establish $f'(K_o) = 0$:

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

Solving for $f'(K_o) = 0$ with $\hat{\theta}_v = 120°$, we get $K_o = 1.5317$ such that the visual importance for objects within the stereo vision will be optimal. The perspective view, top view,

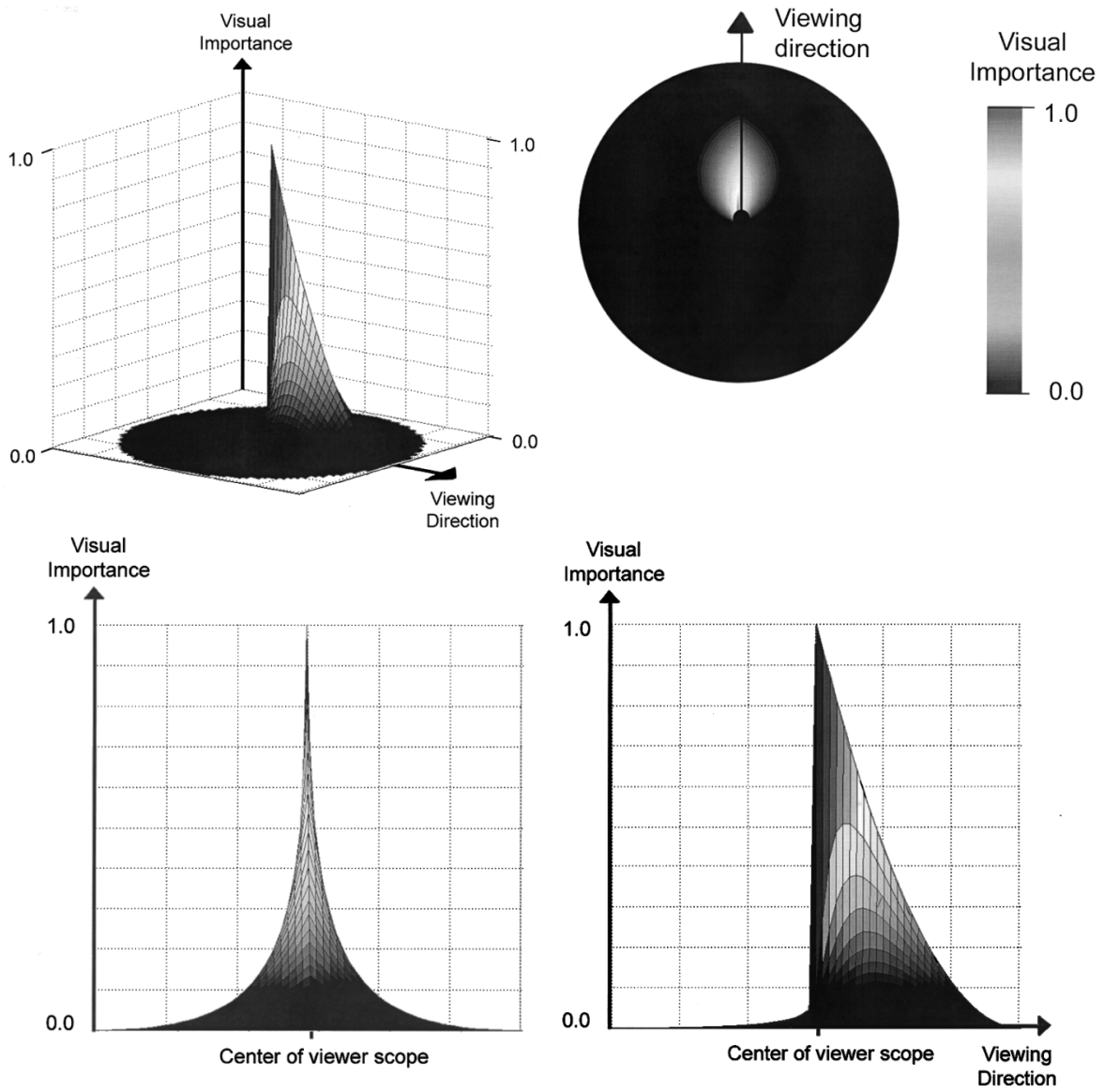


Fig. 4. Visual importance distribution.

front view and side view of the visual importance distribution are illustrated in Fig. 4.

In CyberWalk, visual importance I_o of object o is used to determine the optimal resolution of its model. In effect, I_o indicates the percentage of progressive records of o required to be rendered in addition to its base mesh. During the walkthrough, we continuously determine those cachable objects. When an object scope touches the perimeter of the viewer scope, I_o will be equal to 0 and the optimal resolution 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. Extra progressive records are then transmitted to the client if they are not already available from the local cache, so that the resolution of the object model can be increased to match with its optimal resolution.

C. The Cache Model

At the start of the walkthrough, the locations of all objects in the virtual environment and the radii of their object scopes are sent to the client machine C. Depending on the sizes of the virtual environment and the local cache, we may also preload the base meshes of all or some of the objects in the environment. Since the size of a base mesh can be as small as 0.1% of that of the full resolution model, preloading the base meshes may reduce the system latency at the cost of only slightly increased preloading time.

At the client, the size of the viewer scope and the cachable objects are determined. The cachable object models at their optimal resolutions are then transmitted to C. As the viewer moves within the environment, C continuously determines the cachable objects at each step. Each cached object o is associated with a resolution $\mathcal{R}_o^*$ indicating its current highest possible resolution available for rendering. C will then generate a *request list* for the cachable objects not in the local cache. Each entry of the request list contains id_o (ID of o), $\mathcal{R}_o^*$, and $\mathcal{R}_o$ (expected optimal resolution of o). The server then transmits the outstanding progressive records and/or base meshes to C, with renderable objects transmitted before other cachable objects.

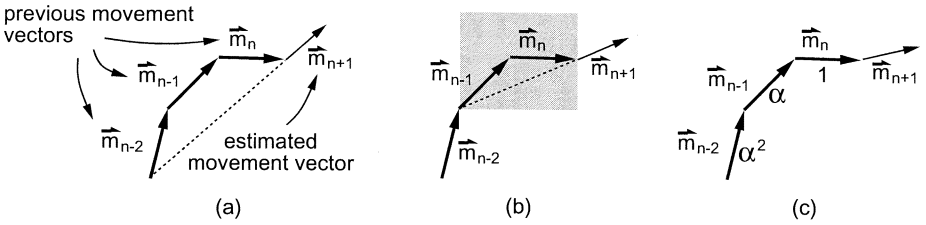


Fig. 5. Prediction of moving direction using (a) mean, (b) window, and (c) EWMA.

Models received by C will be cached in the local storage. If the storage is exhausted, C will employ a replacement policy to identify victim objects to be discarded. An *access score* is determined for each object indicating the prediction of its future access affinity. An object will be cached if it has a higher score. Some progressive records of the cached object with the lowest score will be reclaimed, i.e., CyberWalk will try to decrease the resolution of cached objects with lowest scores. It also attempts to further improve the performance by prefetching objects likely to be accessed in the future, by predicting the next location and viewing direction of the viewer based on his/her past movement profile.

D. Multiresolution Replacement Scheme

CyberWalk employs the *Most Required Movement* (MRM) replacement technique. Normally, the farther an object is from the viewer, the longer it will take for the viewer to move to view the object in greater detail. Similarly, the larger the angular distance of an object is from the viewer's line of sight, the longer it will take for the viewer to rotate to view the object directly in the front. Both lead to a lower value in caching the object. Simulated experiments have shown that such a replacement scheme outperforms traditional LRU replacement scheme [7]. In this work, we have incorporated the idea of the object scope and the viewer scope for visibility determination.

Among all possible formulae for calculating the access score of an object, we prefer a simple one to model the line of sight and the distance factors. The simplest formula is a weighted average of the input factors. To make these factors comparable, we normalize them with their reference values. The distance factor is normalized by the sum of the radii of the viewer scope and the object scope ($D_{o,\max}$). The line of sight factor is normalized by the maximum angular distance from the viewer's line of sight (π). We have developed a formula for estimating the access score with a single parameter ω ($0 \leq \omega \leq 1$). Using the notations in Fig. 3, the access score of object o is defined as

$$S_o = \omega\left(1 - \frac{D_o}{D_{o,\max}}\right) + (1-\omega)\left(1 - \frac{|\theta_o|}{\pi}\right). \quad (6)$$

When the object with the lowest access score is selected for replacement, we first remove the extra progressive records of the object to reduce its resolution to the optimal resolution. If there is still not enough room, the object with the next lowest access score will be taken and this process will be iterated. When there is still not enough room even after all cached objects have been reduced to their optimal resolutions, all progressive records of the object with the lowest score will be removed, leaving only its base mesh. Again, this process will be iterated. Finally, the base mesh of the objects with the lowest scores will be removed.

This multiresolution replacement scheme tries its best to keep at least the base meshes of the cached objects in the client's cache storage. This provides the 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 visible at a low resolution.

E. Prefetching Mechanism

To enable prefetching, the client maintains a profile of the viewer, containing the list of historical *movement vectors*, $\{\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_{n-1}\}$. Each vector is calculated from the viewer's moving direction and location at a particular time. When the viewer moves to a new location loc_n with a new orientation, the n th movement vector $\vec{m}_n$ is determined. The client will then attempt to predict the $(n+1)$ th movement vector as $\hat{m}_{n+1}$ and request for the transmission of the objects that would become cachable if the viewer 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.

Possible prediction schemes include mean, window and EWMA, as shown in Fig. 5. In the mean scheme, the next movement vector $\hat{m}_{n+1}$ is predicted as the average of the previous n movement vectors, as depicted in Fig. 5(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 Fig. 5(b), showing a window of size $W = 2$. A problem for the window scheme is that all movement vectors within the window have equal effect on the prediction.

To better approximate the real movement, and to adapt quickly to changes in the viewer's moving patterns, CyberWalk employs the EWMA (exponentially weighted moving average) approach to predict the next location of the viewer. It assigns exponentially decreasing weights to each previous movement vector $\vec{m}_i$. With parameter α, the most recent vector will receive a weight of 1, the previous α, the next previous α^2, and so on. This idea is depicted in Fig. 5(c), indicating the predicted moving direction. The new $(n+1)$ th movement vector is estimated as $\hat{m}_{n+1} = \alpha\hat{m}_n + (1-\alpha)\vec{m}_n$.

EWMA has been shown to be quite effective in predicting access probabilities of data items in database applications by adapting rather quickly to changes of access patterns [22]. However, it might not perform as satisfactorily in this new context of predicting the next viewer location. This is because the access

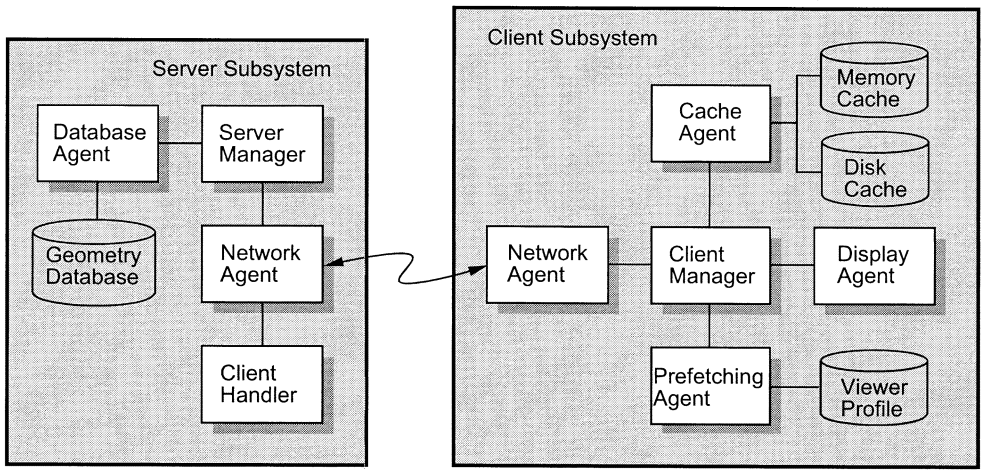


Fig. 6. Architecture of CyberWalk.

probability of a data item is bounded between 0 and 1. EWMA is trying to incorporate the effect of the change into the new estimate and normally, the estimation error would not diverge. In the new context here, we are using EWMA to predict a vector, whose direction is an angle with an unbounded range, 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 "nonstationary" changes. We need to explicitly correct the prediction with adjustment from residuals or error predictions. The residual in predication of $\vec{m}_n$ is $\vec{e}_n = \hat{m}_n - \vec{m}_n$. We consider the angle between $\hat{m}_n$ and $\vec{m}_n$, denoted as $\phi_n = \arg(\hat{m}_n) - \arg(\vec{m}_n)$, where $\arg(\vec{m})$ is the argument of vector $\vec{m}$ in a complex plane. $\vec{m}_n$ can be predicted by rotating $\hat{m}_n$ through an angle of $-\phi_n$, i.e., a multiplication by $e^{-\imath\phi_n}$. Since we do not really know ϕ_{n+1} when we predict $\hat{m}_{n+1}$, we must try to predict ϕ_{n+1} as well. There can be different ways of predicting ϕ_{n+1} from the previous values of ϕ_i. Again, CyberWalk uses EWMA to compute the prediction $\hat{\phi}_i$ of ϕ_i in each step as we compute $\hat{e}_i$. Thus, $\hat{\phi}_{n+1} = \alpha\hat{\phi}_n + (1-\alpha)\phi_n$, and $\vec{m}_{n+1} = \hat{m}_{n+1}e^{-\imath\hat{\phi}_{n+1}}$. We call this the EWMA-R (EWMA with residual adjustment) scheme, since it involves the adjustment to the direction of each movement.

IV. Implementation of CyberWalk

The architecture of CyberWalk is composed of two main parts, the *Client Subsystem* and the *Server Subsystem*, as shown in Fig. 6. The Server Subsystem consists of four main components. The *Server Manager* serves as the coordinator of all other components at the Server Subsystem and handles all clients' requests. The *Database Agent* maintains the database of the virtual environment. It is also responsible for identifying and sending the relevant object models to the client in the form of progressive meshes, upon receiving requests from it. The *Client Handler* receives requests from each client, processes the requests and sends the requested data back to the client. It is implemented as a separate thread for each connected client to reduce response time. The *Network Agent* handles all communications between each client and the server. It also maintains the connection between them once a connection is established. When a client requests for a connection, the Network Agent creates a separate Client Handler thread to serve the client.

The Client Subsystem consists of five main components. The *Client Manager* serves as the coordinator of all other components at the Client Subsystem. All viewer inputs, such as translation or rotation, are directed to and handled by the Client Manager. The *Cache Agent* controls local caches, including memory cache and disk cache at the client. All geometry data would be cached via the Cache Agent. The agent maintains a *Score Table*, containing the access score of each object in the local caches. The *Prefetching Agent* prefetches objects from the server based on historical movement vectors of the client in the form of a viewer profile. The *Network Agent* supports the communication between a client and the server, maintaining the connection between them once a connection is established. It is implemented as a separate thread under the Client Manager to exploit execution concurrency. The *Display Agent* accepts inputs from the viewer and generates output images for display.

Except for the Display Agent, the prototype is implemented using Java, due to its platform independence nature. The Display Agent is implemented using OpenGL, as it would utilize the underlying rendering capability of the client machine, if available, for better performance.

A. Client–Server Interactions

A connection-oriented protocol is used in the prototype. The connection between a client and the server is maintained once it has been established. This is easily implemented by Java Remote Method Invocation. We decide to implement our own protocol instead of employing the standard HTTP protocol since HTTP requires a new connection to be established between the client and server every time before a data transfer. This would increase the response time of the system.

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 connection request, the server replies to the client with a new ClientID and some startup information, such as the radii of object scopes and the positions of all objects within the virtual environment. As the viewer moves or rotates within the virtual environment, the

client sends an REQ command to the server followed by a request list, requesting for the outstanding progressive records and/or base meshes of the cachable objects. If prefetching is used, the IDs of the predicted objects would also be included in the list. In return, the server will send an ACK signal to the client followed by the list of requested records and meshes.

Note that there is a slight variation here from our previous work [6] in the way we distribute the tasks between a client and the server. In our previous work, we did not want to make the assumption that the viewer had a fast client machine. Hence, the server was responsible for determining the cachable objects and running the prediction algorithm. When a viewer moved, the client first sent its new location to the server. The server would then send a list of IDs of the cachable objects and their optimal resolutions to the client. Upon receiving the list, the client would determine and send the request list of outstanding progressive records/base meshes of those cachable objects. Based on this list, the server would forward the outstanding data to the client. Although this approach requires minimal computation at the client machine, it increases the network traffic and causes two roundtrip delays. The delay becomes serious as the transmission distance between the client and the server increases. In this implementation, we have decided to offload the tasks to the client machine, which is becoming more and more powerful in recent years. This causes only a single roundtrip delay.

B. Database Structure and Organization

In order to provide fast response to client requests, virtual objects must be organized and maintained in the server in a way that facilitates efficient retrieval. We store each object in the virtual environment into two database files, the *mesh file* containing the base mesh and the *record file* containing all the progressive records of the object.

When the virtual environment is complex, it will most likely contain a huge number of virtual objects. Identifying the set of cachable objects with respect to the current position of a viewer will be expensive since the Client Manager has to examine all virtual objects to determine if their object scopes overlap with the viewer scope. To reduce the cost of scope comparison, we have adopted a simple indexing method. We divide the virtual environment into two-dimensional (2-D) square cells. Each cell is a pointer to a list of object IDs. These IDs indicate objects whose scopes overlap with the cell. To determine the cachable objects, we may simply identify the cells that the viewer scope overlaps with the objects and the objects IDs found in these cells will simply indicate the cachable objects.

C. Progressive Mesh Transmission

In CyberWalk, a client machine would continuously request the server for objects that are not available in local cache, as the viewer moves around the virtual environment. These requested objects from the server can be classified into three disjointed groups, the renderable objects, the nonrenderable objects and the prefetch objects. The nonrenderable objects are objects which are cachable but not renderable, i.e., cachable objects which are outside the viewing region. The prefetch objects are objects which are expected to become cachable in the near future as predicted by the prefetching mechanism. These three groups define a total order of transmission: renderable $\prec$ nonrenderable $\prec$ prefetch, where $\prec$ denotes the classic *precedes* relation [1]. In other words, CyberWalk guarantees that progressive meshes of the renderable objects will be transmitted before those of the nonrenderable objects, which in turn, will be transmitted before those of the prefetch objects.

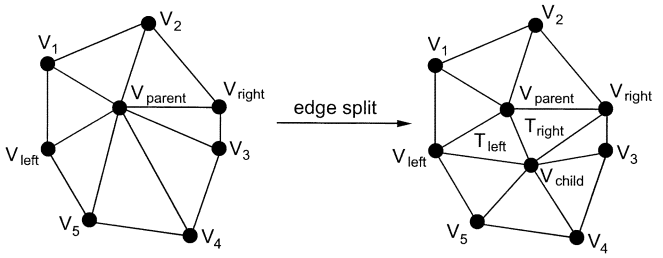


Fig. 7. Example of an edge split operation.

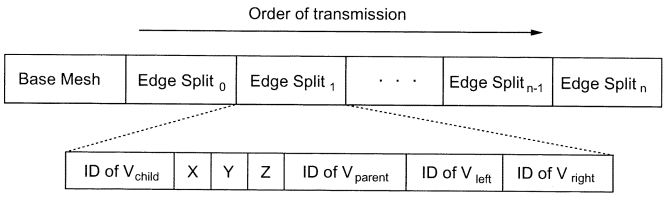


Fig. 8. Data structure of the progressive mesh for transmission.

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. Progressive records are then transmitted in order. Each record stores information of an edge split, thereby increasing the resolution of the object model by a small amount. Fig. 7 shows an example of such an operation. After an edge split, one vertex, i.e., V_{child}, one edge, i.e., $\langle V_{\text{parent}} V_{\text{child}} \rangle$, and two triangles, i.e., T_{left} and T_{right}, are inserted into the model. (Note that at the surface boundary, only one triangle is inserted into the model.)

We use Fig. 7 as an example to explain the data structure of a progressive record shown in Fig. 8. First, we need to store the $\langle x, y, z \rangle$ coordinates of the child vertex V_{child} to be inserted into the model and the ID of the parent vertex V_{parent}. V_{parent} will join with V_{child} to form the inserted edge. We also need to include the IDs of two vertices, V_{left} and V_{right}. They help to identify the locations at which the two new triangles are to be inserted. The two triangles are defined as $T_{\text{left}} = \langle V_{\text{parent}}, V_{\text{child}}, V_{\text{left}} \rangle$ and $T_{\text{right}} = \langle V_{\text{child}}, V_{\text{parent}}, V_{\text{right}} \rangle$. When inserting V_{child}, some neighboring vertices of V_{parent} will move to become the neighboring vertices of V_{child}. To be able to divide the neighboring vertices during an edge split, we maintain a linked list for each vertex of the object model at the client. Each linked list points to all immediate neighboring vertices of the parent vertex, ordered in a clockwise direction. Given the IDs of vertices, V_{left} and V_{right}, the linked list is automatically divided into two segments. As an example, V_{left}, V_1, V_2, V_{right}, and V_{child} will become the neighboring vertices of V_{parent} while V_{right}, V_3, V_4, V_5, V_{left}, and V_{parent} will become the neighboring vertices of V_{child} after the edge split.

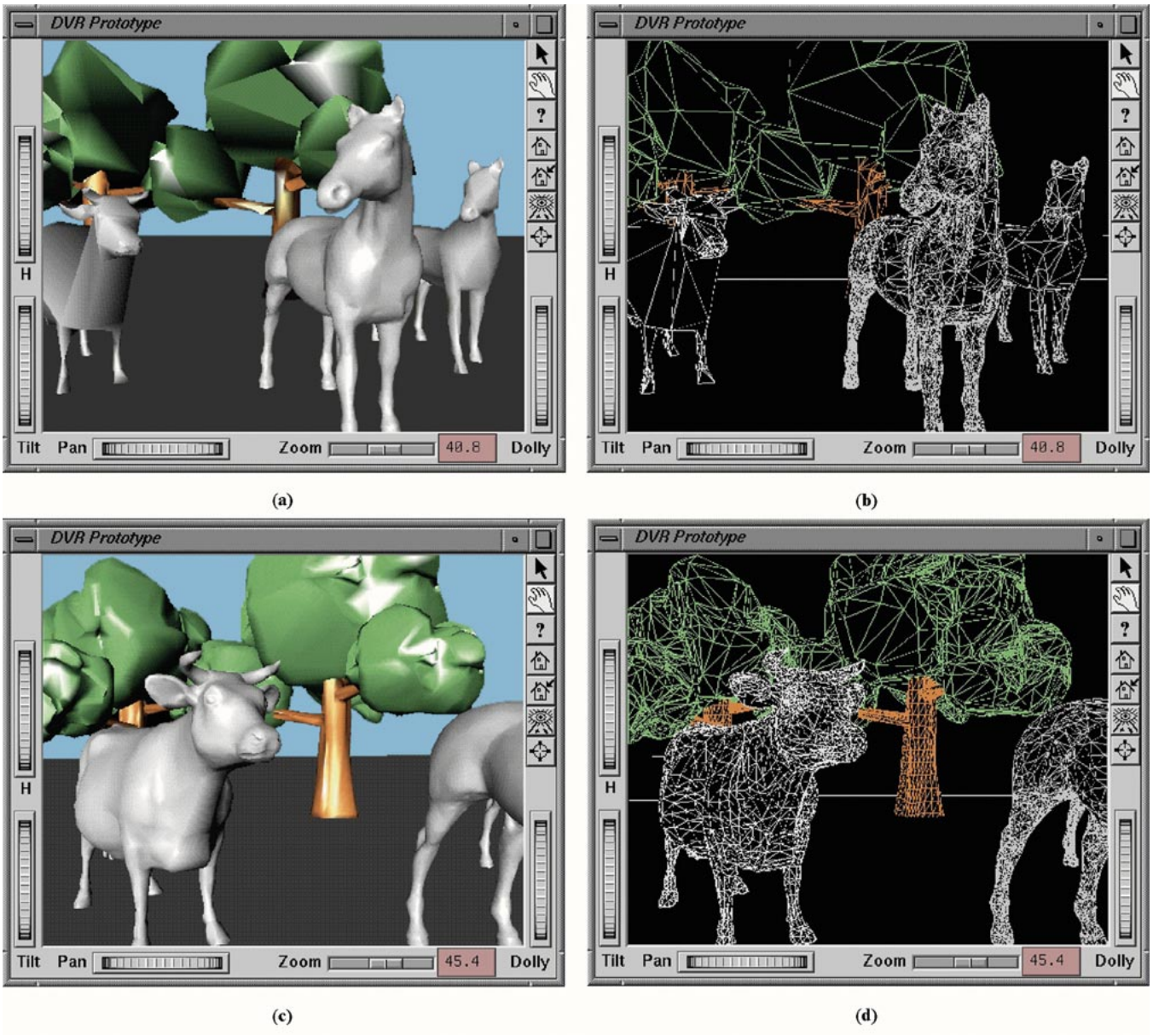


Fig. 9. Sample session with the prototype.

Fig. 9 illustrates a sample session of the prototype. Fig. 9(a) shows a scene in the virtual environment. Fig. 9(b) illustrates the corresponding model resolutions, with distant objects represented by lower resolution models. Fig. 9(c) and (d) show an increase in the resolution of the cow and the tree objects when the viewer moves forward.

V. Results and Discussions

We have conducted extensive experiments to quantify the performance of our multiresolution scheme with MRM cache replacement 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. Due to page limit, we present only a representative subset of experiments in this section. We 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: $1 - (B_o - B_o^*/B_o)^3$, 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 of visual perception is based on

TABLE I
PARAMETERS LISTING FOR EXPERIMENTS

Notation	Description
n	Number of virtual objects
N	Size of storage cache (percentage of database)
ω	Parameter for determining access score (fixed at 0.5)
W	Window size
α	Exponentially decreasing weight (fixed at 0.5)

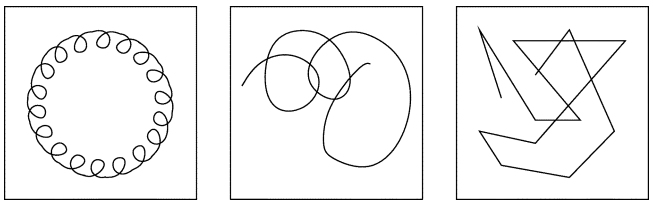


Fig. 10. Moving patterns: (a) CP, (b) CCP, and (c) RW.

the fact that when a viewer makes a move in the virtual environment, a high visual perception would be experienced if all renderable objects could be seen instantaneously (from the local cache) at a resolution close to the optimal one. The visual perception would still be considered as good or acceptable if at a minimum, the base meshes of the models are available locally. On the other hand, if some or all of the renderable objects are not visible until after a long period of waiting time, a low visual perception is experienced.

A. Experimental Environment

The set of parameters used in our experiments are listed in Table I. We focus on a single server and a single client in our study here. The size of the virtual environment is set to 2000×2000 square units. n virtual objects are distributed uniformly among the square units, each containing an average of $n/(4 * 10^6)$ objects. The viewer's viewing angle is set to 120°. The radius of the viewer scope is set to ten units. The cache size is set 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. In the Window scheme, we experimented with four Window sizes, $W = 1, 3, 5, 7$. 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 experimented with three moving patterns of a viewer, as depicted in Fig. 10. 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 15 units along the viewing direction, and then a rotation of the viewing direction by 12°. At each step, the viewer rotates his/her head by $\pm 20°$. 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 10°, after every four 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.

B. Experiments From Simulation

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

The measurements of the metrics are depicted in Fig. 11. We observe that even without prefetching, the caching mechanism performs reasonably well, achieving a hit ratio ranging from 79% to 83%. With prefetching, the hit ratios could be improved by up to 6%. We observe that mean is not very effective in predicting future movements, with performance similar to that of 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-NR 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 exhibits 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.

C. Experiments From Prototype

In our prototype experiments, the server runs on a Sun UltraSparc 2 workstation. The client runs on an SGI Indigo^2 workstation with 64MB RAM. We study the behavior of the caching and prefetching mechanisms under a real system.

Experiment 1: Our first set of experiments resembles our simulated experiment presented in Section V-B. However, since running experiments 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 200 KB. All other parameters remain unchanged. We hope to be able to compare the general behavior of the mechanism under a real system with simulated behavior with this adjustment.

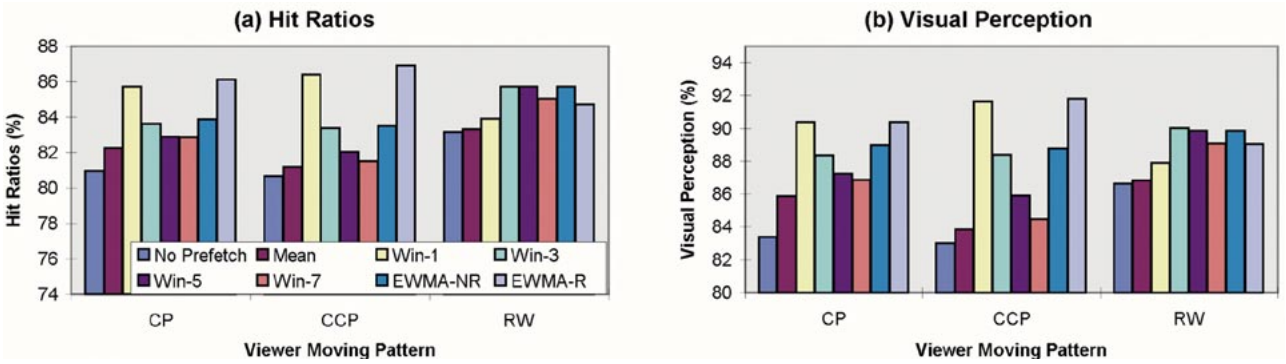


Fig. 11. Performance from simulation.

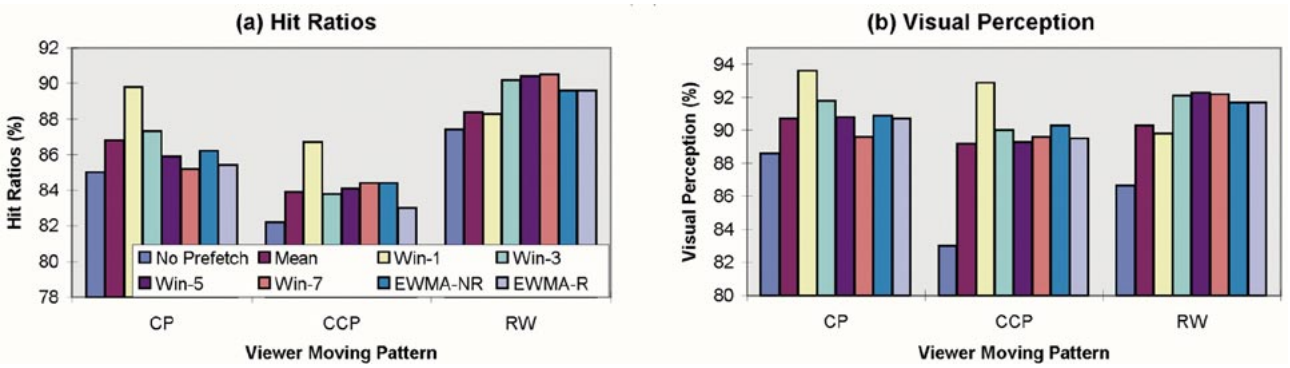


Fig. 12. Performance from experiment 1.

The measurements of the metrics are depicted in Fig. 12. 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 percent in the experiment, across all prefetching schemes. For other moving patterns, the improvement in hit ratios from EWMA seems to be smaller than that brought about by simulation. This is perhaps due to the object distribution in the experimental environment. We are currently investigating this issue. We hope to be able to report our findings in the future. The impact on visual perception is similar to that on hit ratios, but at a smaller scale.

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 of the effect of cache size, we further measure the average *response time* and *latency time* of the prototype. 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 all 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. Fig. 13 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, the response and latency times of the application are already reduced 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. It is simply because a larger cache 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 satisfactorily, yielding similar performance as in the simulation.

Response and latency times are not as stable as hit ratio and visual perception, due to their heavy dependency on the available network bandwidth when the prototype is running. However, a general observation can still be drawn about their relative performance. With caching, latency time is around 0.25 s. 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 50% higher than the latency, i.e., between 0.3 s to 0.4 s with a cache size of 1% for all movement patterns, and higher with a smaller cache size, as depicted in Fig. 13(c). However, when compared with no caching, caching alone could improve the response and latency times of the walkthrough application by quite a few times as shown in Fig. 13(c) and (d). Prefetching also leads to improvement in response times. Finally, with an increasing cache size, improvement in response and latency times is also observed.

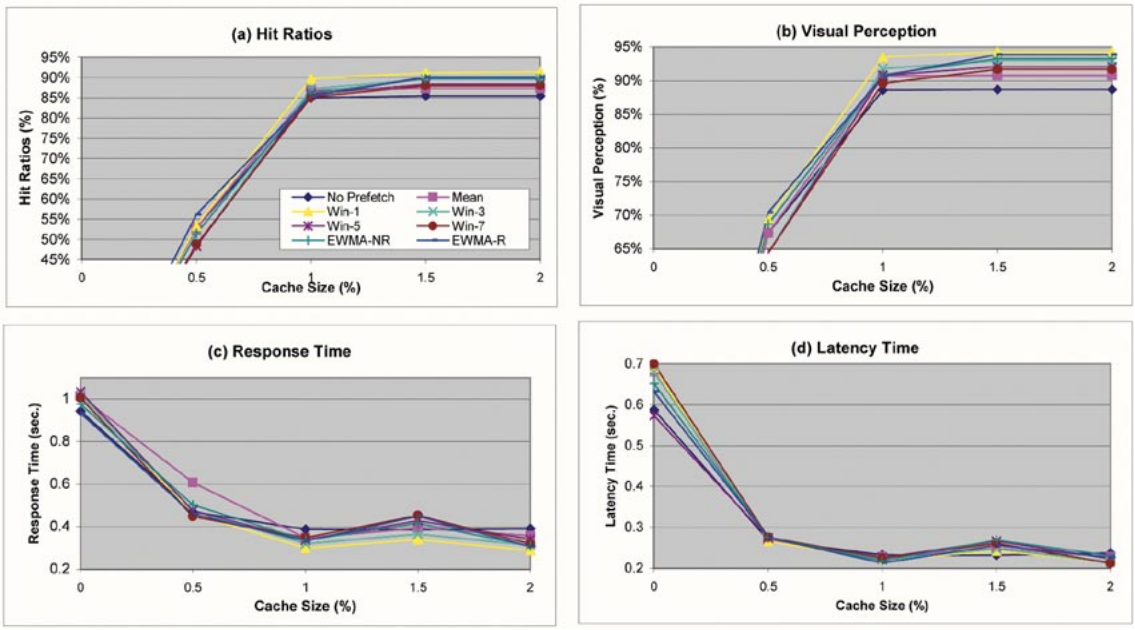


Fig. 13. Performance from Experiment 2.

VI. Conclusions

In this paper, we have described the design and implementation of CyberWalk. We have pointed out the technical challenges that need to be addressed in order to support on-demand transmission of virtual environments without sacrificing the performance of distributed virtual walkthrough applications. As an alternative to improving 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 that may be accessed in the near future. The prediction is based on the semantics of virtual walkthrough application. These caching and prefetching mechanisms attempt to make the objects available at the client machines as soon as they become visible to the users. As demonstrated in our experiments, these mechanisms significantly reduce both response time and latency time of the system.

We are currently extending our studies along several dimensions. We are investigating other means of prefetching virtual objects and comparing their effectiveness with EWMA scheme. We are studying the effectiveness of prioritizing objects for transmission at record level rather than object level on visual perception. We are investigating 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.

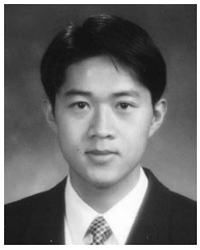
Acknowledgment

The authors would like to thank B. Ng for helping to verify some of the experimental results. We would also like to thank the anonymous reviewers and the associate editor of this paper for their helpful suggestions.

References

[1] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems* . Norwell, MA: Addison-Wesley, 1987.
[2] J. Calvin, A. Dicken, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The SIMNET virtual world architecture," in *Proc. IEEE VRAIS*, 1993, pp. 450–455.
[3] M. Carey, M. Franklin, M. Livny, and E. Shekita, "Data caching tradeoffs in client-server DBMS architectures," in *Proc. ACM SIGMOD.*, 1991, pp. 357–366.
[4] C. Carlsson and O. Hagsand, "DIVE—A multi-user virtual reality system," in *Proc. IEEE VRAIS*, 1993, pp. 394–400.
[5] B. Y. L. Chan, H. V. Leong, A. Si, and K. F. Wong, "MODEC: A multigranularity mobile object-oriented database caching mechanism, prototype and performance," *J. Distrib. Parallel Databases*, vol. 7, no. 3, pp. 343–372, July 1999.
[6] 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 *Proc. ACM Multimedia*, Sept. 1998.
[7] J. Chim, R. W. H. Lau, H. V. Leong, and A. Si, "Multi-resolution cache management in digital virtual library," in *Proc. IEEE Advances in Digital Libraries Conf.*, Apr. 1998, pp. 66–75.
[8] W. Effelsberg and T. Haerder, "Principles of database buffer management," *ACM Trans. Database Syst.*, pp. 560–595, Dec. 1984.
[9] J. Falby, M. Zyda, D. Pratt, and R. Mackey, "NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation," *Comput. Graph.*, vol. 17, no. 1, pp. 65–69, 1993.
[10] M. Franklin, M. Carey, and M. Livny, "Global memory management in client-server DBMS architectures," in *Proc. VLDB*, 1992, pp. 596–609.
[11] C. Greenhalgh and S. Benford, "MASSIVE: A distributed virtual reality system incorporating spatial trading," in *Proc. Int. Conf. Distributed Computing Systems*, 1995, pp. 27–34.
[12] H. Hoppe, "Progressive meshes," in *Proc. ACM SIGGRAPH '96*, Aug. 1996, pp. 99–108.
[13] V. Icsler, R. W. H. Lau, and M. Green, "Real-time multi-resolution modeling for complex virtual environments," in *Proc. ACM VRST*, July 1996, pp. 11–20.
[14] A. Kraiss and G. Weikum, "Integrated document caching and prefetching in storage hierarchies based on Markov-chain predictions," *J. Very Large Database Syst.*, vol. 7, no. 3, pp. 141–162, Aug. 1998.
[15] R. W. H. Lau, M. Green, D. To, and J. Wong, "Real-time continuous multi-resolution method for models of arbitrary topology," *Presence: Teleop. Virtual Environ.*, pp. 22–35, Feb. 1998.

[16] R. W. H. Lau, D. To, and M. Green, "An adaptive multi-resolution modeling technique based on viewing and animation parameters," in *Proc. IEEE VRAIS*, 1997, pp. 20–27.
[17] 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 *Proc. IEEE VRAIS*, Mar. 1995, pp. 2–10.
[18] B. Mannoni, "A virtual museum," *Commun. ACM*, vol. 40, no. 9, pp. 61–62, 1997.
[19] T. Ohshima, H. Yamamoto, and H. Tamura, "Gaze-directed adaptive rendering for interacting with virtual space," in *Proc. IEEE VRAIS*, July 1996, pp. 103–110.
[20] I. Pandzic, T. Capin, E. Lee, N. Thalmann, and D. Thalmann, "A flexible architecture for virtual humans in networked collaborative virtual environments," in *Proc. Eurographics '97*, 1997, pp. 177–188.
[21] D. Schmalstieg and M. Gervautz, "Demand-driven geometry transmission for distributed virtual environments," in *Proc. Eurographics '96*, 1996, pp. 421–432.
[22] A. Si and H. V. Leong, "Adaptive caching and refreshing in mobile databases," *Personal Technol.*, vol. 1, no. 3, pp. 156–170, Sept. 1997.
[23] A. Silberschatz, H. Korth, and S. Sudarshan, *Database System Concepts*. New York: McGraw-Hill, 1996.
[24] G. Singh, L. Serra, W. Png, and H. Ng, "Bricknet: A software toolkit for network-based virtual worlds," *Presence: Teleop. Virtual Environ.*, vol. 3, no. 1, pp. 19–34, 1994.
[25] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation* . Norwell, MA: Addison-Wesley, 1999.
[26] B. Watson, N. Walker, and L. Hodges, "Effectiveness of spatial level of detail degradation in the periphery of head-mounted displays," in *Proc. ACM CHI'96*, April 1996, pp. 227–228.
[27] W. M. R. Wong and R. R. Muntz, "Providing guaranteed quality of service for interactive visualization applications," in *Proc. ACM SIGMETRICS*, June 2000.



Jimmy Chim received both the undergraduate and M.Phil. degrees from the Hong Kong Polytechnic University.

He is currently with the School of Visual Arts in New York, pursuing the M.S. degree in visual arts and computer animation. He has conducted research on the area of multimedia systems, both in performance study and system implementation, publishing several papers in major conferences.

Rynson W. H. Lau (M'88) received a (top) first class honors degree in computer systems engineering in 1988 from the University of Kent, Canterbury, U.K., and the Ph.D. degree in computer graphics in 1992 from the University of Cambridge, U.K.

He is currently an Associate Professor at the City University of Hong Kong. Prior to joining the university in 1998, he taught at the Hong Kong Polytechnic University. From 1992 to 1993, he worked at the University of York, U.K., on a defense project on image processing. His research interests are in computer graphics, virtual reality and multimedia systems.

Dr. Lau is a member of the IEEE Computer Society and the ACM.

Hong Va Leong received the Ph.D. degree from the University of California, Santa Barbara.

He is currently an Associate Professor at the Hong Kong Polytechnic University. He has served on the program committees and organization committees for many international conferences. He is also a reviewer for a number of international journals, including several IEEE Transactions. His research interests are in distributed systems, distributed databases, mobile computing, internet computing, and digital libraries.

Dr. Leong is a member of the ACM and the IEEE Computer Society.

Antonio Si (M'95) received the Ph.D. degree from the University of Southern California, Los Angeles.

He is currently with Oracle Corporation, Redwood Shores, CA. Before joining Oracle, he was an Assistant Professor at the Hong Kong Polytechnic University and a Software Engineer at Sun Microsystems, Inc. in the United States. He has served on the program committees for several international conferences and as external reviewers for a number of international conferences and journals. His research interests are in mobile computing, internet computing, and digital libraries.

Dr. Si is a member of the ACM and IEEE Computer Society.