Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Parallel Computing 29 (2003) 325–338

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Out-of-core sort-first parallel rendering for cluster-based tiled displays

Wagner T. Corrêa [a,*], James T. Klosowski [b], Cláudio T. Silva [c]

[a] *Princeton University, Princeton, NJ 08540, USA*
[b] *IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA*
[c] *Oregon Health and Science University, Beaverton, OR 97006, USA*

Received 22 April 2002; accepted 31 October 2002

Abstract

We present a sort-first parallel system for out-of-core rendering of large models on cluster-based tiled displays. The system renders high-resolution images of large models at interactive frame rates using off-the-shelf PCs with small memory. Given a model, we use an out-of-core preprocessing algorithm to build an on-disk hierarchical representation for the model. At run time, each PC renders the image for a display tile, using an out-of-core rendering approach that employs multiple threads to overlap rendering, visibility computation, and disk operations. The system can operate in approximate mode for real-time rendering, or in conservative mode for rendering with guaranteed accuracy. Running our system in approximate mode on a cluster of 16 PCs each with 512 MB of main memory, we are able to render 12-megapixel images of a 13-million-triangle model with 99.3% of accuracy at 10.8 frames per second. Rendering such a large model at high resolutions and interactive frame rates would typically require expensive high-end graphics hardware. Our results show that a cluster of inexpensive PCs is an attractive alternative to those high-end systems.

Keywords: Parallel rendering; Out-of-core rendering; Cluster of PCs

1. Introduction

No matter how fast graphics hardware technology advances, the demand for power to render larger and more complex models always seems to outpace it. Visualization

* Corresponding author.
E-mail address: wtcorrea@cs.princeton.edu (W.T. Corrêa).

0167-8191/02/$ - see front matter
doi:10.1016/S0167-8191(02)00249-1

of large models has applications in many areas including medicine, engineering, weather forecasting, and entertainment. Traditionally, visualizing large models would require multi-million-dollar, high-end, special-purpose parallel machines.

Recently, due to the explosive growth in rendering capabilities of PC graphics cards and the availability of high-speed networks, using a cluster of PCs has become an attractive alternative to high-end systems. PC clusters have a better price/performance ratio than high-end systems; they may be upgraded more frequently; they can employ different kinds of machines; they may be used for tasks other than rendering; and their aggregate power scales with the number of machines [19,28].

In this paper, we present a parallel rendering system that exploits the power of a cluster of inexpensive PCs to render high-resolution images of large models at interactive frame rates on a tiled display. We focus on truly large models that cannot fit into the small main memory of a typical PC. The system we present here is a parallel extension of the *iWalk* system [7]. Although iWalk is able to handle models larger than main memory, it only produces low-resolution (1024×768) images at interactive frame rates. The parallel system we describe here uses iWalk as a building block, and delivers high-resolution (4096×3072) images at interactive frame rates. Given a large model, we use iWalk's out-of-core preprocessing algorithm to build an on-disk hierarchical representation for the model. At run time, each machine in the cluster renders the image for a display tile, using iWalk's out-of-core rendering approach that employs multiple threads to overlap rendering, visibility computation, and disk operations.

In Section 2, we review related work. In Section 3, we provide an overview of the iWalk system, describing its preprocessing and rendering algorithms. In Section 4, we show how to use these algorithms in a parallel system to render high-resolution images of a large model at interactive frame rates on a cluster-based tiled display. We present our experimental results in Section 5. We conclude and discuss directions for future work in Section 6.

2. Related work

Samanta et al. [29,30] developed a sort-first rendering system using a network of commodity PCs. The focus of their work was on load balancing the work done on each of the PCs, rather than on handling very large models. To achieve a well balanced system, they developed dynamic screen partitioning schemes that predict the rendering costs of groups of triangles and attempt to minimize the amount of overlap between triangles and screen partitions. A limitation of their system was that in some cases the screen partitioning scheme could become the bottleneck. Another limitation was the lack of scalability with respect to model size, as the model had to be replicated in main memory on each of the nodes of their cluster.

In subsequent work, Samanta et al. [28] developed a hybrid sort-first/sort-last parallel rendering algorithm, which scaled better with processor count and screen resolution. Their new approach performs dynamic, view-dependent partitioning of both the 3D model and the 2D screen. The objectives that they are addressing are balanc-

ing the rendering load on the nodes as well as minimizing the screen space overlaps which require the subsequent pixel transfer and compositing step. Once again, the geometry is replicated on each of the nodes, and the dynamic partitioning phase could become a bottleneck and limit the frame rate.

In more recent work, Samanta et al. [27] address the replication problem, storing (in main memory) copies of the model only in k of the available n nodes, where $k < n$. Still, neither their preprocessing phase nor their rendering phase would be able to handle a model larger than the total amount of main memory in their cluster. The system we present in this paper can handle arbitrarily large models (limited only by the size of the available secondary memory).

WireGL [5,14–16] is a system that allows the output resolution of an unmodified graphics application to be scaled to the resolution of a tiled display, with little or no loss in performance. WireGL replaces the OpenGL driver on the client machine, intercepts the OpenGL calls, and sends the calls over a high-speed network to servers which render the geometry. WireGL includes an efficient network protocol, a geometry bucketing scheme, and an OpenGL state tracking algorithm, which makes the interactive performance possible. WireGL is able to sustain rendering performance of over 70,000,000 triangles per second on a 32-node cluster. It assumes, however, that the entire model fits in the main memory of the client machine. Another limitation is that the geometry bucketing algorithm assumes that the geometry primitives that are close to each other in the GL stream are also close together spatially, which may not be the case.

Mueller [25,26] has performed extensive experiments using a sort-first rendering system. He emphasizes that sort-first has an advantage over sort-middle, because it can exploit the frame-to-frame coherence inherent in interactive applications. He also points out that sort-first has an advantage over sort-last, because it does not require high communication bandwidth for pixel traffic. Mueller designed a dynamic scheme for partitioning the screen so that each processor has a balanced rendering load. His algorithm is the basis for the work of Samanta et al. [29] Mueller focused his work on retained-mode databases that fit in the memory of the graphics hardware. In contrast, we focus our work on immediate-mode databases that are larger than the main memory of the host hardware.

Wald et al. [34] have developed a ray tracing system for out-of-core rendering of large models on a cluster of PCs. A key difference between our work and theirs is that they use ray tracing, and we use the Z-buffer. Although ray tracing allows them to use more sophisticated rendering algorithms, the Z-buffer allows us to exploit better hardware support. Another difference is that our preprocessing algorithm is an order of magnitude faster than theirs [7]. Finally, they create 640×480-pixel images, while our images have 4096×3072 pixels.

Aliaga et al. [3] have developed the massive model rendering (MMR) system. MMR replaces geometry that is far from the user's point of view with textured depth meshes (TDMs), which are image impostors that contain depth information, and are displayed using projective texture mapping. MMR also employs occlusion culling and detail elision. We believe MMR was the first system to handle models with tens of millions of polygons at interactive frame rates. One disadvantage of MMR is the

preprocessing step, which requires user intervention and takes weeks to run. In contrast, the preprocessing step of iWalk is fully automatic and takes only a few minutes to run. Another difference between MMR and iWalk is that MMR is designed for large SGI multi-processor machines with several gigabytes of main memory, while iWalk is designed for off-the-shelf PCs with small memory.

Funkhouser et al. [12] developed a system that supported models larger than main memory and performed speculative prefetching. Their system used the from-region visibility algorithm of Teller et al. [32], which requires long preprocessing times and assumes that the models are made of axis-aligned cells. Funkhouser et al. [10,11] made improvements to their original system, but the preprocessing stage remained limited to models made of axis-aligned cells. iWalk uses the from-point visibility algorithm of Klosowski and Silva [17,18], which requires very little preprocessing and makes no assumptions about the geometry of the model.

Other research projects involving parallel rendering using commodity components that are of interest include the work of Lombeyda et al. [20] and Zhang et al. [36].

3. Overview of iWalk

The iWalk system [7] lets a user walk through a large model at interactive rates using a PC with small memory. The main parts of iWalk are the out-of-core preprocessing algorithm and the out-of-core multi-threaded rendering approach.

The out-of-core preprocessing algorithm creates an on-disk hierarchical representation of the input model. More specifically, it creates an octree [31] whose leaves contain the geometry of the input model. 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. To store the octree on disk, the preprocessing algorithm saves the geometric contents of each octree node in a separate file. The preprocessing algorithm also creates a *hierarchy structure* (HS) file. The HS file has 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. We assume that the HS file fits in memory, which is usually a trivial assumption. For example, the size of the HS file for a 13-million-triangle model is only 1 MB.

At run time, iWalk uses an out-of-core multi-threaded rendering approach. A rendering thread uses the PLP [17] algorithm to determine the set of octree nodes that are visible from the user's point of view. For each visible node, the rendering thread sends a fetch request to the fetch threads, which will process the request, and bring the contents of the node from disk into a memory cache. If the cache is full, the least recently used node in the cache is evicted from memory. To minimize the chance of I/O bursts, there is a look-ahead thread that runs concurrently with the rendering thread. The look-ahead thread tries to predict where the user is going to be in the next few frames, uses PLP to determine the nodes that the user would see then, and sends prefetch requests to the prefetch threads. If there are no fetch requests

pending, the prefetch threads will bring the requested nodes into memory (up to certain limit per frame based on the disk's bandwidth). This speculative prefetching scheme amortizes the bursts of I/O over frames that require little or no I/O, and produces faster and smoother frame rates.

The rendering thread and the look-ahead thread both use PLP [17] to determine the nodes that are visible from a given point. PLP is an approximate, from-point visibility algorithm that may be understood as a set of modifications to the traditional hierarchical view frustum culling algorithm [6]. 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 the node to the *visible set*, removes the node 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. 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.

The key feature of PLP that iWalk exploits is that PLP can generate an approximate visible set based only on the information stored in the HS file created at preprocessing time. In other words, PLP can estimate the visible set *without* access to the actual scene geometry.

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 rendering thread may optionally use cPLP [18], a conservative extension of PLP that guarantees 100% accurate images. On the other hand, cPLP cannot determine the visible set from the HS file only, and needs to read the geometry of all potentially visible nodes. The additional I/O operations make cPLP much slower than PLP.

4. Parallelizing iWalk

When interacting with large models, it is natural to want to visualize these models at high resolution. The iWalk system can only produce low-resolution images (1024×768 pixels) at interactive frame rates. We now show how to use iWalk as a building block for a parallel system that produces high-resolution images (4096×3072 pixels) at the same or faster rates.

4.1. Choosing the hardware

A traditional approach to parallel rendering has been to use a high-end parallel machine. More recently, with the explosive growth in power of inexpensive graphics

cards for PCs, and the availability of high-speed networks, using a cluster of PCs for parallel rendering has become an attractive alternative to high-end systems. A PC cluster has many advantages over high-end systems [19,28]:

Lower cost: A cluster of commodity PCs, each costing a few thousand dollars, typically has a better price/performance ratio than a high-end, highly-specialized supercomputer that may cost up to millions of dollars.

Technology tracking: High-volume off-the-shelf parts tend to improve at faster rates than special-purpose hardware. We can upgrade a cluster of PCs much more frequently than a high-end system, as new inexpensive PC graphics cards become available every 6–12 months.

Modularity and flexibility: We can easily add or remove machines from the cluster, and even mix machines of different kinds. We can also use the cluster for tasks other than rendering.

Scalable capacity: The aggregate computing, storage, and bandwidth capacity of a PC cluster grows linearly with the number of machines in the cluster.

Thus we have chosen to use a cluster of PCs to drive a multi-projector tiled display to create high-resolution images.

4.2. Choosing the parallelization strategy

Many approaches to parallel rendering have been proposed over the years. Molnar et al. [22] classify parallelization strategies in three categories based on where in the rendering pipeline sorting for visible-surface determination takes place. Sorting may happen during geometry preprocessing, between geometry preprocessing and rasterization, or during rasterization. The three categories of parallelization strategies are sort-first, sort-middle, and sort-last.

Sort-first algorithms [15,25,29,30] distribute raw primitives (with unknown screen-space coordinates) during geometry preprocessing. These approaches divide the 2D screen into disjoint regions (or tiles), and assign each region to a different processor, which is responsible for all of the rendering in its region. For each frame, a pretransformation step determines the regions in which each primitive falls. Then a redistribution step transfers the primitives to the appropriate renderers. Sort-first approaches take advantage of frame-to-frame coherence well, since few primitives tend to move between tiles from one frame to the next. Sort-first algorithms can also use any rendering algorithm, since each processor has all the information it needs to do a complete rendering. Furthermore, as rendering algorithms advance, sort-first approaches can take full advantage of them. One disadvantage of sort-first is that primitives may cluster into regions, causing load balancing problems between renderers. Another disadvantage is that more than one renderer may process the same primitive if it overlaps screen region boundaries.

Sort-middle algorithms [2,9,24] distribute screen-space primitives between the geometry preprocessing and rasterization stages. Sort-middle approaches assign a subset of primitives to each geometry processor, and a portion of the screen to each rasterizer. A geometry processor transforms and lights its primitives, and then sends them to the appropriate rasterizers. Until recently, this approach has been the most

popular due to the availability of high-end graphics machines. Sort-middle approaches may suffer from load imbalance between rasterizers when primitives are distributed unevenly over the screen. Sort-middle also requires high bandwidth for the transfer of data between the geometry processing and rasterization stages.

Sort-last approaches [13,23,35] distribute pixels during rasterization. They assign a subset of the primitives to each renderer. A renderer computes pixel values for its subset, no matter where they fall in the screen, and then transfer these pixels (color and depth values) to compositing processors. Sort-last approaches scale well with respect to the number of primitives, since they render each primitive exactly once, but they need a high bandwidth network to handle all the pixel transfers. Another disadvantage of sort-last approaches is that they only determine the final depth of a pixel during the composition phase, and therefore make it difficult (if not impossible) to use certain rendering algorithms, e.g., transparency and anti-aliasing.

Given our goal and constraints, we have chosen a sort-first approach for two main reasons. First, sort-first processors implement the entire pipeline for a portion of the screen [22], which is exactly the case for which PC graphics cards are optimized. And second, walkthrough applications tend to exhibit high frame-to-frame coherence, which sort-first approaches exploit well. We rejected sort-middle approaches because they require a tight integration between the geometry processing and rasterization stages, which is only available on high-end graphics machines [2,9,24]. On PC graphics cards, there is no fast access to the results of the geometry processing [28]. We rejected sort-last approaches because they require extremely high pixel bandwidth [22], which is not yet available on PC graphics cards.

4.3. Our parallel rendering system

To implement a sort-first approach, the main challenge is to handle the redistribution step [27]. During the geometry processing, after a pretransformation step determines into which screen tiles each primitive falls, the primitives must become available in main memory at the renderers responsible for those tiles. To get around the redistribution step, some systems simply replicate in main memory the entire model on each renderer. This approach, of course, does not scale with respect to model size. More sophisticated systems replicate the model only on a subset of the renderers [27]. Our system keeps a hierarchical representation of the model on disk, and each renderer loads the visible parts of the model into its memory cache on demand. Since the disk where we keep the model may be a shared network disk or a local disk, this approach imposes virtually no limit on the model size.

Fig. 1 shows a diagram of our system. A client machine is responsible for processing user interface events. For each display tile there is a dedicated rendering server. At each frame, the client sends the current viewing parameters to the rendering servers. Note that the client does essentially no work. The rendering servers run the sequential rendering algorithms (from iWalk) that we presented in Section 3, with a few modifications that we will discuss below. Each renderer reads the parts of the model it needs from a shared network disk in the file server, and sends the resulting image to one of the display projectors. Optionally, each renderer may read its

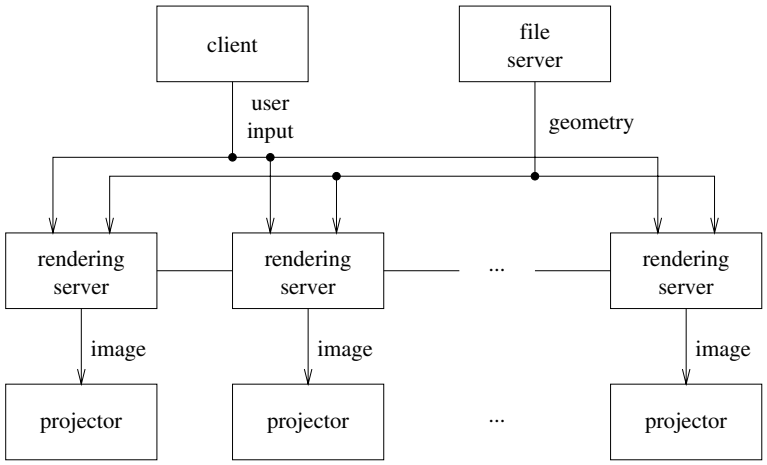

Fig. 1. The out-of-core sort-first architecture.

primitives from a local copy of the model. Note that this copy is on disk, not in main memory. Since disk space is cheap, having a local copy of the model on disk might not hurt the scalability of the system.

Each rendering server is an MPI task and runs basically the same code that iWalk runs, with a few differences. First, since each renderer is responsible for a tile of the display wall, it performs occlusion culling using only the part of view frustum that belongs to it. Second, each renderer receives input events from the client through socket communication, instead of directly from the user. Finally, to synchronize the renderers, we add an MPI barrier at the end of the rendering loop, right before swapping front and back buffers.

We only use MPI to start and synchronize the servers. The client does not need to have an MPI implementation available. The client machine only transmits the current viewing parameters to the rendering servers, and may therefore be as lightweight as a handheld computer. Some systems perform load balancing computations on the client machine, in which case the client may become a bottleneck [30].

5. Experimental results

We ran many experiments to evaluate the performance and the scalability of our system. Each experiment consisted of rendering the 13-million-triangle UNC power plant model [33], shown in Fig. 2, for a prerecorded camera path of 500 frames.

For each test, we collected statistics for both approximate visibility mode (using PLP) and conservative visibility mode (using cPLP). We ran tests on clusters of 1, 2, 4, 8, and 16 rendering servers. Each rendering server is a 900 MHz AMD Athlon with 512 MB of main memory, an nVidia geForce2 graphics card, and an IDE hard disk. For each cluster size, we first ran the tests storing the model at a shared

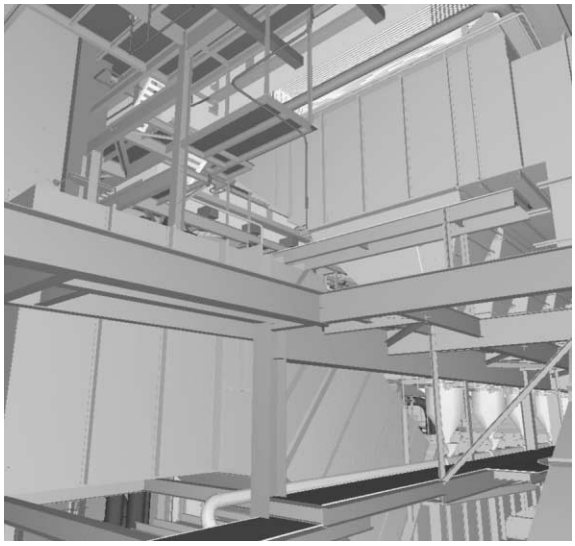

Fig. 2. The 13-million-triangle power plant model. Courtesy of UNC Chapel Hill [33].

network disk on a file server, and then storing copies of the model at local disks. The file server has a 400 GB disk array composed of eight SCSI disks configured as two 200 GB striped disks. As we have discussed, the client machine does very little processing, and therefore its hardware details are not important. In our tests, the client machine was a 700 MHz Pentium III. The network connecting the machines is a switched gigabit ethernet. All machines run Red Hat Linux 7.2. The servers use MPI/Pro 1.6.3, running on top of TCP/IP, for synchronization.

5.1. PLP results

Here we report the results of the experiments we ran in approximate visibility mode, i.e., using PLP to estimate the visible geometry. In typical use, we configure the system according to the triangle throughput of the graphics cards, the bandwidth of the disks, the desired frame rate, and the desired image accuracy. When using a cluster of 16 rendering servers, we usually give each renderer a budget of 70,000 triangles per frame and a geometry cache of 256 MB. This configuration allows us to generate 12-megapixel images of the power plant with a median accuracy of 99.3% at a median frame rate of 10.8 fps. For the scalability analysis that follows, we used instead a total budget of 400K triangles per frame, so that the system would be usable even when configured with only one rendering server.

When we run our system in approximate mode on a single machine, the frame rates depend mostly on the number of triangles rendered and the number of disk accesses; the image resolution has a smaller influence. As we add more machines to the cluster, the total resolution increases, but the resolution of each renderer remains

fixed. The total triangle budget per frame for PLP also remains fixed, thus the triangle budget of each renderer decreases.

Ideally, if we doubled the number of machines in the cluster, we would get twice the frame rate and the same image quality. In practice, several factors prevent us from achieving that. First, there is duplication of effort. In sort-first, if a primitive overlaps multiple tiles, it is fetched and rendered multiple times. Since the chances of overlap increase as we add processors, the demands for I/O bandwidth and triangle throughput also increase. There are additional communications costs as well. At the end of each frame, there is an MPI barrier to synchronize all the servers. Finally, the likelihood of load imbalance increases as the number of processors increases, which may have a negative effect on both the frame rate and the image accuracy.

Fig. 3a shows the frame rates achieved by our system when using PLP, as we vary the cluster size (1, 2, 4, 8, and 16 PCs) and the type of disk (network or local). [1] For these small clusters, the median frame rates (the horizontal lines in the interior of the boxes) improved substantially with the number of PCs. On the other hand, the spread of the frame rates (the height of the boxes) increased. For all configurations, there were very few stalls (the horizontal lines outside the whiskers). A surprising fact is that the disk type has almost no influence on the frame rates. The bandwidth of our network disk, measured using the Bonnie benchmark [4] from a rendering server, is 7.8 MB/s. The similarity between the frame rates indicates that the total bandwidth required by the rendering servers is usually less than the bandwidth of the network disk. We believe the bandwidth requirement is so low because our caching and prefetching schemes are exploiting well the frame-to-frame coherence in our test paths.

We measured the accuracy achieved by our system for the tests above by comparing the pixels in the images produced by PLP and the pixels in the correct images. For this particular camera path, which was inside the power plant, in an area with high depth-complexity, PLP estimates the visible set very well. For a single machine, PLP achieves a median accuracy of 99.6%. If the triangles were uniformly distributed across the screen, for a constant total triangle budget B, a cluster with $P > 1$ rendering servers, each of which with a triangle budget of B/P to render its screen tile, would achieve the same accuracy as a single machine. But typically the distribution of the triangles is not uniform, and B/P triangles may be too few for some servers and too many for others. For paths inside the model, this load imbalance is usually small, and the accuracy drops slowly with the cluster size. For the test path, the median accuracy achieved by the cluster with 16 servers was 93%, which is high and typical for paths inside the model. For paths outside the model, the accuracy may be

[1] *How to read the box plots*. The horizontal line in the interior of the box is located at the median of the data, and estimates the center of the data. The height of the box is equal to the *interquartile distance*, or IQD, which is the difference between the third quartile of the data and the first quartile, and indicates the spread of the data. The whiskers (the dotted lines extending from the top and bottom of the box) extend to the extreme values of the data or a distance of $1.5 \times$ IQD from the center, whichever is less. For data having a Gaussian distribution, approximately 99.3% of the data falls inside the whiskers. Data points that fall outside the whiskers may be outliers, and are indicated by horizontal lines [21].

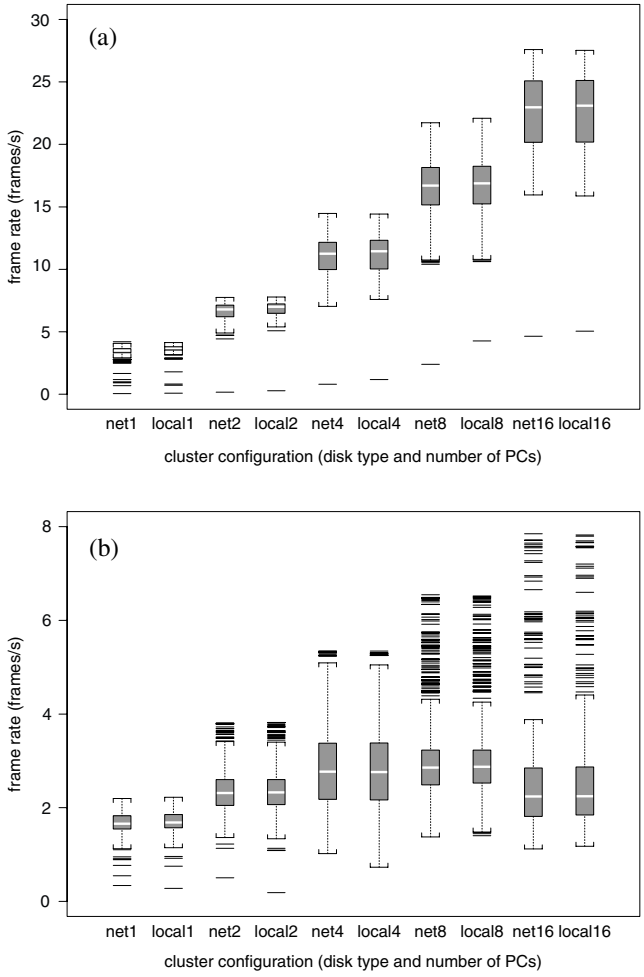

Fig. 3. Frame rates for PLP and cPLP as we vary the cluster size and the disk type. We ran tests on clusters of 1–16 PCs, for network and local disks. For PLP (a), the median frame rates improve substantially with the number of PCs. For cPLP (b), the median frame rates stay roughly the same. In both cases, the disk type makes almost no difference.

much lower, because we have not yet added level-of-detail management to our system.

5.2. cPLP results

Here we report the results of the experiments we ran in conservative visibility mode, i.e., using cPLP to estimate the visible geometry. Conservative visibility introduces another obstacle for ideal scalability. Remember that there is a one-to-one

correspondence between servers and projectors. Thus, when you increase the number of servers, although each server becomes responsible for a smaller part of the view frustum, that part will be rendered at higher resolution. As a result, the amount of geometry visible through that part of the view frustum that we need to fetch and render may not decrease. In theory, it could even increase. Since the size of the problem may grow with the cluster size, we cannot expect linear scalability. Adding level-of-detail management to our system would address this problem.

Fig. 3b shows the frame rates achieved by our system when using cPLP, as we vary the cluster size and the type of disk. Recall that PLP can estimate a visible set based only on the hierarchy structure file created at preprocessing time, but cPLP needs to read the actual scene geometry. Thus cPLP needs to perform many more disk accesses than PLP, and the frame rates for cPLP are much lower than those for PLP. In terms of scalability, even though the *maximum* frame rates increase substantially with cluster size, the *median* frame rates remain roughly the same. In terms of disk type, the network disk was able to match once again the performance of the local disks, which indicates that making local copies of the model on each server may be unnecessary.

6. Conclusion

We have presented a scalable system that renders high-resolution images of large models using a cluster of PCs to drive a tiled display. By employing out-of-core algorithms for preprocessing and rendering, the system is able to handle models much larger than the main memory of a typical commodity PC. These algorithms are easy to implement, run fast, and make no assumption about the input models. The system exploits frame-to-frame coherence well, and is able to render these large models at interactive frame rates. We are therefore able to use inexpensive off-the-shelf components to visualize models that would traditionally require expensive high-end graphics hardware.

There are several avenues for future work. First, we intend to add level-of-detail management [1,6,8,11] to the system. Second, we also intend to add load balancing schemes [25,28–30]. Third, we want to investigate better PLP heuristics to estimate and propagate solidity. Finally, all the algorithms we use are based on the assumption that the model is static. Extending these algorithms to handle dynamic scenes is still an unexplored area of research.

Acknowledgements

We thank the University of North Carolina at Chapel Hill for the power plant model. We also thank Daniel Aliaga, David Dobkin, Thomas Funkhouser, Jeff Korn, Wagner Meira, and Emil Praun for encouragement and helpful suggestions. This research was partly funded by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), Brazil.

References

[1] J.M. Airey, J.H. Rohlf, F.P. Brooks, Jr., Towards image realism with interactive update rates in complex virtual building environments, in: 1990 Symposium on Interactive 3D Graphics, vol. 24 (2), 1990, pp. 41–50.

[2] K. Akeley, RealityEngine graphics, in: Proceedings of SIGGRAPH 93, 1993, pp. 109–116.

[3] D. Aliaga, J. Cohen, A. Wilson, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stürzlinger, E. Baker, R. Bastos, M. Whitton, F. Brooks, D. Manocha, MMR: an interactive massive model rendering system using geometric and image-based acceleration, in: 1999 ACM Symposium on Interactive 3D Graphics, 1999, pp. 199–206.

[4] T. Bray, The Bonnie file system benchmark, http://www.textuality.com/bonnie/.

[5] I. Buck, G. Humphreys and P. Hanrahan, Tracking graphics state for networked rendering. 2000 SIGGRAPH/Eurographics Workshop on Graphics Hardware, 2000, pp. 87–96.

[6] J.H. Clark, Hierarchical geometric models for visible surface algorithms, Communications of the ACM 19 (10) (1976) 547–554.

[7] W.T. Corrêa, J.T. Klosowski, C.T. Silva, iWalk: Interactive out-of-core rendering of large models. Technical Report TR-653-02, Princeton University, 2002.

[8] J. El-Sana, N. Sokolovsky, C.T. Silva, Integrating occlusion culling with view-dependent rendering, in: IEEE Visualization 2001, October 2001, pp. 371–378.

[9] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, L. Israel, Pixel-planes 5: a heterogeneous multiprocessor graphics system using processor-enhanced memories, Computer Graphics (Proceedings of SIGGRAPH 89) 23 (3) (1989) 79–88.

[10] T.A. Funkhouser, Database management for interactive display of large architectural models. Graphics Interface '96, 1996, pp. 1–8.

[11] T.A. Funkhouser, C.H. Séquin, Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments, in: Proceedings of SIGGRAPH 93, 1993, pp. 247–254.

[12] T.A. Funkhouser, C.H. Séquin, S.J. Teller, Management of large amounts of data in interactive building walkthroughs, in: 1992 Symposium on Interactive 3D Graphics, vol. 25(2), 1992, pp. 11–20.

[13] A. Heirich, L. Moll, Scalable distributed visualization using off-the-shelf components, in: Symposium on Parallel Visualization and Graphics, 1999, pp. 55–59.

[14] G. Humphreys, I. Buck, M. Eldridge, P. Hanrahan, Distributed rendering for scalable displays, in: Proceedings of IEEE Supercomputing 2000, 2000.

[15] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, P. Hanrahan, WireGL: a scalable graphics system for clusters, in: Proceedings of SIGGRAPH 2001, 2001, pp. 129–140.

[16] G. Humphreys, P. Hanrahan, A distributed graphics system for large tiled displays, in: Proceedings of IEEE Visualization '99, 1999, pp. 215–223.

[17] J.T. Klosowski, C.T. Silva, The prioritized-layered projection algorithm for visible set estimation, IEEE Transactions on Visualization and Computer Graphics 6 (2) (2000) 108–123.

[18] J.T. Klosowski, C.T. Silva, Efficient conservative visibility culling using the prioritized-layered projection algorithm, IEEE Transactions on Visualization and Computer Graphics 7 (4) (2001) 365–379.

[19] K. Li, H. Chen, Y. Chen, D.W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, T. Housel, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J.P. Singh, G. Tzanetakis, J. Zheng, Early experiences and challenges in building and using a scalable display wall system, IEEE Computer Graphics and Applications 25 (4) (2000) 671–680.

[20] S. Lombeyda, L. Moll, M. Shand, D. Breen, A. Heirich, Scalable interactive volume rendering using off-the-shelf components, in: Proceedings of IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2001, pp. 115–121.

[21] MathSoft, Data Analysis Products Division. S-Plus 5 for UNIX User's Guide, 1998.

[22] S. Molnar, M. Cox, D. Ellsworth, H. Fuchs, A sorting classification of parallel rendering, IEEE Computer Graphics and Applications 14 (4) (1994) 23–32.

[23] S. Molnar, J. Eyles, J. Poulton, Pixelflow: High-speed rendering using image composition, Computer Graphics (Proceedings of SIGGRAPH 92) 26 (2) (1992) 231–240.

[24] J.S. Montrym, D.R. Baum, D.L. Dignam, C.J. Migdal, InfiniteReality: a real-time graphics system, in: Proceedings of SIGGRAPH 97, 1997, pp. 293–302.
[25] C. Mueller, The sort-first rendering architecture for high-performance graphics, in: 1995 Symposium on Interactive 3D Graphics, 1995, pp. 75–84.
[26] C. Mueller, Hierarchical graphics databases in sort-first, in: Proceedings of the IEEE Symposium on Parallel Rendering, 1997, pp. 49–58.
[27] R. Samanta, T. Funkhouser, K. Li, Parallel rendering with k-way replication, in: Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2001, pp. 75–84.
[28] R. Samanta, T. Funkhouser, K. Li, J.P. Singh, Hybrid sort-first and sort-last parallel rendering with a cluster of PCs, in: 2000 SIGGRAPH/Eurographics Workshop on Graphics Hardware, 2000, pp. 97–108.
[29] R. Samanta, T. Funkhouser, K. Li, J.P. Singh, Sort-first parallel rendering with a cluster of PCs, in: Sketches and Applications, SIGGRAPH 2000, 2000, pp. 260.
[30] R. Samanta, J. Zheng, T. Funkhouser, K. Li, J.P. Singh, Load balancing for multi-projector rendering systems, in: 1999 SIGGRAPH/Eurographics Workshop on Graphics Hardware, 1999, pp. 107–116.
[31] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, 1990.
[32] S.J. Teller, C.H. Séquin, Visibility preprocessing for interactive walkthroughs, Computer Graphics (Proceedings of SIGGRAPH 91) 25 (4) (1991) 61–69.
[33] The Walkthru Project at UNC Chapel Hill. Power plant model. http://www.cs.unc.edu/~geom/Powerplant/.
[34] I. Wald, P. Slusallek, C. Benthin, Interactive distributed ray tracing of highly complex models, Rendering Techniques 2001, 2001, vol. 23, pp. 277–288.
[35] B. Wei, D.W. Clark, E.W. Felten, K. Li, G. Stoll, Performance issues of a distributed frame buffer on a multicomputer. 1998 SIGGRAPH/Eurographics Workshop on Graphics Hardware, 1998, pp. 87–96.
[36] X. Zhang, C. Bajaj, W. Blanke, D. Fussell, Scalable isosurface visualization of massive datasets on COTS clusters, in: Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2001, pp. 51–58.