The Visual Computer

A data management scheme for effective walkthrough in large-scale virtual environments

Tsai-Yen Li,
Wen-Hsiang Hsu

Computer Science Department, National Chengchi University, 64, Sec. 2, Zhih-Nan Rd., Taipei, Taiwan 11623, ROC
E-mail: {li, g8804}@cs.nccu.edu.tw

Published online: 18 November 2004

As the complexity of virtual environments increases, it becomes a critical issue to maintain the quality of a walkthrough experience. In this paper, we propose an effective data management scheme to address this issue in client-server architecture. First, we propose using real-time scene management to manage the computing resources on the client side by reducing the amount of transmitted geometry data. Second, we propose a prioritized most likelihood movement model to prefetch potential future objects based on the user's current motion intention. Lastly, a hybrid coherence cache model is proposed to take advantages of both the temporal and spatial localities of the walkthrough process. We have done extensive experiments to demonstrate how these techniques can improve the effectiveness of walkthrough in a large virtual environment.

Key words: Large-scale virtual environment – Spatial data management – Effective walkthrough – Prefetching – Caching

1 Introduction

Virtual environment (VE) is a form of multimedia presentation that aims to simulate realistic worlds with virtual reality (VR) technologies. As the computer and communication technologies evolve, the applications of virtual environments are also emerging. For example, virtual museums, virtual shopping malls, virtual factories, traditional 3D games such as DOOM and DIABLO, and the popular online 3D games are all example of VE applications.

As the computer technologies are rapidly advancing, user demands for better and more delicate scenes also increase. Consequently, the complexity of a virtual scene is rapidly expanding. Although the computer hardware and software developments on 3D technologies have recently made great process, we think it is difficult to meet user demands for better content. In addition, most users are only equipped with a 2D input device on a regular personal computer. Therefore, the quality of navigation experience is still difficult to control with such computational and input constraints. It is even more challenging for a large virtual environment, such as the virtual factory example in Fig. 1, that usually contains a great number of object models located dispersedly. Most of the current 3D navigation systems partition such a large virtual environment into smaller scenes connected with portals. A scene is loaded only if the viewpoint enters its region through the portal. The user's navigation flow is interrupted since he/she needs to wait for some amount of time until the new scene is loaded.

In this paper, we propose a data management scheme to address the navigation issues in a large virtual environment. First, we developed a real-time scene management method to reduce the amount of retrieved data as well as rendering times. Second, when accessing geometric models of relevant objects, we use a cache mechanism to account for both spatial and temporal localities. Third, we propose a most likelihood movement (MLM) prefetch model that considers the mouse controllability and the existence of obstacles. Finally, we have implemented the front-end and back-end systems for the virtual environment in which we have done extensive experiments to show the effectiveness of such a data management scheme.

2 Related work

The related research addressing the issues of handling large virtual environments started in the

Digital Object Identifier (DOI) 10.1007/s00371-004-0264-0

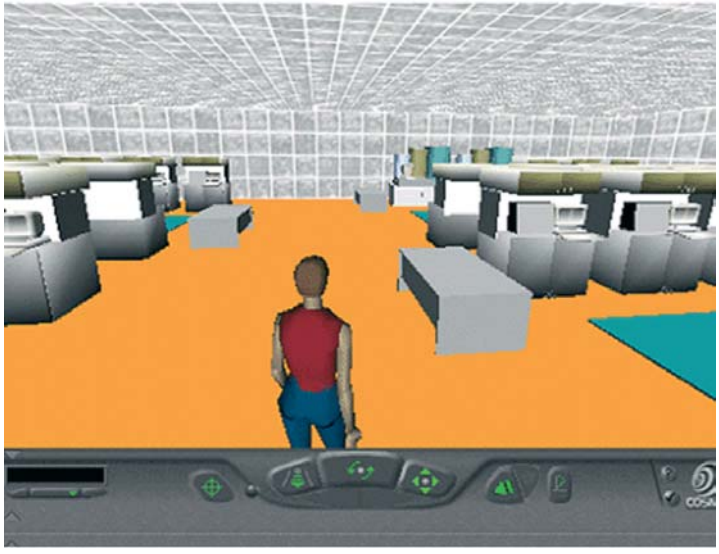


Fig. 1. Virtual factory: an example of a large scale virtual environment

1990s [1, 8] and have had made significant advances since then. According to the problems that have been addressed in the literature, we can classify the problems into two categories: *geometry replication* and *data management*. Geometry replication addresses the problem of efficiently transferring 3D geometry models from the server to the clients, while data management addresses the problem of managing data on the client side in order to increase navigation smoothness.

2.1 Geometry duplication methods

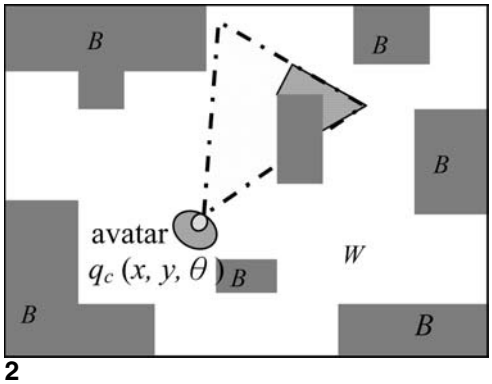
In a navigation system with the client-server architecture, most geometric data are stored on the server side. Therefore, a common problem is how to copy data to the clients efficiently. The simplest method would be that the client preloads all models on the client side before starting the navigation. NPSNET [15] is an example that preloads models before navigation. With such a method, the client side must be willing to wait and have enough memory space.

Instead of transferring all geometry data to the client, some researchers proposed using on-demand transmission, which only transfers the necessary data to the client as they are needed during navigation. Visibility culling [6, 10, 11, 18], level of detail (LOD) [5, 7, 8, 12], and image-based rendering [1] are all examples along this line of research. In addition, other research has used 3D compression techniques [17] to reduce the amount of geometric data for storage and transmission.

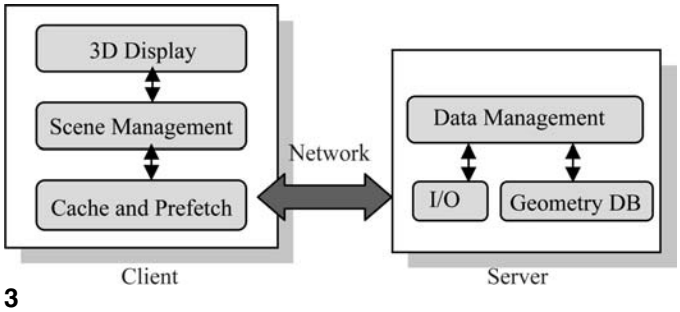
2.2 Data management methods

Navigation in a virtual environment is a sequence of interactions between a user and a virtual environment through user inputs and visual feedback. Data management for navigation in a virtual environment aims to use data processing techniques to make the navigation process smooth and effective. These techniques include caching, prefetching [6], efficient memory management, etc.

In order to have effective prefetching, it is critical to be able to predict a user's intended motion. Some prediction research used statistical methods to increase accuracy [8], but usually required a significant amount of processing time. On the other hand, dead reckoning (DR) [13] is one of the simple and common methods to predict user locations and reduce transmitted data in distributed virtual environments (DVEs). It is a general prediction method that uses extrapolation from the current motion to predict future motions. However, it does not take advantages of the navigation characteristics of VE. Chan et al. [3] proposed to use the characteristics of mouse control to predict user motions. However, since the experi-



2



3

Fig. 2. An example of visible objects from a viewpoint configuration in a sample world
Fig. 3. System architecture

ments are done on paths of artificial patterns instead of real navigation paths, the effects of such a prediction method on realistic navigation paths are difficult to evaluate.

Caching is a common method for reducing data access time. The effectiveness of such a method usually results from selecting a good replacement policy. A good caching model could retain useful old data for future use while a bad caching model could waste cache space and even increase the processing time. Among the commonly used cache replacement policies, the least-recently-used (LRU) method is a popular one that takes advantage of temporal locality. However, the policy might not be very effective for the data access pattern in a virtual environment where strong spatial locality usually exists [4, 8].

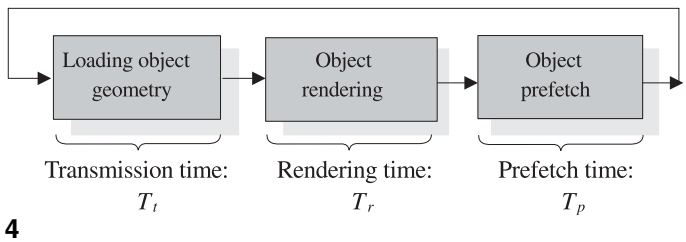
3 System architecture

We first give a brief description of the problem we consider in this paper and then describe the architecture of the proposed system. As depicted in Fig. 2, the Euclidean space where an avatar exists is called a *workspace* and is denoted by W. The set of parameters for describing an avatar is called the *Configuration*. The set of all possible configurations for an avatar is called the *Configuration Space*, or *C-space*. The environmental objects that an avatar cannot penetrate are called obstacles, denoted by B.

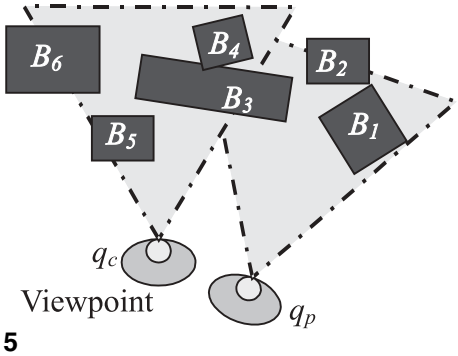
We assume that the virtual environment is static, which means that the obstacles do not move during user navigation. In addition, we assume that a mouse is the only means for the user to control the navigation on a 2D plane in the virtual environment. The current configuration of an avatar, denoted by q_c, contains three parameters: (x, y, θ). x and y represent the current horizontal location of the avatar while θ represents the orientation. Since an avatar usually uses a viewpoint in its local coordinate system to percept the environment, we will assume that the terms of avatar configuration and viewpoint configuration can be used interchangeably in this paper.

The proposed system uses a client-server architecture as depicted in Fig. 3. The client side contains three components: *3D display component*, *scene management component*, and *cache and prefetch component*. The 3D display component is in charge of graphics rendering while the scene management component decides on the objects that need to be fetched for rendering. The cache and prefetch component prepares or acquires necessary objects from the network. On the other hand, the server includes a geometry and scene database, a data management component, and an I/O component. The data management component is responsible for retrieving and assembling geometry data for a given scene, and the I/O component transfers the data to the client via the network.

We further assume that the client system uses a one-processor machine. In such a machine, the up-



4



5

Fig. 4. Frame update procedure
Fig. 5. Maintaining visible objects for real-time scene management

date of each frame is divided into three phases, as shown in Fig. 4. First, the object geometry must be loaded from the server side by the scene management component. The time for transferring this data is called the *transmission time* (T_t). Once the geometric data is ready, the system renders the scene from the current viewpoint. The time for rendering is called *rendering time* (T_r). Finally, the cache and prefetch component uses the remaining *prefetch time* (T_p) to do prefetching. In other words, a frame update consists of T_t, T_r, and T_p. The available time for prefetching depends on the acceptable frame rate, $FR_{acceptable}$, specified by the user. That is, $T_p \leq \left(\frac{1}{FR_{acceptable}}\right) - T_t - T_r$.

3.1 Real-time scene management

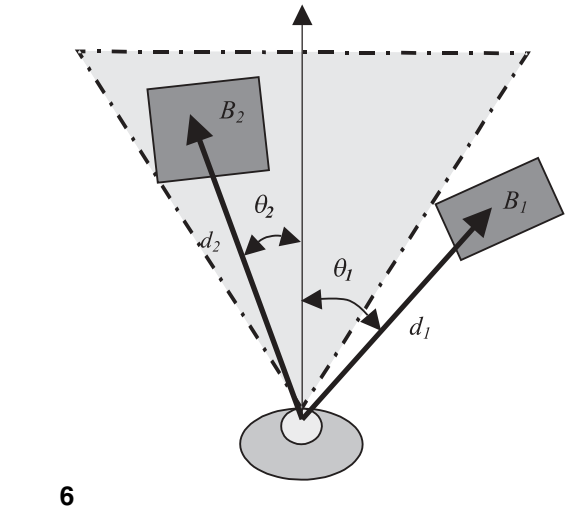
The scene management component uses on-demand transmission to load geometry data as object models are needed. We use a real-time visibility-culling algorithm to determine the visible objects from the current viewpoint configuration. The algorithm is a plane-sweeping algorithm [2] that takes a viewpoint configuration and a 2D geometric description of the objects in the world as input. We assume that the visible region from a given configuration q is denoted by $R(q)$. An object is visible if any portion of its boundary intersects with the visible region. We denote the set of visible objects at q by $V(q)$.

Since only a small number of objects are visible by a viewpoint at a time, the amount of geometry data that need to be retrieved and rendered can be greatly reduced. For example, in Fig. 5, $V(q_p) = \{B_1, B_2, B_3\}$ and $V(q_c) = \{B_3, B_5, B_6\}$. Objects that are visible in the previous viewpoint configuration q_p, but not visible in the current configuration q_c, (for example, $\{B_1, B_2\}$ in Fig. 5) are managed by the cache and prefetch component. The component will determine when to release these objects according to the rules described in the next section. Since objects are handled in an on-demand fashion, a user will be able to perform navigation in a large virtual environment for a long time without exhausting system resources. As the number of visible objects is limited, T_t and T_r can be reduced, which leaves more time for the system to prefetch objects for future use.

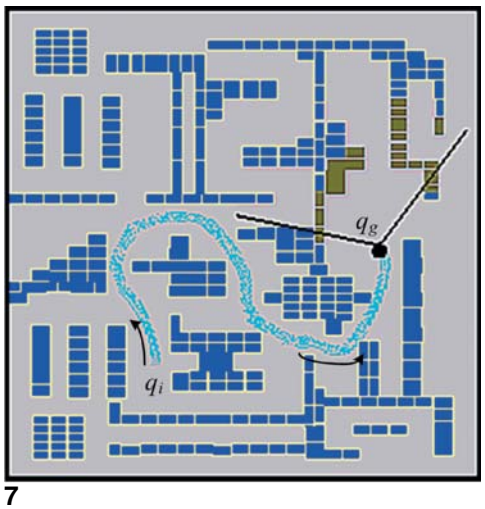
3.2 Hybrid coherence cache model

LRU is a common replacement policy that has been shown to be effective in many applications. When the cache is full, the LRU policy replaces the objects that are not accessed for the longest time. The policy is mainly based on the principle of temporal locality. Objects that are visible during navigation in a virtual environment also have such temporal cohesion. However, spatial locality could be as important for such an application. For example, objects that are closer to the current viewpoint would be more likely to be viewed again in the near future. Consequently, we propose to use a hybrid cache model to perform caching.

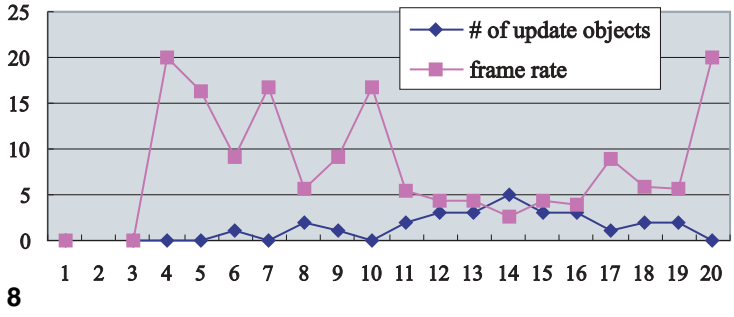
The hybrid cache model determines the order of replacement by both temporal and spatial coherences. Objects that are retrieved at difference frames will be ordered according to the LRU policy. That is, objects that are used the longest time ago will be ordered highest and replaced first. However, for objects that



6



7



8

Fig. 6. Distance and orientation measures computed for the cache model
Fig. 7. An example of navigation path in a typical virtual environment
Fig. 8. Relation between frame rate and number of updated objects

are retrieved on the same time frame, they are ordered by the distance and orientation relative to the viewpoint. More specifically, the distance is computed according to a metric $s(v, o)$ based on the local polar coordinate system of the viewpoint, as defined in Eq. 1 and illustrated in Fig. 6:

$$s(v,o) = w\left(1 - \frac{d(v,o)}{D}\right) + (1-w)\left(1 - \frac{|\theta(v,o)|}{\pi}\right), \quad (1)$$

where v and o denote the avatar and the object, respectively, $d(v, o)$ denotes the distance between them while $\theta(v, o)$ represents the angle between the viewing direction and the vector toward the object.
We have adopted such a hybrid cache model because objects that are more likely to be seen are loaded earlier at the same frame according to the prefetch method described in the next subsection. Therefore, it is important to consider the factor of spatial coherence in addition to the temporal-based policy such as LRU.

3.3 MLM prefetch model

Good prefetch mechanisms make smooth navigation in a retrieve-on-demand VE system. In such a system, a user often experiences sluggishness when the viewpoint encounters abrupt changes since the number of object models that need to be retrieved increases suddenly. For example, for the navigation path in a typical virtual environment shown in Fig. 7, the number of objects that can be seen in each frame may vary significantly. Consequently, the frame rate fluctuates greatly during the navigation as shown in Fig. 8.
A prefetch mechanism retrieves useful models that are likely to be used in the next few frames. Therefore, a good prefetch mechanism should be able to

distribute the load of retrieving object models into earlier frames, which results in smoother navigation experiences. In order to retrieve the right object models in advance, it is critical to have a good prediction method based on user navigation behavior. We propose the MLM model, which aims to obtain a set of possible future viewpoint locations based on the current user navigation behavior.

In the proposed MLM model, since more than one viewpoint configuration is predicted, the concept of object priority becomes important. Most prediction methods used in other related research only predicts the next single configuration and retrieves all object models that can be seen in the configuration. However, whenever the result of prediction is not accurate, retrieving these models only wastes system resources. In order to increase the probability of accurate prediction, we propose predicting a set of possible configurations in a region instead of a single configuration. We order the configurations by giving them priorities computed according to the rules described later. Within the available prediction time, T_p, in a frame update, the system retrieves as many models as possible according to the priorities.

3.3.1 Effects of mouse control on navigation behavior

The effectiveness of a prediction method greatly depends on the test cases. In a lot of other related research, we found that the test cases are either computer-generated paths of certain patterns or smooth paths resembling human motions. A prediction method that works well for these test cases might not yield good results for paths that are sampled from input devices in real navigation. We think the missing factor accounts for human ergonomics, the possible constraints of the input devices, and their effects on the generated paths. Therefore, in this paper, in order to increase the prediction accuracy of the MLM model, we attempt to define the prediction problem in the Cartesian space of the most commonly used input device: the mouse.

As shown in Fig. 9, a vector dragged by a mouse in a VE system is usually decomposed into vertical and horizontal components, which represent linear and angular velocities of the viewpoint, respectively. Transforming the vectors into velocities is usually done via some mapping functions intuitive to human users. Since mouse motion must obey physical rules, the continuity of the mouse trace can be assumed. Therefore, instead of performing prediction in the viewpoint configuration space, we perform prediction on the *Mouse Space*, which is in the plane where a mouse is operated and dragged. The origin of the mouse space is defined at the point where the user presses a button and starts a drag motion. The vertical and horizontal axes in the space are called the *M* and *N* axes, respectively. During a prefetch period, we predict the next possible location of the mouse based on its current motion. The predicted mouse location is then mapped to the corresponding viewpoint configuration that is in turn used to retrieve necessary object models.

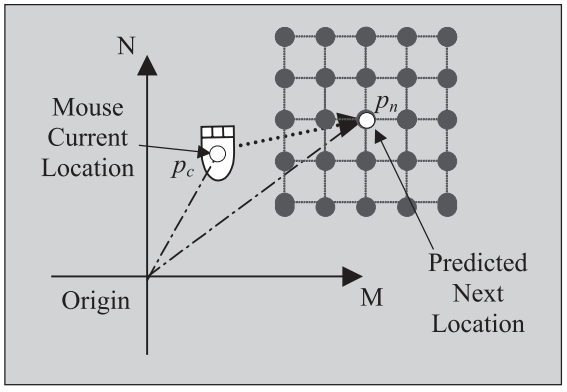
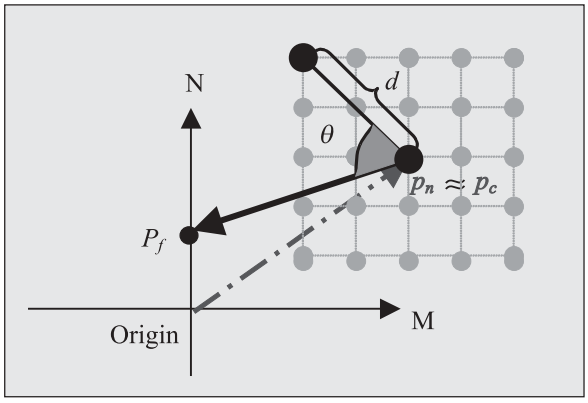


Fig. 9. Predicting the possible set of mouse locations in the mouse space

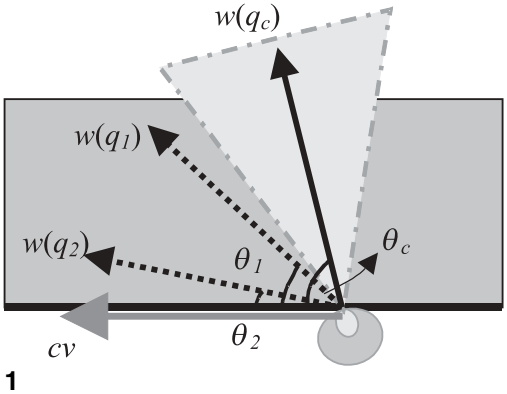
The predicted mouse location, denoted by p_n, is computed according to the current location, p_c, and the commonly used second-order DR method. However, instead of predicting a single location, we define a rectangle grid region centered at the predicted location. Each point in the grid represents a possible mouse location for the next frame. This grid of locations is called the *Possible Set of Configurations for mouse* (Mouse PCS), as shown in Fig. 8. Each mouse location is associated with a priority S_d in the range of (0,1), which is computed according to the following formula:

$$S_d(p_i) = \left(1 - \frac{d(p_n, p_i)}{Q_d}\right), \tag{2}$$

where d is a distance function and Q_d is a constant for quantization. The farther from the center of the region, the lower the priority.



10



11

Fig. 10. Predicting mouse withdraw motion
Fig. 11. Effects of obstacles on the possible set of configurations

Although the DR algorithm works quite effectively in most cases, exceptions occasionally occur when the user's hand withdraws the mouse due to the constraints of the available desktop space or the finger's kinematic limits. To overcome this problem, we first observe that the mouse usually comes to a stop before the withdraw motion starts and the mouse usually returns to a more comfortable location, p_f, along the N axis as shown in Fig. 10. Therefore, when the velocity and acceleration of the mouse are less than some threshold, we apply the following formula to compute the priorities of the grid locations, S_r, in PCS:

$$S_r(p_n, p_i) = w\left(1 - \frac{d(p_n, p_i)}{Q_d}\right) + (1-w)\left(1 - \frac{|\theta(p_n, p_i)|}{\pi}\right). \quad (3)$$

This formula mainly increase the priorities of those grid locations along the potential withdraw path by accounting for the orientation factor.
In sum, according to the velocity and acceleration of the mouse (whether they are above some threshold), S_d or S_r in Eq. 2 and 3 is selected to compute the priorities of the mouse configurations as shown in Eq. 4.

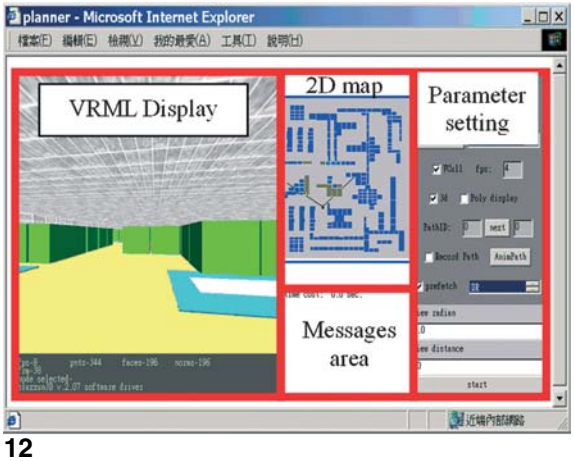
$$S_m(p_n, p_i) = \begin{cases} S_d(p_n, p_i) & \text{if mouse moves} \\ S_r(p_n, p_i) & \text{if mouse stops} \end{cases} \quad (4)$$

3.3.2 Effects of obstacles on navigation behavior

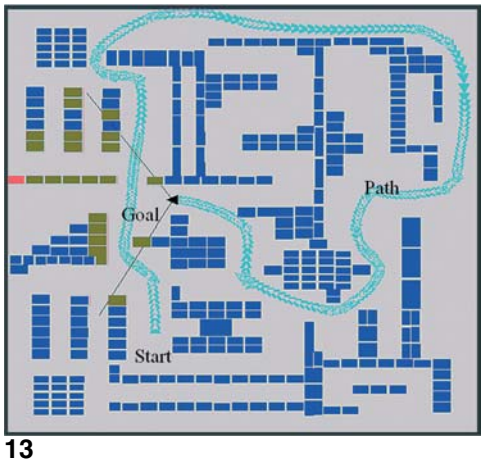
We also consider the effects of environmental obstacles to the next possible viewpoint configuration. A common feature of a virtual environment system is the ability of detecting potential collisions of the viewpoint with environmental obstacles. A viewpoint will usually be forced to move along an obstacle if collisions are detected. Therefore, in the possible set of viewpoint configurations, we would like to account for the influence of the obstacles by giving higher priorities to those configurations that is making a smaller angle with obstacle boundaries.
Assume that the facing direction of a viewpoint configuration q is represented by $w(q)$. Suppose the current configuration is denoted by q_c, and q_1 and q_2 are two other neighboring configurations. The tangential component of $w(q_c)$ along the obstacle boundary is called the *collision edge vector*, or *cv*. The priority of a viewpoint configuration q after being mapped from the mouse space is defined as

$$S_{obs}(q) = \left(1 - \frac{|\theta(w(q), cv)|}{\pi}\right). \quad (5)$$

Therefore, as shown in the example of Fig. 11, since $\theta_2 < \theta_1$, q_2 will be given a higher priority than q_1.



12



13

Fig. 12. Graphical user interface in the navigation experiment
Fig. 13. An example of a large virtual environment and navigation path for experiments

3.3.3 Combining the effects of mouse control and obstacles

In our MLM approach, the object models are retrieved according to the order of configuration priorities, which are in turn determined by the combined influences of mouse control and environment obstacles. We first compute the possible set of mouse locations p_n and their associated distance priorities using Eq. 4. Then we map p_n from the mouse space into the viewpoint configuration to determine if the viewpoint will collide with the obstacles. If there is no collision, the priority will be the distance priority S_m only (Eq. 4). Otherwise, we will take a linear combination of the distance priority $S_m(p)$ and obstacle priorities $S_{obs}(B_{m,c}(p))$ for the final priority of a mouse location p as shown in Eq. 6 below.

$$S(p) = \begin{cases} S_m(p) & \text{if no collision} \\ w \times S_m(p) + (1-w) & \\ \quad \times S_{obs}(B_{m,c}(p)) & \text{if collision} \end{cases} \quad (6)$$

$B_{m,c}(p)$ in Eq. 6 is the mapping function from the mouse space to the viewpoint configuration space. At run time, the priorities of each possible mouse locations and their viewpoint configurations are computed according to the above equations. The weight, w, in Eq. 6 is determined pragmatically. Object models with higher priorities will be retrieved earlier as long as the prefetch time does not exceed the available prefetch time.

4 Experiments

The system proposed in this paper has been fully implemented in the Java language. The architecture of the system follows the common three-tier architecture for Web applications. All the client machine needs is a Web browser, such as Internet Explorer or Netscape Navigator, supporting Java applets. The server side is a regular Web server (an Apache server in our case) and a database server (the MS SQL server in our case). The 3D models of the objects in the virtual environments are stored in individual files in Virtual Reality Modeling Language (VRML).[1]

The computer that has been used in our experiments is a personal computer with an AMD TB 1.2 GHz CPU and 256 MB RAM, and the operating system is MS Windows 2000. An example of the graphical user interface at the client side is shown in Fig. 12. A blaxxun3D[2] VRML browser has been integrated into the system for 3D display. The paths that have been used in our experiments are recorded from navigations controlled by a real user. The same path for a scene is tested on different experimental settings such as different cache and prefetch methods. Fig. 13 shows an example of a large virtual environment of size 64×64 m con-

[1] http://www.web3d.org/technicalinfo/specifications/vrml97/index.html
[2] http://www.blaxxun.com/products/blaxxun3d/

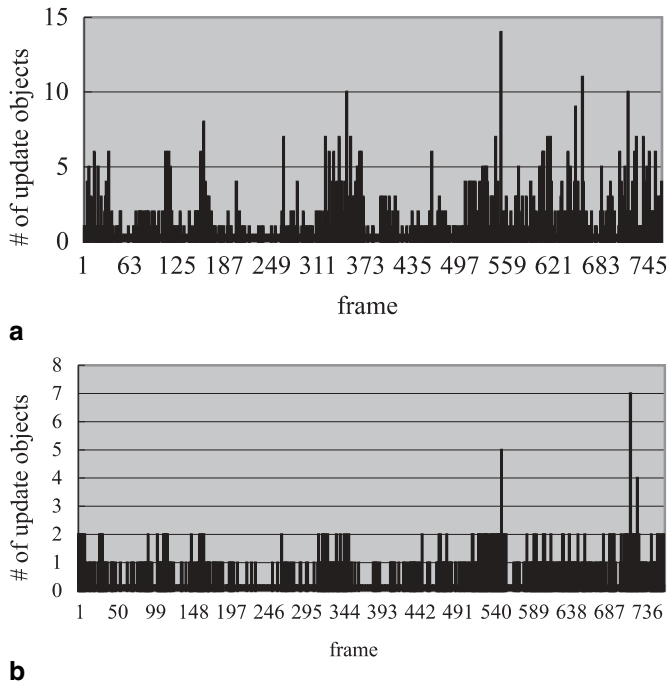


Fig. 14a,b. Comparison of the number of updated objects during navigation for case 1 and case 5

taining 343 objects. The largest view distance is set to 20 m while the view angle is 2 rad. The example path shown in the figure consists of 760 configurations. We test the scene with five different cases of mechanisms. In case 1, no cache and prefetch methods are used, while case 2 uses the LRU cache method. Case 3 use the hybrid cache method, and case 4 uses the hybrid cache method and the second order dead reckoning method for prefetch. Case 5 uses the MLM prefetch method and the hybrid cache method.

5 Results

We first try to observe if the real-time scene management component can effectively manage the resources on the client side. Figure 14 shows the numbers of objects retrieved during the 760-step navigation for case 1 and 5, respectively. We notice that as we navigate through the whole path, the number of objects that need to be retrieved for case 5 is much lower and stable compared to case 1. We also observe that the memory consumption is only slightly increased from 30 588 KB to 32 400 KB during the navigation, which illustrates the effectiveness of our scheme in term of resource management.

The average frame update rates and standard deviations for five different cases are shown in Fig. 15. The data for case 1, 2, and 3 show that the cache mechanism can help the average frame rate by as much as 24%. Both case 2 and 3 are effective in saving model retrieval time. However, the frame rate for the hybrid method is not necessarily much higher than the LRU-based method. We think the main reason is that the cache size we used in our experiment is large enough to accommodate the objects seen in most recent frames. Therefore, the spatial cohesion for objects that are seen long time ago is not as important.

We then compare the effectiveness of the prefetch model. In Fig. 15(b), we observe that the standard deviations (STDVs) for case 4 and 5 are less than case 2 and 3. The STDV for case 4 that uses the DR method is about 12% less, while case 5 (the MLM method) is 83% less. This means that the MLM method can greatly improve the smoothness of the navigation experience. However, we also observe that the average frame rates for case 4 and 5 decrease since the objects that are prefetched in each frame update may

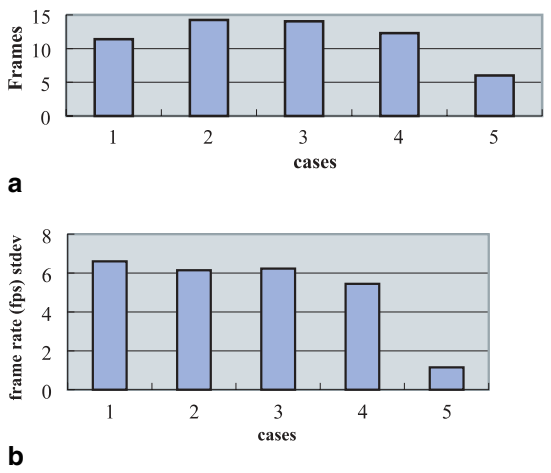


Fig. 15a,b. Performance comparison. **a** Average frame rates for different cases. **b** Standard deviations on frame rates for difference cases

not be always used in the later frames due to prediction errors. In other words, the average and standard deviation of the frame rates are tradeoffs affecting each other. A user can tune the system behavior by adjusting the acceptable frame rate ($FR_{acceptable}$) to affect the prefetch time in each frame update. For example, a user can trade the degree of smoothness for higher frame rates by increasing $FR_{acceptable}$. Nevertheless, we think the frame rate does not always reflect the usability of a navigation system since fast but sluggish navigation could be even more troublesome than slow but consistent navigation.

6 Conclusion and future work

In this paper, we have proposed a data management scheme to solve the problem of smooth navigation in a large virtual environment. We have used the visibility culling method for real-time scene management in order to reduce the amount of data that needs to be transmitted from the server. We have also designed a hybrid cache mechanism to replace objects with the principles of spatial as well as temporal coherences. In addition, we have adopted an MLM prefetch model to predict the possible viewpoint configurations. In order to increase prediction accuracy, we have taken the mouse control characteristics and environment obstacles into consideration. The experiments we have done in our navigation system show that the proposed data management scheme can increase the average frame rate and the navigation smoothness.

In our current real-time scene management scheme, we have used a plane sweeping method to cull an invisible object. However, the computation of such an operation could be rather time consuming for a large environment. Consequently, it could become a bottleneck for real-time scene management. One possible solution for this scalability problem is dividing the environment into several subdivisions so that we can focus on only a small portion of the world at a time. In addition, an important parameter that needs to be adjusted at run time in our system is $FR_{acceptable}$. Since there are no objective criteria for this parameter, a user has to set its value by experience. In the future, we would like to develop a mechanism that allows feedback of the degree of navigation smoothness to provide automatic adjustment of the overall frame rate.

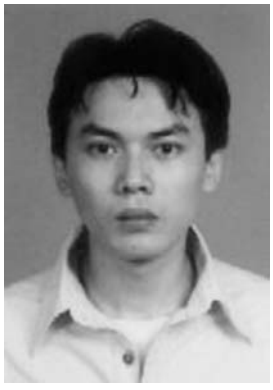
Acknowledgements. This work was partially supported by grants from the National Science Council under NSC 90-2218-E-004-009.

References

1. Airey J, Rohlf J, Brooks F (1990) Towards image realism with interactive update rates in complex virtual building environments. In: Proceedings of ACM 1990 Symposium on Interactive 3D Graphics, pp 41–50
2. Berg M, Kreveld M, Overmars M, Schwarzkopf O (1997) Computational geometry-algorithms and applications. Springer, Berlin Heidelberg New York, pp 20–29
3. Chan A, Lau RWH, Si A (2001) A motion prediction method for mouse-based navigation. In: Proceedings of Computer Graphics International 2001. IEEE Computer Society Press, pp 139–146
4. Chan BYL, Si A, Leong HV (1998) Cache management for mobile databases: design and evaluation. In: Proceedings of the IEEE International Conference on Data Engineering, pp 54–63
5. Chim J, Lau R, Leong H, Si A (1998) Multi-resolution cache management in digital virtual library. In: Proceedings of IEEE Advances in Digital Library, pp 66–75
6. Coorg S, Teller S (1996) Temporally coherent conservative visibility. In: Proceedings of the Twelfth Annual Symposium on Computational Geometry, pp 78–87
7. DeHaemer M, Zyda M (1991) Simplification of objects rendered by polygonal approximations. Comput Graph 15(2):175–184
8. Franklin M, Carey M, Livny M (1992) Global memory management in client-server DBMS architectures. In: Proceedings of the International Conference on Very Large Database, pp 596–609

9. Funkhouser A, Seqin H, Teller J (1992) Management of large amounts of data in interactive building walkthroughs. In: Proceedings of the 1992 Symposium on Interactive 3D Graphics, pp 11–20
10. Gigus Z, Canny Seidel R (1991) Efficiently computing and representing aspect graphs of polyhedral objects. IEEE Trans Pattern Anal Mach Intell 13(4):542–551
11. Gigus Z, Malik J (1990) Computing the aspect graph for line drawings of polyhedral objects. IEEE Trans Pattern Anal Mach Intell 12(2):113–122
12. Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W (1993) Mesh optimization. In: Proceedings of ACM Computer Graphics Conference (SIGGRAPH'93), pp 19–26
13. Katz A, Graham K (1994) Dead reckoning for airplanes in coordinated flight. In: Proceedings of Workshop on Standards for Interoperability of Defense Simulations, pp 5–13
14. Liu G, Maguire G (1996) A class of mobile motion prediction algorithms for wireless mobilecomputing and communications. Mobile Netw Appl 1(2):113–121
15. Macedonia MR, Zyda MJ, Pratt DR, Barham PT, Zeswitz S (1994) NPSNET: A network software architecture for large-scale virtual environments. Presence 3(3):265–287
16. Park S, Lee D, Lim M, Yu C (2001) Scalable data management using user-based caching and prefetching in distributed virtual environments. In: Proceedings of Symposium on Virtual Reality Software and Technology, pp 121–126
17. Taubin G, Rossignac J (1998) Geometric compression through topological surgery. ACM Trans Graph 17(2):84–115
18. Teller S, Sequin C (1991) Visibility preprocessing for interactive walkthroughs. In: Proceedings of ACM Computer Graphics Conference (SIGGRAPH91), pp 61–69

TSAI-YEN LI received his BS in 1986 from the National Taiwan University, Taiwan, and MS and PhD in 1992 and 1995, respectively, from Stanford University. He is currently an associate professor in the Computer Science Department of the National Chengchi University in Taiwan. His main research interests include computer animation, intelligent user interface, motion planning, virtual environment, and artificial life. He is a member of IEEE, ACM, IICM of Taiwan, and TAAI of Taiwan.



WENHSIANG HSU received a MS (2002) from the Computer Science Department, National Chengchi University in Taiwan. His research is in area of virtual environment systems at the Intelligent Media Laboratory.