Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Data Mining (CIDM 2007)

An Efficient Data Management Scheme Based on Spatial and Temporal Characteristics in Virtual Environments

Hsing-Jen Chen and Damon Shing-Min Liu

***Abstract*—In a distributed interactive walkthrough system, there are two major bottlenecks which cause performance degradation. One is the server-side workload in the client-server architecture; the other is the network transmission delay. In this paper, we present a knowledge-based data management scheme which takes consideration of both internal (memory) and external (disk) data storage management to ease server-side workload and reduce network transmissions. Our system first analyzes users' logs to discover the spatial and temporal semantic patterns in the virtual environment. Using these patterns, we can determine the proper data layout on disk, and better improve our caching mechanism. Experimental results show good prediction rates and achieve improvements in overall system performance.**

I. Introduction

VIRTUAL Reality plays an important role in modern three-dimensional graphics domain. Based on this attractive technology, many applications provide a more pleasing interaction between users and computers. For example, people can admire virtual exhibits in Virtual Museum without visiting the real museum; pilots are trained in simulated aircrafts instead of real planes. But unlike traditional text and two-dimensional images, a three-dimensional scene requires more resources. This causes problems that do not arise in traditional applications.

Usually, Virtual Reality is modeled as a walkthrough system. In such a system, users control their avatars to explore the virtual world. Walkthrough systems are often built using a client-server architecture because of portability. Under this architecture, the virtual environment data are stored in the server's storage base. When client makes a connection to the server, the server sends required data back to the client. But as the complexity of the virtual environment grows, the amount of data becomes very large. As a result, it becomes infeasible to transmit the whole-world geometrics to the client. Hence, a scheme that sends only the data needed in current frame is exploited to reduce the frame construction time [1, 9, 16]. But once the data resides only on the server, the client has to request from the server every time the user moves. To prevent the delay of network, a good prefetching technique should be employed.

Hsing-Jen Chen is with Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 621, Republic of China (corresponding author: +886-5-272-0411 ext 23101; fax: +886-5-272-0859; e-mail: chj94@cs.ccu.edu.tw).

Damon Shing-Min Liu is with Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 621, Republic of China. (e-mail:damon@cs.ccu.edu.tw).

Also, how data are organized on server-side is another important task which should be concerned. In a client-server architecture, one or more clients make connections with the server simultaneously. The server has to handle all requests from the clients and perform all necessary I/O to read the requested data. Hence, an efficient technique to speed up I/O access is crucial to the system.

In this paper, we present a data management scheme which takes consideration of both spatial and temporal coherence. We utilize previous users' navigational logs to discover spatial and temporal semantic patterns of the virtual environment in the walkthrough system. Our system applies to both internal and external data management. On the server-side, we determine the arrangement of data on disk storage according to the discovered semantic patterns. On the client-side, the patterns are exploited in buffer replacement. Our method achieves good prediction quality in buffer management, and makes a performance improvement to the system in the conducted experiments.

The rest of the paper is organized as follows: Section 2 surveys previous related research works. Section 3 describes how we construct the Correlation Array, which we stores the relation knowledge. Section 4 discusses the influence of disk layout. The caching scheme is formulated in Section 5. Section 6 shows the results of our experiments, and Section 7 concludes this paper.

II. Related Work

This section addresses issues caused in walkthrough systems and discusses approaches proposed in the literature.

Distributed walkthrough systems are designed to provide remote users to enjoy the immersion in virtual environments. However, disturbance due to latency of network diminishes the population of this application. Several approaches, such as SIMNET [4] and VLNET [6], are proposed to resolve this problem by replicating the whole virtual environments to the client side. But these approaches resulted in long setup time. Other works suggest on-demand transmission [1, 9, 16]. This design employs partial replication. The server only transmits necessary data to the client at setup time. After initial setup, the client requests the server to send other data every time it takes a movement. Although only part of data is transmitted, it still needs a good prefetching and caching mechanism to provide users smooth walkthroughs.

There are two major approaches of prefetching and caching mechanism. One is LOD (Levels of Detail) based approaches. LOD-based methods [12, 14, 15] regard a virtual environment as a collection of 3D objects. Each object has one or more representations, such as several different levels of LOD models, or image-based imposters. The server's task is to dynamically choose one representation which would add most contribution to the scene currently rendered at the client's display and transmit it to the client. A representation with higher contribution may add more fidelity to the scene, but it also takes more time to transmit. A benefit function is defined to balance the contribution and the cost based on current configuration to choose one representation with most profit.

The other studies are based on spatial distance [3, 5, 16]. These systems design their prefetching strategy or cache replacement policy based on spatial relationship. They assume that near objects are more likely to be demanded by the client. They divide the virtual environments into small cells. According to which cell the user's position belongs to, the system prefetches (or keeps caches of) the neighboring cells [3, 16].

In addition to previous works that exploit only spatial relationship, Park *et al.* [7] proposes their approach which also takes users' behavior into account. Their method is based on three parts: user's navigational behavior, object popularity, and the spatial relationship. Combining the three factors, they apply the prefetching and caching policy to users according to each user's personal interests. Their work is similar to ours. However, our approach exploits users' behavior to generate the spatial and temporal semantics automatically.

Most of the works consider the problem of internal storage management, but only a few discuss the efficiency of external object data storage. Shou *et al.* states the influence of I/O bottleneck upon overall system performance [10]. They propose to optimize I/O performance by incorporating with a complementary search algorithm. The algorithm reduces I/O access by performing I/O only for those have not been in main memory. But they did not discuss the arrangement of objects on disk. The proposed I/O reduction technique is also employed in our system.

III. Correlation Array

A virtual environment in a walkthrough system consists of numerous numbers of object geometries. The problem, from our aspect, is to design a data management scheme that can efficiently reduce the server overhead and network transmission delays.

In previous designs [3, 16], the relationships of objects have to be assigned manually and only spatial relationships are employed. In our system, we utilize existing user logs to automatically discover spatial and temporal relationships between objects. The user logs consist of users' position coordinates and motions information. Analyzing these logs can help to understand user's navigational preferences and objects' relationships both in space and in time.

Knowing which objects are most related is important in deciding the arrangement of objects and caching policy. In this section, we explain how we gain and maintain such knowledge. We analyze the recorded user logs, and construct a data structure, *Correlation Array*. This array reveals the relationships between objects. With this array, we have the capability to know which set of objects are most related. Given a certain object, we can also determine those objects with most relevance.

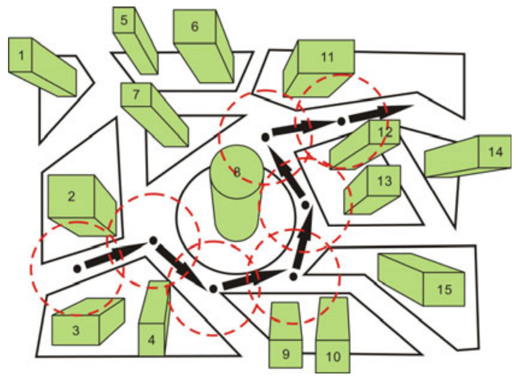

Figure 1. An example of a user's path. The circle is Visibility Limit. Any object within the circle is inside the user's view. The observed path is {(2,3)(2,4)(4)(9,10)(8,12,13)(8,11)(8,11,12)}.

A. The Walkthrough System

Before describing our method, we define the notions of the walkthrough system here.

In a virtual environment, users can see objects within some distance when they explore the virtual environment. The distance is called *Visibility Limit*. When the user is at some position, the objects within that distance form a *view*. While the user exploring the virtual environment, we can observe a sequence of views. The sequence of views, from the view of the user's starting position, to the location where the user exits, is a user's navigational *path*. As shown in Figure 1. Formally, we define a view as $v = (o_1, o_2, \ldots, o_n)$, where o_i is an object in virtual world. And the path is an ordered sequence of views $p = v_1 v_2 \ldots v_l$, where l is the length of the path.

B. Correlation Pairs

Correlation pair is a topic in data mining research domain [2, 13]. The problem can be formulated as follows: Given a user-specified threshold θ and a market database with N items and T transactions, the aim is to find all pairs of items with correlation higher than minimum threshold θ. The correlation is defined as the rate that two items appear together in the same transaction. The more two items appear together, the higher their correlation will be. There are some researches related to this problem, including how to find the pairs efficiently [13]. But they are beyond our discussion. Here we only show how we translate the correlation pair problem to our problem.

For each pre-recorded user's path, or, a sequence of views, we divided it into a set of single views, where each view contains a set of objects. After dividing all the paths we record, we get a set of views. Then, we treat the views as transactions in market database mining and feed them into an existing method that finds correlation pairs. The method returns pairs of items, or in our case, objects.

C. Constructing a Correlation Array

From the above discussions, we have pairs of objects with correlations higher than θ. These correlation pairs are found because they often appear together within a view. They have several important properties: First, any two objects of a correlation pair is geometrically close to each other. Second, the frequency of any pair appears together is higher than the threshold. Using these pairs, we need to construct a correlation array such that any neighboring objects in the array have a high correlation.

We formalize the problem as follows: Given a set of correlation pairs *C*, the output array should minimize

$$\sum_{(i,j)\in C} cor(i,j) * dist(i,j)$$

, where *cor(i, j)* is correlation between *i* and *j*, *dist(i ,j)* is distance of *i* and *j* in the output array.

We use a greedy algorithm to construct the correlation array. The main idea is to put objects with high correlation as close to each other as possible. Details of this algorithm are described in Algorithm 1.

Algorithm 1 Form the Correlation Array

```
Input: the Correlation pair set C
Output: the Correlation Array CA

1:  initialize ArraySet = { };
2:  while C is not empty
3:      choose pair p with maximum correlation from C;
4:      if two objects of p are found in different arrays in
        ArraySet
5:          join the two arrays
6:      if only one object is found in ArraySet
7:          join the pair with the found array in ArraySet;
8:      if no object is found in ArraySet
9:          generate an array of pair p and add it to ArraySet
10:     remove p from C;
11: end of while
12: CA = concatenate all array in ArraySet;
13: return CA;
```

Note that in Algorithm 1 we use several array operations. Below we define these operations.

1. *concat(A, B)*: The concatenation of two arrays is defined as concat(*A*, *B*) = *C*, where *C* is an array that expands *A* with *B*. For example, concat([*abc*], [*de*]) = [*abcde*]. For convenience, we denote it as concat(*A*, *B*) = *AB*.
2. $\overline{A}$: The reversion of *A*. That is, $\overline{[abcd]} = [dcba]$.
3. I_a^A : The index of object *a* in array *A*.
4. *cost(A, p)*: *A* is an array, *p* is a pair, let *p*=(*a*,*b*), define that cost(*A*, *p*) = $| I_a^A - I_b^A |$.
5. *join(A, B, p)*: *A* and *B* are both arrays, *p* is a pair, this operation returns an array $d \in J$ that minimizes cost(*d*, *p*), where $J = \{AB, \overline{A}B, A\overline{B}, \overline{A}\overline{B}, BA, \overline{B}A, B\overline{A}, \overline{B}\overline{A}\}$.

The join operation forms a new array from two existing arrays. The array here is a sequence of objects. We only care about the relation between objects within an array. The absolute position of an object does not matter. Hence, when joining two arrays, we should consider all the possibilities and choose the one with minimum cost. Some of the resulting arrays are equal in our context. Thus, they can be reduced to

$J = \{AB, \overline{A}B, A\overline{B}, BA\}$.

The constructed Correlation Array contains valuable information about relations between objects. We utilize this information in our disk layout and cache management.

IV. Disk Layout

In a many-to-one architecture, the server which carries all the geometry data has to perform the I/O requests for many clients. It could be a bottleneck if numerous clients explore in the virtual environment at the same time. Therefore, a technique to speed up I/O access is essential for the overall performance.

A disk is usually considered as a linear model, which stores data in a one-dimensional manner. That is, sequential accesses are much faster than random ones [11]. However, many applications such as walkthrough systems with strong spatial coherence require multidimensional data structures to retain the locality. Other works have shown that efficient multidimensional data access relies on maintaining locality so that "neighboring" objects in the multidimensional space are stored in "neighboring" disk locations [8]. By constructing a correlation array, the multidimensional locality has been transformed into a one-dimensional form, which matches the disk characteristic.

In our disk layout object arrangement, we simply place the objects of the virtual environment on disk in the order of the correlation array, such that objects with higher correlation are arranged together. The relation-oriented organization efficiently reduces the distance the disk reading head has to move for a single request, and hence the server can complete an I/O request faster.

V. The Prefetching Scheme

In Section 4, we described how we determine data arrangement on disk according to correlation array. In this section, we explain how we exploit this array in prefetching scheme.

We integrate our prefetching with caching mechanism into a predictive caching scheme. This scheme not only manages the cache, but does prefetching for quicker response time.

The aim of a prefetching scheme is to fetch the objects before the system actually demands them. A successful

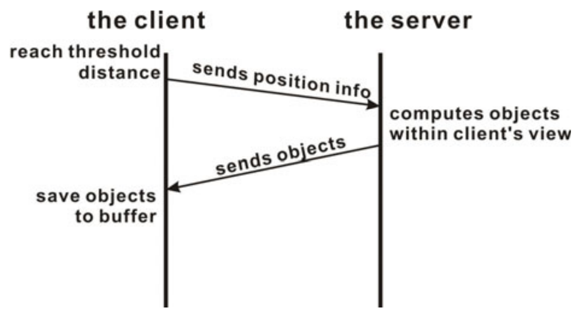

Figure 2. Mechanism of our client-server architecture.

prefetching scheme can serve the client a smooth walkthrough without the latency of network. If objects are not in the client's memory when needed, a prefetching failure occurs. A walkthrough system typically can handle this situation in two ways. First, the system is stalled to wait for the needed objects to be retrieved. Second, the system skips the object that is absent in memory. Both of them are undesirable and cause system performance degradation.

A. *Client-server Architecture*

In the client-server architecture, the geometric data of objects are stored on the server side. When user navigates the virtual environment, the client sends a request of objects to the server once it reaches the threshold distance. The request consists of the client's current position and direction. The server uses this information to calculate which objects are inside the client's view frustum and transfers those objects to the client. See Figure 2. A fixed-size buffer is maintained by the client to store the received data.

The server also keeps track of the client's buffer. Before sending object geometries, the server checks its record to decide whether the data has been residing in the client's buffer or not. This process helps to avoid unnecessary network loads. This technique is exploited in many existing implementations [5, 10, 16].

B. *Predictive Caching Mechanism*

Our caching mechanism takes place when the server has decided which objects are inside the client's view frustum, see Figure 3. The mechanism utilizes the constructed correlation array and predicts the objects which the client may see in the near future.

When server receives the client's request, it employs a culling algorithm to find the objects O which are within the client's view frustum. Assume that the size of O is k. Having these objects, the server checks the correlation array CA and finds the *(ClientBufferSize – k)* objects R those are *most related* to O and sends them to the client for the replacement of buffer. As shown in Algorithm 2.

Algorithm 2 Finding Objects to Be Prefetched

Input: a Correlation Array CA, the object set within the client's view frustum O, the client's buffer size N
Output: the object set R to be prefetched

```
1:  initialize $w = N - |O|$; $R = \{\}$; $j = 1$;
2:  while $w > 0$
3:      for each object $t \in O$
4:          find object $N_j(t)$ that is the $j$th nearest object of $t$ in $CA$;
5:          if object $N_j(t)$ is not in $O$ or $R$;
6:              if($w == 0$)
7:                  return $R$;
8:              add $N_j(t)$ to $R$;
9:              $w$--;
10:     $j$++;
11: end of while
12: return $R$;
```

Note that $N_j(t)$ is defined as the jth closest object in the correlation array.

Using correlation array, we can prefetch objects even when they are not directly related. Consider the situation, object a and object b have a high correlation, while object b and object c have a high correlation too. When client sees object a within user's view frustum, the server not only prefetches object b which is correlated to a, but prefetches object c if the client has more space in the buffer. The hidden (indirect) correlations are suitable for predicting the client's future requests.

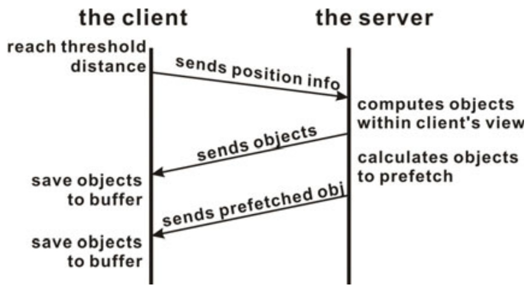

Figure 3. Mechanism of our client-server architecture with prefetching.

VI. Experiments

We evaluated our approach on a walkthrough system of a power plant scene. The power plant model is created by Walkthrough Laboratory of Computer Science Department of University of North Carolina. The client-server architecture of walkthrough system is implemented in Java and the objects are stored in VRML format. The entire scene consists of 11949 objects and the storage size of objects is 335 MB. All objects data are stored in the server's hard disk. The client uses a VRML browser to navigate the virtual world and retrieve objects from the server via network. A snapshot of the system is given in Figure 4.

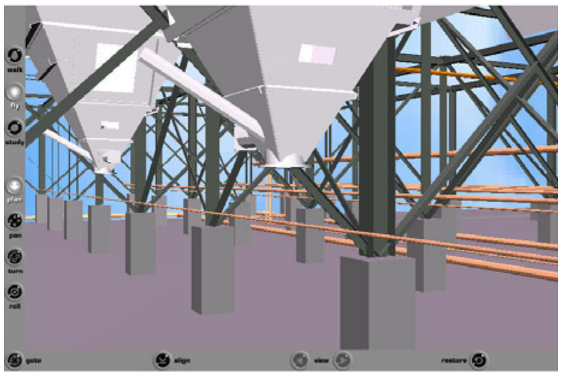

Figure 4. A snapshot of our walkthrough system.

We first evaluated the performance of disk layout. To measure the performance, we made two assumptions for simplicity. First, each object occupies one disk track. Second, moving disk head from one track to its neighboring track costs one seek distance. The assumptions simplify the complicated disk model which has many low-level details. Based on the assumptions, we can measure the performance of a disk layout by comparing their seek distances. A longer seek distance implies more time required to move the reading head of disk to the next demanded data, and hence takes more time for the server to accomplish the I/O access.

We compared the disk layout before and after our arrangement. The test dataset is 410 user logs which are generated by randomly anonymous users. For easy comparison, we sum the seek distances in the 410 experiments. As shown in Figure 5, the arrangement significantly reduces the seek distance. This is not surprising as the disk layout before our arrangement is randomly distributed. However, our correlation-based layout coheres objects according to their relevancy.

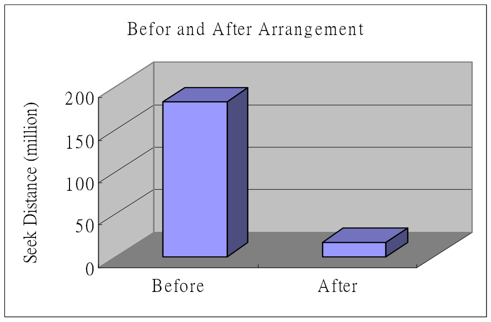

Figure 5. Performance comparison before and after our arrangement.

To compare the performance of different designs of caching mechanism, we define a walkthrough quality measure *Miss Ratio* - the number of objects those are absent in the client's memory when they should be rendered, divided by the number of all objects inside the client's view frustum. Miss ratio can be considered as a fidelity measurement of a virtual environment. Since a high miss ratio implies more objects are missing during the navigation.

We compared our caching mechanism to a simple LRU (Least Recently Used) strategy. LRU always keeps objects those are accessed recently, and deletes the one which has not been accessed for the longest time. Due to LRU's principle which takes advantage of temporal coherence, it achieves good prediction quality.

Figure 6 shows the result of our experiment. We tested LRU and our method with different sizes of buffer. In all the experiments, our method has lower miss ratio than LRU. As the buffer size decreases, miss ratio of LRU keeps increasing, while our method remains constant.

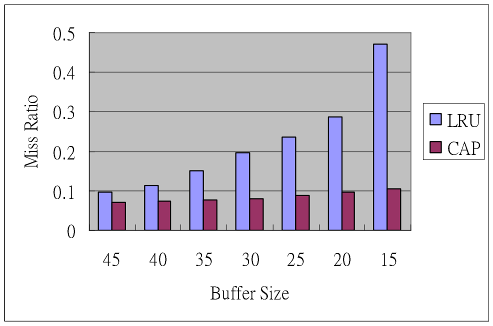

Figure 6. Caching performance comparison under different buffer size.

Figure 7 shows another result of our experiments. We tested the two approaches with different lengths of paths. In all experiments, our method has lower miss ratios than LRU. The average miss ratio of LRU is 0.245, while our method is 0.105. In the experiment of path with 100 steps, the miss ratio of LRU is even four times that of our method.

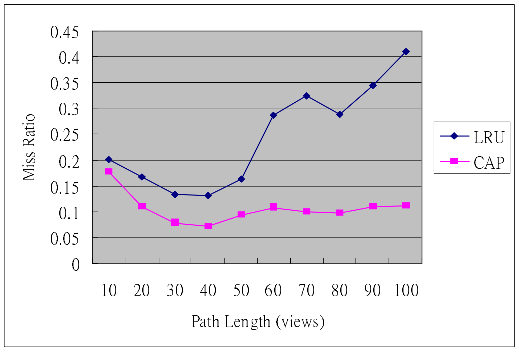

Figure 7. Caching performance comparison under different lengths of paths.

It should be noted that at the initialization of a walkthrough, the client has to request all objects inside the avatar's initial view frustum. This is the reason that the miss ratios of both methods are high in paths with short length. After the setup-time transmission, our method remains low miss ratio and hence provides the client a smooth walkthrough.

VII. Conclusion

We have described a client-server based data management scheme utilizing prior knowledge to efficiently ease server workload and speed up the client response time. This method first constructs a correlation array to maintain the correlations between objects. According the correlation array, we determine the placement of data on disk. When handling object requests from the interactive client, the server exploits this array to get the objects which are highly correlated to those inside the client's view frustum for predictive caching. The experiments show that this technique improves our walkthrough system performance.

The work is still in progress. We plan to extend the utilization of the correlation array to improve the walkthrough system in other different aspects.

References

[1] M. Capps, "The QUICK framework for task-specific asset prioritization in distributed virtual environments," *Proceedings of IEEE Virtual Reality 2000*, 2000, pp. 143-150.

[2] S.K. Kachigan, *Multivariate Statistical Analysis: A Conceptual Introduction.* Radius Press, 1991.

[3] V. Koltun, Y. Chrysanthou, and D. Cohen-Or, "Hardware-accelerated from-region visibility using a dual ray space," *EGWR01: 12th Eurographics Workshop on Rendering*, June, 2001, pp. 204-214.

[4] D. Miller and J. Thorpe, "SIMNET: the advent of simulator networking," *Proceedings of IEEE*, vol.83, 1995, pp. 1114-1123.

[5] C. Ng, C. Nguyen, D. Tran, T. Tan, and S. Yeow, "Analyzing pre-fetching in large-scale visual simulation," *Computer Graphics International 2005*, 2005, pp. 100-107.

[6] I. Pandzic, T. Capin, E. Lee, N. Thalmann, and D. Thalmann, "A Flexible architecture for virtual humans in networked collaborative virtual environments," *Proceedings of Eurographics'97*, 1997, pp. 177-188.

[7] S. Park, D. Lee, M. Lim, and C. Yu, "Scalable data management using user-based caching and prefetching in distributed virtual environments," *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 2001, pp. 121-126.

[8] S. Schlosser, J. Schindler, S. Papadomanolakis, M. Shao, A. Ailamaki, C. Faloutsos, and G. Ganger, "On multidimensional data and modern disks," *Proceedings of the 4th USENIX Conference on File and Storage Technology (FAST '05)*, 2005.

[9] D. Schmalstieg, and M. Gervautz, "Demand-driven geometry transmission for distributed virtual environments," *Proceedings of Eurographics*, 1996, pp. 421-433.

[10] L. Shou, J. Chionh, Z. Huang, Y. Ruan, and K. Tan, "Walking through a very large virtual environment in real-time," *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 401-410.

[11] E. Shriver, A. Merchant, and J. Wilkes, "An analytic behavior model for disk drives with readahead caches and request reordering," *ACM SIGMETRICS Performance Evaluation Review*, Vol.26, Issue 1, 1998, pp. 182-191.

[12] E. Teler, D. Lischinski, "Streaming of complex 3D scenes for remote walkthroughs," *Computer Graphics Forum*, 20(3), September, 2001, pp. 17-25.

[13] H. Xiong, S. Shekhar, P.N. Tan, and V. Kumar, "Exploiting a support-based upper bound of Pearson's correlation coefficient for efficiently identifying strongly correlated pairs," *ACM Special Interest Group on Knowledge Discovery and Data Mining* 2004, 2004, pp. 334-343.

[14] C. Zach, "Integration of geomorphing into level of detail management for realtime rendering," *Proceedings of the 18th Spring Conference on Computer Graphics*, 2002, pp. 115-122.

[15] C. Zach, and K. Karner, "Prefetching Policies for Remote Walkthroughs," *Technical report 2002-010*, VRVis Research Center, 2002.

[16] Z. Zheng, and T. K.Y. Chan, "Optimized neighbour prefetch and cache for client-server based walkthrough," *Proceedings of the 2003 International Conference on Cyberworlds*, 2003, pp. 143-150.