Visibility-Based Prefetching for Interactive Out-Of-Core Rendering

Wagner T. Corrêa* Princeton University
James T. Klosowski† IBM Research
Cláudio T. Silva‡ University of Utah

Abstract

We present a new visibility-based prefetching algorithm for interactive out-of-core rendering of large models on an inexpensive PC. Using an approximate visibility technique, we can very accurately and efficiently determine which geometry will be visible in the near future and prefetch that geometry from disk before it must be rendered. Our prefetching algorithm is a key part of a visualization system capable of rendering a 13-million triangle model with 99% accuracy at interactive frame rates. Our prefetching algorithm is the first of its kind to be based on a from-point visibility technique, and enables interactive rendering on a commodity PC, as opposed to expensive high-end graphics workstations or parallel machines.

CR Categories: I.3.2 [Graphics Systems]: Computer Graphics—Computing Methodologies; I.3.3 [Picture/Image Generation]: Computer Graphics—Computing Methodologies

Keywords: out-of-core rendering, interactive rendering, commodity PCs, occlusion culling, prefetching, walkthrough

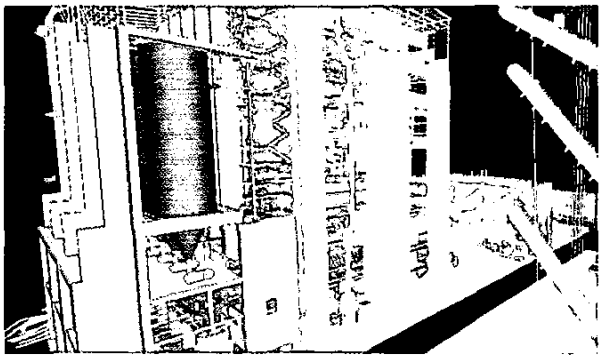


Figure 1: Our out-of-core rendering system can preprocess a 13-million-triangle model in 3 minutes, and then render it with 99% accuracy at 10 frames per second on an inexpensive PC.

1 Introduction

In this paper, we present a new visibility-based prefetching algorithm for retrieving out-of-core 3D models and rendering them at interactive rates on an inexpensive PC. Interactive rendering of large models has applications in many areas, including computer-aided design, engineering, entertainment, and training. Traditionally, interactive rendering of large models has required triangle throughput only available on high-end graphics workstations or parallel machines that cost hundreds of thousands of dollars. Recently, with the explosive growth in performance of PC graphics cards that cost a few hundred dollars, inexpensive PCs are becoming an attractive alternative to high-end machines. Although inexpensive PCs can match the triangle throughput of high-end machines, inexpensive PCs have much less main memory than high-end machines. Today a typical high-end machine has 16 GB of main memory, while a typical inexpensive PC has 512 MB (32 times less). Thus, a challenge in exploiting the performance of PC graphics cards is designing rendering systems that work under tight memory constraints.

Our rendering system, iWalk, overcomes the memory constraints of an inexpensive PC by using an out-of-core preprocessing algorithm and a new multi-threaded out-of-core rendering approach to overlap rendering, visibility computation, and disk operations. The preprocessing algorithm breaks the geometry of a model into manageable pieces, and creates on disk a spatial subdivision of the geometry. At runtime, the system maintains in memory a cache of the most recently used geometry.

The system can run in two visibility modes: approximate or conservative. In approximate mode, the system tries to maximize the quality of the rendered images, given a user-defined budget of geometry per frame. This budget is based on the hardware capabilities (such as the graphics card triangle throughput and the disk bandwidth) and the target frame rate. In conservative mode, for those instances that cannot tolerate any errors in the rendered images, the system determines and renders all the visible geometry, potentially at lower frame rates. In either visibility mode, as the viewpoint changes, the visible geometry changes, and the geometry cache has to load from disk the visible geometry that is not in the cache. Even small changes in the viewpoint can cause large changes in visibility. This problem manifests itself as abrupt drops in frame rates because of bursts of disk operations.

The prefetching algorithm we present in this paper addresses this problem. The goal of prefetching is to have the geometry already in memory by the time it is needed. The prefetching algorithm runs as a separate thread, and is orthogonal to the visibility mode. The prefetching thread uses a from-point visibility algorithm to find the geometry the user is likely to see in the near future, and sends prefetch requests to the geometry cache. If the geometry cache is busy loading geometry needed for the current frame, it ignores prefetch requests; otherwise, it loads the requested geometry from disk. By amortizing the cost of bursts of disk operations over frames with few disk operations, prefetching improves the performance of the system in either visibility mode.

The main contribution of this paper is a multi-threaded out-of-core rendering approach which to our knowledge is the first to combine speculative prefetching with a from-point visibility algorithm.

*Department of Computer Science, Princeton University, 35 Olden St., Princeton, NJ 08540; wtcorrea@cs.princeton.edu.

†Visual Technologies, IBM T. J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598; jklosow@us.ibm.com.

‡Work performed while at AT&T. Currently at Scientific Computing and Imaging Institute, School of Computing, University of Utah, 50 S. Central Campus Dr., Salt Lake City, UT 84112; csilva@cs.utah.edu.

IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003,
October 20-21, 2003, Seattle, Washington, USA
0-7803-8120-3/03/$17.00 ©2003 IEEE

Our prefetching algorithm plays a critical role in our iWalk system, which is capable of rendering a model with tens of millions of polygons at interactive frame rates on an inexpensive PC (Figure 1).

2 Related Work

Researchers have studied the problem of rendering complex models at interactive frame rates for many years. Clark [1976] proposed many of the techniques for rendering complex models used today, including the use of hierarchical spatial data structures, level-of-detail (LOD) management, hierarchical view-frustum and occlusion culling, and working-set management (geometry caching). Garlick et al. [1990] presented the idea of exploiting multiprocessor graphics workstations to overlap visibility computations with rendering. Airey et al. [1990] described a system that combined LOD management with the idea of precomputing visibility information for models made of axis-aligned polygons.

Funkhouser et al. [1992] described the first published system that supported models larger than main memory, and performed speculative prefetching. Their system was based on the from-region visibility algorithm of Teller and Séquin [1991], which required long preprocessing times, and was limited to models made of axis-aligned cells. Our system is based on the from-point visibility algorithm of Klosowski and Silva [2000; 2001], which requires very little preprocessing, and can handle any 3D polygonal model.

Aliaga et al. [1999] presented the Massive Model Rendering (MMR) system, which employed many acceleration techniques, including replacing geometry far from the user's point of view with imagery, occlusion culling, LOD management, and from-region prefetching. MMR was the first published system to handle models with tens of millions of polygons at interactive frame rates, although it did require an expensive high-end multi-processor graphics workstation.

Wald et al. [2001] developed a ray tracing system that used a cluster of 7 dual-processor PCs to render low-resolution images of models with tens of millions of polygons at interactive frame rates. Avila and Schroeder [1997] and El-Sana and Chiang [2000] developed systems for interactive out-of-core rendering based on LOD management, but these systems did not perform occlusion culling. Varadhan and Manocha [2002] describe a system for out-of-core rendering that uses hierarchical LODs [Erikson et al. 2001] and prefetching, but their system does not perform occlusion culling, and their preprocessing step is in-core.

Wonka et al. [2001] employed a from-point visibility algorithm and used two processors to overlap visibility computation and rendering at runtime (similarly to the idea introduced by Garlick et al. [1990]), but they only reported results for 2.5D environments that were smaller than main memory. Many other researchers have also developed systems for out-of-core rendering, but without focusing on achieving interactive frame rates [Chiang and Silva 1997; Chiang et al. 1998; Cox and Ellsworth 1997; Pharr et al. 1997; Shen et al. 1999; Sutton and Hansen 2000].

3 Out-Of-Core Preprocessing

Recall that our main goal is to render a large model using an inexpensive PC with small memory. To accomplish this, we must first construct an out-of-core hierarchical representation for the model during a preprocessing step, and then at runtime load on demand the hierarchy nodes that the user sees. Our current algorithm builds an out-of-core octree [Samet 1990] whose leaves contain the geometry of the model. To store the octree on disk, we save the geometric contents of each octree node in a separate file, and create a *hierarchy structure* (HS) file, which stores information about the spatial relationship of the nodes in the hierarchy, and for each node it contains the node's bounding box and auxiliary data needed for visibility culling. The HS file is the main data structure that our system uses to control the flow of data and is assumed to fit in memory. For the 13-million triangle model used throughout this paper, the HS file was only 3 MB.

An in-core approach to build an octree for a model would process the entire model in one pass, using a machine with large enough memory to hold both the model and the resulting octree. We avoid this brute-force approach because we do not want to use a separate expensive machine with large memory just to build the octree. Our out-of-core algorithm builds an octree for a model directly on machines with small memory. The algorithm first breaks the model in sections that fit in main memory, and then incrementally builds the octree on disk, one pass for each section, keeping in memory only the section being processed. Our preprocessing algorithm requires no user intervention and is very fast, often orders of magnitude faster than previous approaches. It constructs the octree for the UNC power plant model [UNC 1999] in just 3 minutes, while the MMR system [Aliaga et al. 1999] required over two weeks, and the system by Wald et al. [2001] spent 2.5 hours preprocessing the same model.

Our preprocessing is similar in nature to several other construction algorithms [Cignoni et al. 2002; Ueng et al. 1997; Wald et al. 2001]. It is most akin to the algorithm of Cignoni et al. [2002], and therefore has comparable preprocessing times. Other recent construction algorithms are presented in [Durand et al. 2000; Schaufler et al. 2000; Wonka et al. 2000; Wonka et al. 2001]. As our construction algorithm is not the main focus of this paper, we refer the reader to [Corrêa et al. 2002] for a thorough examination of the differences between all of these algorithms.

4 Out-of-Core Rendering

Figure 2 shows a diagram of iWalk's rendering approach. The user interface (a) keeps track of the position, orientation, and field-of-view of the user's camera. For each new set of camera parameters, the system computes the visible set — the set of octree nodes that the user sees. According to the user's choice, the system can compute an approximate visible set (b), or a conservative visible set (c). To compute an approximate visible set, iWalk uses the prioritized-layered projection (PLP) algorithm [Klosowski and Silva 2000]. To compute a conservative visible set, iWalk uses cPLP [Klosowski and Silva 2001], a conservative extension of PLP. For each node in the visible set, the rendering thread (d) sends a fetch request to the geometry cache (i), which will read the node from disk (j) into memory. The rendering thread then sends the node to the graphics card (e) for display (f). To avoid bursts of disk operations, the look-ahead thread (g) predicts where the user's camera is likely to be in the next frame. For each predicted camera, the look-ahead thread uses PLP (h) to estimate the visible set, and then sends prefetch requests to the geometry cache (i).

To better understand our rendering approach, we need to briefly review the visibility algorithms that iWalk uses. PLP is an approximate, from-point visibility algorithm that may be thought of as a set of modifications to the traditional hierarchical view frustum culling algorithm [Clark 1976]. First, instead of traversing the model hierarchy in a predefined order, PLP keeps the hierarchy leaf nodes in a priority queue called the *front*, and traverses the nodes from highest to lowest priority. When PLP visits a node, it adds it to the *visible set*, removes it from the front, and adds the unvisited neighbors of the node to the front. Second, instead of traversing the entire hierarchy, PLP works on a budget, stopping the traversal after a certain number of primitives have been added to the visible set. Finally, PLP requires each node to know not only its children, but also all of its neighbors.

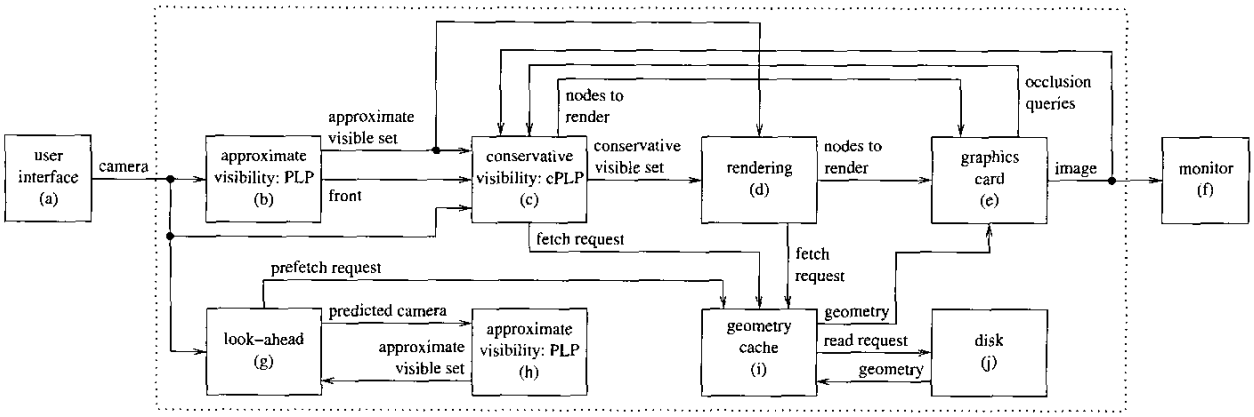


Figure 2: The multi-threaded out-of-core rendering approach of the iWalk system. For each new camera (a), the system finds the set of visible nodes using either approximate visibility (b), or conservative visibility (c). For each visible node, the rendering thread (d) sends a fetch request to the geometry cache (i), and then sends the node to the graphics card (e). The look-ahead thread (g) predicts future cameras, estimates the nodes that the user would see then (h), and sends prefetch requests to the geometry cache (i).

In addition to being time-critical, another key feature of PLP that iWalk exploits is that PLP can generate an approximate visible set based on just the information stored in the hierarchy structure file created at preprocessing time. In other words, PLP can estimate the visible set *without* access to the actual scene geometry.

An implementation of PLP may be simple or sophisticated, depending on the heuristic to assign priorities to each node. Several heuristics precompute for each node a value between 0.0 and 1.0 called *solidity*, which estimates how likely it is for the node to occlude an object behind it. At run time, the priority of a node is found by initializing it to 1.0, and attenuating it based on the solidity of the nodes found along the traversal path to the node (Figure 3).

Although PLP is in practice quite accurate for most frames, it does not guarantee image quality, and some frames may show objectionable artifacts. To avoid this potential problem, the system may use cPLP [Klosowski and Silva 2001], a conservative extension of PLP that guarantees 100% accurate images. However, cPLP cannot find the visible set from the HS file only, and needs to read the geometry of all potentially visible nodes. These additional disk operations may make cPLP much slower than PLP. Our implementation of cPLP can use either an item-buffer technique that is portable to any platform that supports OpenGL, or occlusion query extensions (such as the HP test [Severson 1999] and the nVidia occlusion query [Rege 2002]) when they are available. Thus, cPLP needs to fetch geometry from the geometry cache, and read pixels or occlusion queries from the graphics card.

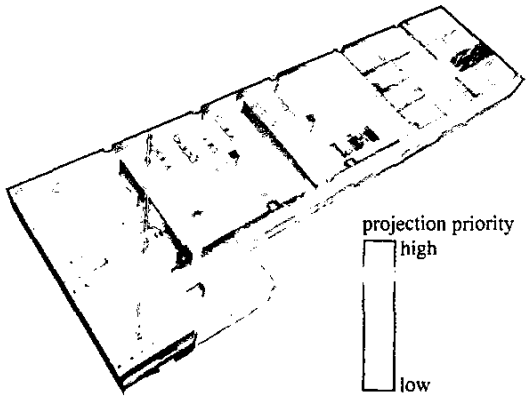


Figure 3: A section of the UC Berkeley Soda Hall model. At run-time, the iWalk system uses the prioritized-layered projection (PLP) algorithm to estimate the nodes potentially visible from the current view frustum (outlined in yellow). The transparent color of each node indicates the projection priority of the node.

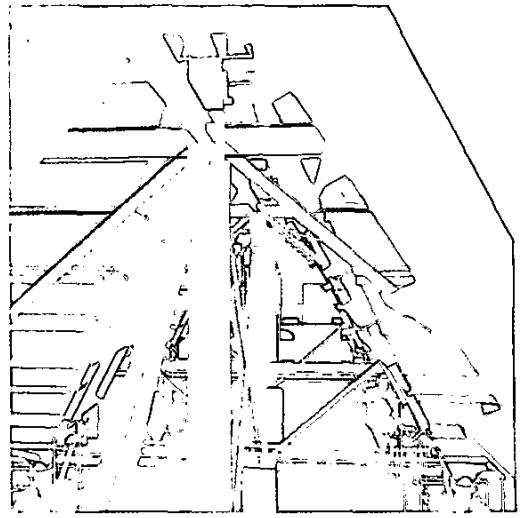
5 From-Point Visibility-Based Prefetching

The idea behind prefetching is to predict a set of nodes that the user is likely to see next, and bring them to memory ahead of time. Ideally, by the time the user sees those nodes, they will be already in the geometry cache, and the frame rates will not be affected by the disk latency. Systems researchers have studied prefetching strategies for decades [Gindele 1977; Przybylski 1990], and many previous rendering systems [Aliaga et al. 1999; Funkhouser 1996; Funkhouser et al. 1992; Varadhan and Manocha 2002] have used prefetching successfully. To our knowledge, all previous prefetching methods that employ occlusion culling have been based on from-region visibility algorithms, and were designed to run on multiprocessor machines. Our prefetching method works with from-point visibility algorithms, and runs as a separate thread in a uniprocessor machine.

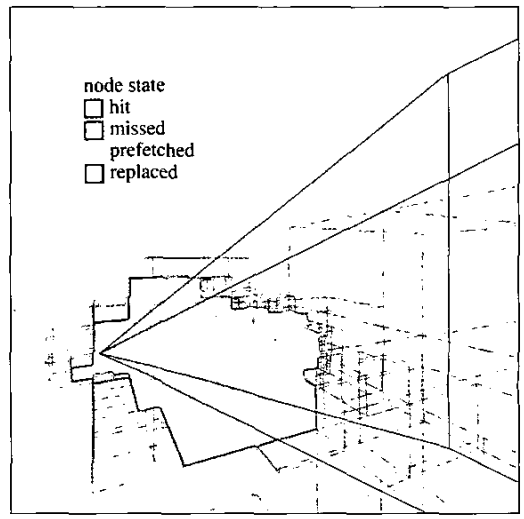
Our prefetching method exploits the fact that PLP can very quickly compute an approximate visible set. Given the current camera (Figure 2a), the look-ahead thread (Figure 2g) predicts the next camera position by simply extrapolating the current position and the camera's linear and angular speeds. More sophisticated prediction schemes could consider accelerations and several prior camera locations. For each predicted camera, the look-ahead thread uses PLP (Figure 2h) to determine which nodes the predicted camera is likely to see. For each node likely to be visible, the look-ahead thread sends a prefetch request to the geometry cache (Figure 2i). The geometry cache puts the prefetch requests in a queue and a set of prefetch threads process the requests. If there are no fetch requests pending, and if the maximum amount of geometry that can be prefetched per frame has not been reached, a prefetch thread will pop a request from the prefetch queue, and read the requested node from disk (if necessary) (Figure 2j). If the cache is full, the least recently used nodes are evicted from memory. The requested nodes are then placed in a queue of nodes that are ready to be rendered.

```
fetch(node, ready_queue)
{
    lock cache;
    while (node is busy)
        wait until node is free;
    mark node as busy;
    if (node is valid) {
        miss = false;
        update node position;
    } else {
        miss = true;
        allocate memory;
    }
    unlock cache;

    if (miss)
        read node;

    lock cache;
    if (miss)
        add node to cache;
    if (no fetches pending)
        broadcast no fetches pending;
    unlock cache;
    add node to ready_queue;
}

prefetch(node, ready_queue)
{
    lock cache;
    while (there are fetch requests pending)
        wait until no fetch requests pending;
    while (node is busy)
        wait until node is free;
    mark node as busy;
    if ((node is valid)
        || (reached max prefetch amount per frame)
        || (reached max prefetch request age))
        can_read = false;
    else {
        can_read = true;
        allocate memory;
    }
    unlock cache;

    if (can_read) {
        read node;
        lock cache;
        add node to cache;
        unlock cache;
    }
    add node to ready_queue;
}

release(node)
{
    lock cache;
    mark node as free;
    if (node is valid)
        broadcast memory available;
    broadcast node is free;
    unlock cache;
}
```

Figure 4: Pseudo-code for the main cache routines.



(a) user's view



(b) cache view

Figure 5: A sample frame inside the power plant model. (a) The image that the user sees. (b) The state of the nodes in the cache.

Figure 4 shows the pseudo-code for the main routines run by the threads in the cache. When a client makes a fetch request, a thread executes the fetch routine (and similarly for a prefetch request). When the client is done using that node, it must call the release routine. These routines have to be very careful about sharing the cache data structures. To guarantee mutual exclusion, there is a lock to access the cache, and each node has a flag indicating whether it is free or busy. This scheme is similar to the one used in the UNIX buffer cache [Bach 1986]. Figure 5a shows the user's view of the UNC power plant model [UNC 1999] during a walkthrough session, and Figure 5b shows the state of the octree nodes in the cache.

Unlike our from-point prefetching method, from-region prefetching methods decompose the model into cells, and precompute for each cell the geometry that the user would see from any point in the cell. At runtime, from-region methods guess in which cell the user will be next, and load the geometry visible from that cell ahead of time. Our from-point prefetching method has several advantages over from-region prefetching methods. First, from-region methods typically require long preprocessing times (tens of hours), while our from-point method requires little preprocessing (a few minutes). Second, the set of nodes visible from a single point is typically much smaller than the set of nodes visible from any point in a region. Thus, our from-point prefetching method avoids unnecessary disk operations, and has a better chance than a from-region method of prefetching nodes that actually will be visible soon. Third, some from-region methods require that cells coincide with axis-aligned polygons in the model. Our from-point method imposes no restriction on the model's geometry. Finally, the nodes visible from a cell may be very different from the nodes visible from a neighbor of that cell. Thus, a from-region method may cause bursts of disk activity when the user crosses cell boundaries, while a from-point method better exploits frame-to-frame coherence.

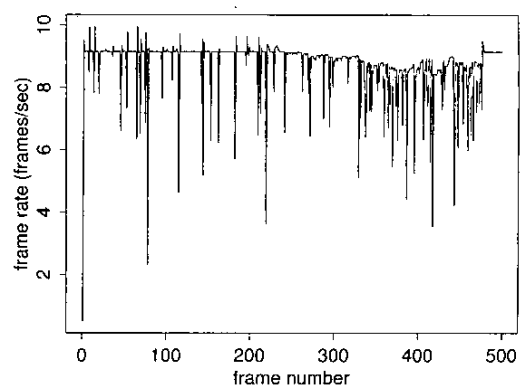
Since the cost of disk read operations is high, most systems try to overlap all of these operations with other computations by running several processes on a multiprocessor machine [Aliaga et al. 1999; Funkhouser 1996; Garlick et al. 1990], or on a network of machines [Wald et al. 2001; Wonka et al. 2001]. Along these same lines, our system uses multiple threads on a single processor machine to overlap disk operations with visibility computations and rendering.

6 Experimental Results

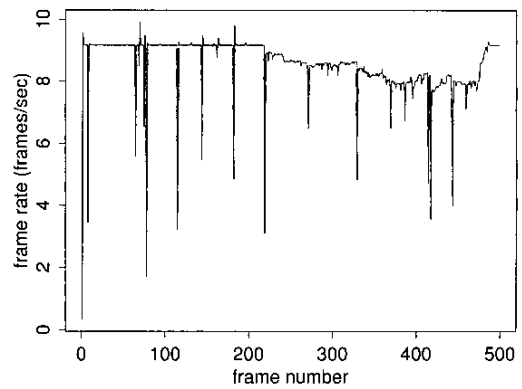
To evaluate our system, and in particular our prefetching algorithm, we experimented with the 13-million-triangle UNC power plant model [UNC 1999]. The raw model was roughly 600 MB in size; after our preprocessing step, the model size increased to 1 GB. To our knowledge, no other system has been able to render this model at interactive rates on a single PC. Our system ran Red Hat Linux 8.0, had a 2.8 GHz Pentium IV CPU, 512 MB of main memory, a 35 GB SCSI disk, and an nVidia Quadro 980 XGL card. Using `top`, we found that the operating system and related utilities used roughly 64 MB of main memory. For our test machine, we found that the following configuration worked well: 256 MB of geometry cache, 8 fetch threads, 1 prefetch thread, a maximum of 2 MB of prefetched geometry per frame, approximate visibility with a budget of 280,000 triangles per frame, a target frame rate of 10 fps, and image resolution of 1024×768.

To analyze the overall performance of our system, we measured the frame rates achieved when walking through the power plant model along several predefined paths (which enabled repeatable conditions for our experiments). Note that our algorithms made no assumptions on the paths being known beforehand; therefore, complete camera interactivity is always available to the user. The first path used has 36,432 viewpoints, visits almost every part of the model, and requires fetching a total of 900 MB of data from disk. Using the above configuration, our system rendered the frames along that path in 74 minutes. Only 95 frames (0.26%) caused the system to achieve less than 1 fps. The mean frame rate was 9.2 fps, and the median frame rate was 9.3 fps. To analyze the detailed performance of our system, it is easier to use shorter paths. For this purpose, we used a 500-frame path which required 210 MB of data to be read from disk. If fetched independently, the maximum amount of memory necessary to render any given frame in approximate mode would be 16 MB.

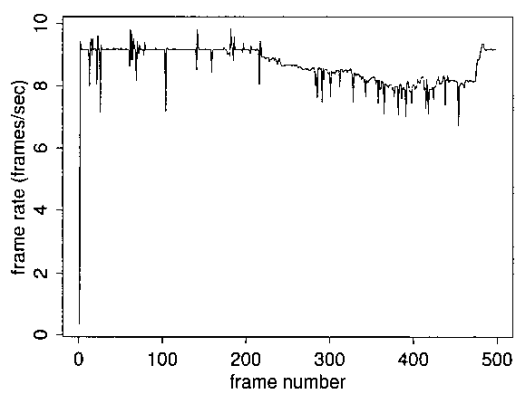
To study how multiple threads improve the frame rates, we ran tests using three different configurations. The first configuration



(a) sequential fetching and rendering



(b) concurrent fetching and rendering



(c) concurrent fetching, rendering, and prefetching

Figure 6: Using multiple threads to improve frame rates. We measured the frame rates during a 500-frame walkthrough of the power plant model under three configurations: (a) using one thread for fetching and rendering; (b) using multiple threads to overlap fetching and rendering; and (c) using multiple threads to overlap fetching, rendering, and prefetching. Concurrent fetching eliminates some downward spikes, and adding concurrent speculative prefetching eliminates almost all of the remaining spikes. The first spike happens because the cache is initially empty. The three configurations produce identical images.

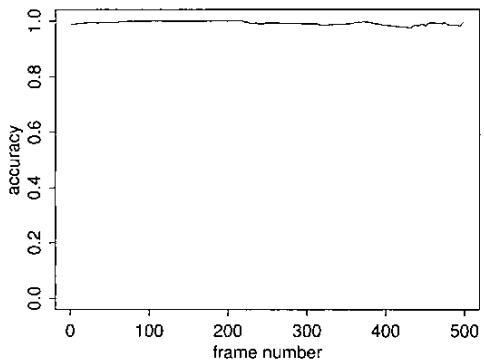
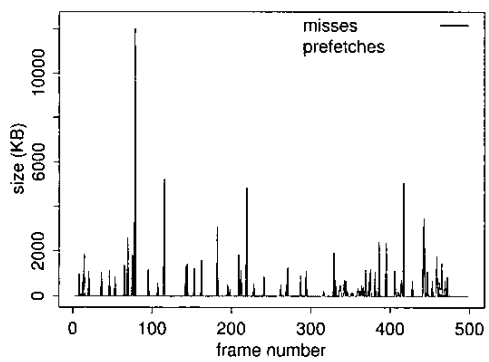
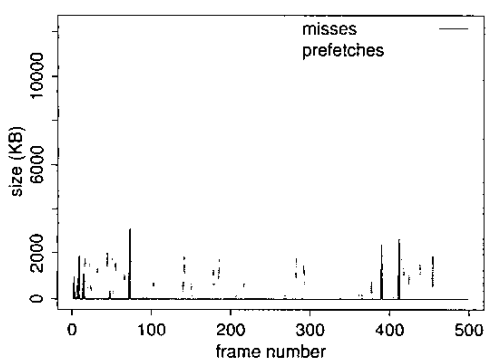


Figure 7: Image accuracy for a 500-frame walkthrough of the power plant model when using approximate visibility. The vertical axis represents the fraction of correct pixels in the approximate images in comparison to the conservative images. The minimum accuracy was 97.3%, and the median accuracy was 99.2%.



(a) without prefetching



(b) with prefetching

Figure 8: Using prefetching to amortize the cost of disk operations. We measured the amount of geometry fetched per frame without prefetching (a) and with prefetching (b). Prefetching amortizes the cost of bursts of disk operations over frames with few disk operations, thus eliminating most frame rate drops. The system was configured to prefetch at most 2 MB per frame.

is entirely sequential: a single thread is responsible for computing visibility, performing disk operations, and rendering. The second configuration adds asynchronous fetching to the first configuration, allowing up to 8 fetch threads. The third configuration adds an extra thread for speculative prefetching to the second configuration, allowing up to 2 MB of geometry to be prefetched per frame. Figure 6 shows the frame rates achieved by these three configurations for the 500-frame path. For the purely sequential configuration, we see many downward spikes that correspond to abrupt drops in frame rates, which are caused by the latency of the disk operations, and spoil the user's experience (The first spike happens because the cache is initially empty). When we add asynchronous fetching, many of the downward spikes disappear, but too many still remain. The user's experience is much better, but the frame rate drops are still disturbing. When we add speculative prefetching, all significant downward spikes disappear, and the user experience is smooth. Note that the gain in interactivity comes entirely from overlapping the independent operations. The three configurations achieve exactly the same image accuracy (Figure 7).

Figure 8 shows why prefetching improves the frame rates. The charts compare the amount of geometry that the system reads from disk per frame for the second and third configurations described above. Prefetching greatly reduces the need to fetch large amounts of geometry in a single frame, and thus helps the system to maintain higher and smoother frame rates.

Figure 9 shows that the user speed is another important parameter in the system, and has to be adjusted to the disk bandwidth. When the user speed increases, the changes in the visible set are larger. In other words, as the frame-to-frame coherence decreases, the amount of data the system needs to read per frame increases. Thus, caching and prefetching are more effective if the user moves at speeds compatible with the disk bandwidth. The figure also indicates that higher disk bandwidth should improve frame rates.

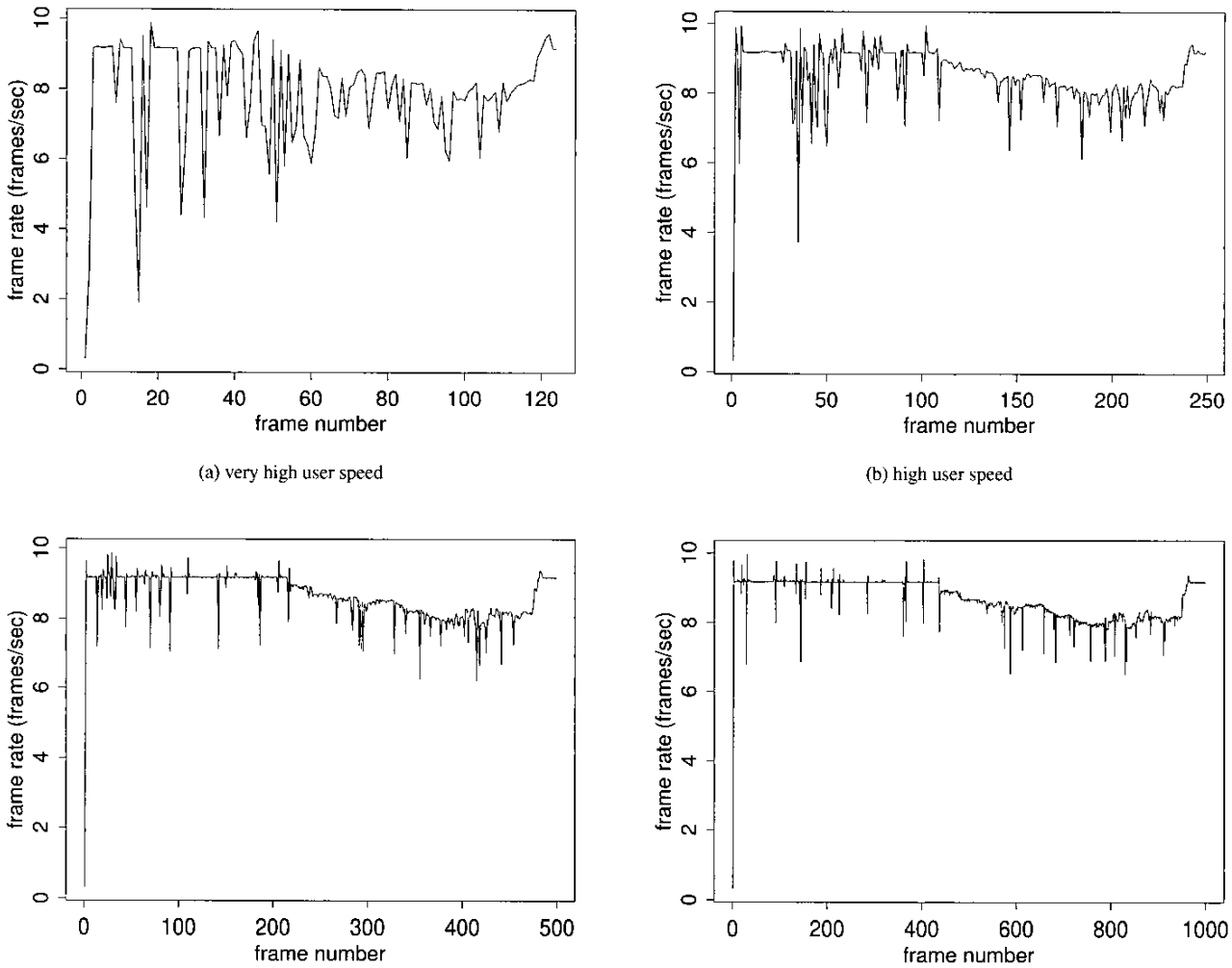
7 Conclusion

We have presented a system for rendering large models on machines with small memory at interactive frame rates. A key component of our out-of-core rendering approach is a new prefetching algorithm based on a from-point visibility algorithm. The prefetch algorithm accurately and efficiently predicts what geometry will be visible in subsequent frames and prefetches them from disk. We believe our system is the first to be able to preprocess the 13-million triangle UNC power plant model and render it interactively on a single PC.

One area of future work is adding level-of-detail (LOD) management to our entire system. In approximate mode, our system may produce images with low accuracy if the camera sees the entire model. El-Sana et al. [2001] show how to integrate LOD management with PLP-based occlusion culling. Another possible area for future work is speeding up rendering in conservative mode, which currently can be much slower than rendering in approximate mode. Finally, we also would like to extend the system to support dynamic scenes.

Acknowledgments

We thank Daniel Aliaga, David Dobkin, Juliana Freire, Thomas Funkhouser, Jeff Korn, Patrick Min, and Emil Praun for suggestions and encouragement. We also thank the University of North Carolina at Chapel Hill and the University of California at Berkeley for providing us with datasets. This research was partly funded by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brazil.



(a) very high user speed

(b) high user speed

(c) normal user speed

(d) low user speed

Figure 9: Adjusting the user speed to the disk bandwidth. We measured the frame rates along a camera path inside the power plant model for different user speeds (or equivalently, for different number of frames in the path). If the user moves too fast, the frame rates are not smooth. The faster the user moves, the larger the changes in occlusion, and therefore the larger the number of disk operations.

References

AIREY, J. M., ROHLF, J. H., AND FREDERICK P. BROOKS, J. 1990. Towards image realism with interactive update rates in complex virtual building environments. *1990 ACM Symposium on Interactive 3D Graphics 24*, 2 (Mar.), 41–50.

ALIAGA, D., COHEN, J., WILSON, A., ZHANG, H., ERIKSON, C., HOFF, K., HUDSON, T., STÜRZLINGER, W., BAKER, E., BASTOS, R., WHITTON, M., BROOKS, F., AND MANOCHA, D. 1999. MMR: An interactive massive model rendering system using geometric and image-based acceleration. *1999 ACM Symposium on Interactive 3D Graphics* (Apr.), 199–206.

AVILA, L. S., AND SCHROEDER, W. 1997. Interactive visualization of aircraft and power generation engines. In *IEEE Visualization '97*, IEEE, 483–486.

BACH, M. J. 1986. *The Design of the UNIX Operating System*. Prentice Hall.

CHIANG, Y.-J., AND SILVA, C. T. 1997. I/O optimal isosurface extraction. *IEEE Visualization '97* (Nov.), 293–300.

CHIANG, Y.-J., SILVA, C. T., AND SCHROEDER, W. J. 1998. Interactive out-of-core isosurface extraction. *IEEE Visualization '98* (Oct.), 167–174.

CIGNONI, P., ROCCHINI, C., MONTANI, C., AND SCOPIGNO, R. 2002. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*.

CLARK, J. H. 1976. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM 19*, 10 (Oct.), 547–554.

CORRÊA, W. T., KLOSOWSKI, J. T., AND SILVA, C. T. 2002. iWalk: Interactive out-of-core rendering of large models. Technical Report TR-653-02, Princeton University. Available at: http://www.cs.princeton.edu/~wtcorrea/papers/iwalk.pdf.

COX, M. B., AND ELLSWORTH, D. 1997. Application-controlled demand paging for out-of-core visualization. *IEEE Visualization '97* (Nov.), 235–244.

DURAND, F., DRETTAKIS, G., THOLLOT, J., AND PUECH, C. 2000. Conservative visibility preprocessing using extended projections. In *Proceedings of Siggraph 2000*, 239–248.

EL-SANA, J., AND CHIANG, Y.-J. 2000. External memory view-dependent simplification. *Computer Graphics Forum 19*, 3 (Aug.), 139–150.

EL-SANA, J., SOKOLOVSKY, N., AND SILVA, C. T. 2001. Integrating occlusion culling with view-dependent rendering. In *IEEE Visualization 2001*, 371–378.

ERIKSON, C., MANOCHA, D., AND BAXTER III, W. V. 2001. HLODs for faster display of large static and dynamic environments. In *2001 ACM Symposium on Interactive 3D Graphics*, 111–120.

FUNKHOUSER, T. A., SÉQUIN, C. H., AND TELLER, S. J. 1992. Management of large amounts of data in interactive building walkthroughs. *1992 ACM Symposium on Interactive 3D Graphics 25*, 2 (Mar.), 11–20.

FUNKHOUSER, T. A. 1996. Database management for interactive display of large architectural models. *Graphics Interface '96* (May), 1–8.

GARLICK, B. J., BAUM, D. R., AND WINGET, J. M. 1990. Interactive viewing of large geometric databases using multiprocessor graphics workstations. In *Siggraph Course: Parallel Algorithms and Architectures for 3D Image Generation*. ACM Siggraph, 239–245.

GINDELE, B. S. 1977. Buffer block prefetching method. *IBM Technical Disclosure Bulletin 20*, 2, 696–697.

KLOSOWSKI, J. T., AND SILVA, C. T. 2000. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (Apr.-June), 108–123.

KLOSOWSKI, J. T., AND SILVA, C. T. 2001. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics 7*, 4 (Oct.-Dec.), 365–379.

PHARR, M., KOLB, C., GERSHBEIN, R., AND HANRAHAN, P. 1997. Rendering complex scenes with memory-coherent ray tracing. *Proceedings of Siggraph 97* (Aug.), 101–108.

PRZYBYLSKI, S. A. 1990. *Cache and Memory Hierarchy Design: A Performance-Directed Approach*. Morgan Kaufmann.

REGE, A., 2002. Occlusion extensions. http://developer.nvidia.com/docs/-IO/2645/ATT/GDC2002_occlusion.pdf.

SAMET, H. 1990. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley.

SCHAUFLER, G., DORSEY, J., DECORET, X., AND SILLION, F. X. 2000. Conservative volumetric visibility with occluder fusion. In *Proceedings of Siggraph 2000*, 229–238.

SEVERSON, K., 1999. VISUALIZE workstation graphics for Windows NT. HP product literature.

SHEN, H.-W., CHIANG, L.-J., AND MA, K.-L. 1999. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. *IEEE Visualization '99* (Oct.), 371–378.

SUTTON, P. M., AND HANSEN, C. D. 2000. Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (Apr.-June), 98–107.

TELLER, S. J., AND SÉQUIN, C. H. 1991. Visibility preprocessing for interactive walkthroughs. *Proceedings of Siggraph 91 25*, 4 (July), 61–69.

UENG, S.-K., SIKORSKI, C., AND MA, K.-L. 1997. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics 3*, 4 (Oct.-Dec.), 370–380.

UNC, 1999. Power plant model. http://www.cs.unc.edu/~geom/-Powerplant/. The Walkthru Group at UNC Chapel Hill.

VARADHAN, G., AND MANOCHA, D. 2002. Out-of-core rendering of massive geometric environments. In *IEEE Visualization 2002*.

WALD, I., SLUSALLEK, P., AND BENTHIN, C. 2001. Interactive distributed ray tracing of highly complex models. *Rendering Techniques 2001*, 277–288.

WONKA, P., WIMMER, M., AND SCHMALSTIEG, D. 2000. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Rendering Techniques 2000*, 71–82.

WONKA, P., WIMMER, M., AND SILLION, F. 2001. Instant visibility. *Computer Graphics Forum 20*, 3, 411–421.