# Compression-Based Ray Casting of Very Large Volume Data in Distributed Environments

*Chandrajit Bajaj*
Department of Computer Sciences
The University of Texas at Austin, U.S.A.
`bajaj@cs.utexas.edu`

*Insung Ihm* *Sanghun Park* *Dongsub Song*
Department of Computer Science
Sogang University, Korea
{`ihm,hun,kubrick`}`@grmanet.sogang.ac.kr`

## Abstract

*This paper proposes a new parallel/distributed ray-casting scheme for very large volume data that can be effectively used in distributed environments. Our method, based on data compression, attempts to enhance the rendering speedups by quickly reconstructing voxel data from local memory rather than expensively fetching them from remote memory spaces. Our compression-based volume rendering scheme minimizes communications between processing elements during rendering computation, hence is very appropriate for both distributed-memory multiprocessors and PC/workstation clusters, where the relatively high communication costs often hinder efficient parallel/distributed processing. We report experimental results on both a Cray T3E and a PC/workstation cluster for the Visible Man dataset.*

**Keywords:** volume visualization, very large volume data, data compression, parallel/distributed ray-casting, distributed-memory multiprocessor, PC/workstation cluster.

## 1. Introduction

Volume visualization is the research area that attempts to extract meaningful visual information from abstract and complex datasets generated in a variety of scientific and engineering fields. In various fields such as computational fluid dynamics, earth, space and environmental science, and medical science, volume data are often so huge, ranging from several hundred megabytes to several dozen gigabytes, that they need special treatment for effective manipulation. With the significant advances in computation, measurement, and storage technologies, terascale datasets have become increasingly commonplace [17].

One example of very large volume data in medical imaging is the dataset disseminated by the National Library of Medicine (NLM) of the U.S.A. They created huge volume datasets made of computer tomography (CT), magnetic resonance imaging (MRI), and color cryosection images of male and female human cadavers in an effort to offer a complete digital atlas of the human body [15]. The "Visible Man" data consists of axial scans of the entire body taken 1 mm intervals at a resolution of $512 \times 512$, in which the whole dataset has over 1870 cross-sections. The "Visible Woman" data is made of cross-sectional images taken at one-third the interval of the male. The datasets amount to 15 Gbytes and 40 Gbytes, respectively. Many efforts have been made to visualize these datasets in various parallel and distributed environments. (For the previous works on use of parallelism for rendering of the Visible Human, such as MPIRE, refer to [15].).

Visualizing such very large volume data requires intensive computing time and memory space. In particular, the ray-casting algorithm is known to produce the highest quality of rendered images, although it is one of the most compute- and memory-intensive tasks for volume visualization. There have been several previous works to parallelize the ray-casing algorithm on various supercomputing environments [1, 7, 8, 12]. In attempt to ray-cast large volume datasets at interactive rates, we have developed an effective parallel/distributed ray-casting scheme that can be used to visualize very large volume data in the distributed environments. In particular, we are concerned with two platforms. First, we consider the Cray T3E, a distributed-memory multiprocessor which offers high-performance computing power. Secondly, we consider clusters of networked PCs and workstations that are highly available in most environments. While PC/workstation clusters are considered as a more practical solution in solving large scale scientific problems, latency is pretty high and bandwidth is relatively low compared to parallel computers. This has made it difficult to achieve high performance for most applications. Our rendering method tries to achieve high performance by minimizing, through compression, communications between processing elements during rendering, hence is very appropriate for clusters as well as distributed-memory multiprocessors.

Our parallel ray-casting scheme is different from the previous approaches in that it is based on a compression method that is well-suited for developing interactive applications. In [3], a new compression method, called zerobit encoding, for 3D gray-scale and RGB volume data was developed for real-time or interactive applications. In particular, it, based on 3D Haar wavelets, provides very fast random access ability to compressed volume data. Most previous parallel rendering algorithms for very large volumes often partition the data into subblocks that can fit into local memory of processing elements, and distribute them over the local memory spaces in the system. During rendering, load balancing is usually done dynamically for efficient computation, and this often causes data redistribution between processing elements. The data redistribution, or remote memory fetch, when implemented carelessly in distributed environments, is one of the most serious factors that deteriorate the speedup of parallel volume rendering, especially when the data is very large [13]. In the rendering scheme, presented in this paper, on the other hand, the entire volume is stored in compressed form at local memory, and is locally reconstructed on the fly as necessary. Since no data communication is required between processors for data redistribution, our scheme produces better speedups in distributed environments than the previous approaches based on data redistribution.

## 2. Wavelet-Based 3D Compression of Volume Data for Interactive Applications

Our parallel/distributed ray-casting algorithm utilizes a 3D volume data compression scheme which was designed to meet the following desirable properties: high compression ratio, minimal distortion in the reconstructed images, fast random access ability, multi-resolution representations, effective exploitation of data redundancy, selective blockwise compression, and so on. Most of all, very large volume data must be compressed into smaller sizes that can fit into local memory spaces, retaining the contents as best as possible after reconstruction. When an individual voxel in the compressed data is accessed in an unpredictable fashion as in many applications, it must be reconstructed quickly during run-time. This ability of fast decompression to random access becomes one of the most important factors when interactive or real-time applications are developed.

Vector quantization [5], that meets some of the above properties, has been used frequently in developing interactive real-time applications because it offers fast random decoding through simple table lookups. Recent applications of vector quantization in the computer graphics field, include compression of CT/MRI datasets [14], light fields [10], and 2D textures [4].

In an effort to provide a better compression method suitable for interactive/real-time applications, we have developed a 3D wavelet-based compression method [6], and improved its performances substantially in such a way that the

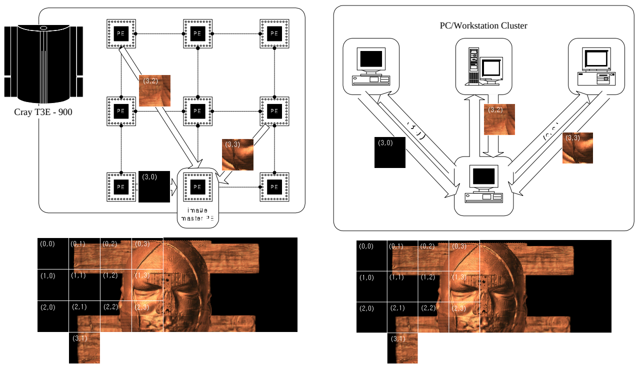
| | | UC | Target Ratios | | | |
|---|---|---|---|---|---|---|
| | | | 3% | 5% | 7% | 10% |
| Pure Random | | 2.78 | 3.33 | 3.49 | 3.62 | 3.85 |
| CW | All | 18.88 | 3.88 | 4.88 | 5.99 | 7.52 |
| | Skin | 6.50 | 2.28 | 2.91 | 3.53 | 4.36 |

**Figure 1. Voxel Reconstruction Time (UC: Uncompressed, CW: Cell-Wise)**

new encoding scheme, called zerobit encoding, can be applied to 3D volume data having either gray-scale or RGB-color voxel values [3]. Figure 1 briefly show the timing performances of the compression method, tested on an SGI machine with 195MHz R10000 CPU using the fresh CT dataset of the Visible Man that amounts 720Mbytes ($512 \times 512 \times 1440 \times 2$bytes). We tested with the four target ratios of wavelet coefficients where the target ratio is roughly the rates of non-zero wavelet coefficients used after truncation. Compression rates 11.90 to 29.27 were achieved for the target ratios. When more than 7% of wavelet coefficients are used, the ray-cast images are pretty good compared with those generated from the uncompressed dataset. Two situations were considered to evaluate reconstruction overheads: First, the timings (in seconds) for Pure Random access were taken by repeatedly fetching voxel values one million times with randomly generated indices $(i, j, k)$ from uncompressed and compressed data, respectively. The test results indicate that fetching voxel values from compressed data is about 1.20 to 1.38 times slower. The timing differences can be ignored in many compute-intensive applications such as volume rendering, which usually take more than a hundred seconds. Secondly, the timings for Cell-Wise access were taken when voxels are grouped into $4 \times 4 \times 4$ subblocks, called *cells*, and are reconstructed cell-by-cell. Cell-wise reconstruction, which is used in our ray-caster, is more efficient for the applications, such as volume rendering, where data are accessed with some regular pattern. The results show the timings taken for accessing, cell-wise, all cells in the dataset (All), and only cells classified as skin (Skin). Notice that the access speed is even faster when the voxels are accessed from compressed data. This is because most of the null detail coefficients are not even traversed in reconstruction.

## 3. Compression-Based Parallel Ray-Casting in Distributed Environments

Our compression-based volume ray-casting scheme is straightforward. First, the image screen is divided into a set of regular spaced pixel tiles of small sizes that forms a pool of tasks. The virtual rendering system is made of one master processing element (MPE) and multiple slave processing elements (SPE) where a SPE can be run on the same physical processor as the MPE. The MPE manages the entire rendering process by dynamically assigning tiles in the task list, and collecting the rendered image segments. The SPEs perform ray-casting repeatedly on assigned tiles



(a) On Cray T3E

(b) On PC/Workstation Cluster

**Figure 2. Mapping of the Compression-Based Rendering Algorithm**

until the task pool becomes empty. Load balancing is carried out dynamically during rendering. Figure 2 (a) and (b) show how our rendering method is ported on the two different platforms. The simplicity of the rendering scheme makes it possible to implement it on the two environments with little modification.

The major difference of our method from the previous parallel ray-casting techniques, devised for distributed environments, is that the entire volume dataset, compressed with the zerobit encoding technique, is loaded at each local memory in a compressed form. Voxels necessary for ray-casting are quickly reconstructed on the fly rather than non-local voxels are fetched on demand from remote processors. Hence, the data communicated by processors are only the control information and rendered image segments whose total sizes are much smaller than that of 3D volume data. In spite of fast communication networks of the recent distributed-memory multiprocessors like Cray T3E, preventing volume data from being redistributed greatly helps achieve high speedups. It is very critical for more practical distributed systems like PC/workstation clusters, where communications between remote processors are regarded as quite costly.

For rendering on each SPE, we use a true volume ray-caster, optimized by min-max octree and early ray termination [9]. In zerobit-encoding, a cell of $4 \times 4 \times 4$ voxels is the decoding unit while a $16 \times 16 \times 16$ unit block is the encoding unit. For each unit block, we build an min-max octree of level three where the leaves correspond to $4 \times 4 \times 4$ cells. In [2], they used a ray-caster based on object-order traversal, and adopted an image-space quadtree to simulate the early ray termination technique. A similar blockwise ray-casting scheme that attempts to minimize the communication overheads by prefetching blocks necessary for rendering a tile, was proposed [8]. Contrary to the ray-by-ray traversal, the block-by-block traversal guarantees that each voxel needs to be reconstructed as in [2] or remotely fetched as in [8] at most once. Furthermore, it is more amenable to exploiting data coherence in object space than the ray-by-ray computation. However, implementing the ray-casting algorithm, developed inherently in image space, in the object-order fashion entails the additional chores for managing ray segments. With the improved decoding speed of zerobit encoding, and the simple ray-casting structure, we found that the renderer used in this paper is three to four times faster than that used in [2].

|  | Data Reconst. | Data Redist. on $n$ Proc's |  |  |  |  |
|---|---|---|---|---|---|---|
|  |  | 4 | 8 | 16 | 32 | 64 |
| All | 4.50 | 10.89 | 13.03 | 14.45 | 15.41 | 16.68 |
| Skin | 2.61 | 3.90 | 4.73 | 5.17 | 5.52 | 5.83 |

**Figure 3. Data Reconstruction v.s. Redistribution on Cray T3E**

## 4. Experimental Results

### 4.1. Issues on Data Redistribution v.s. Reconstruction

Our volume ray-casting method relies on the assumption that the on-the-fly reconstruction of voxels from zerobit-encoded volume is faster than the on-demand remote fetches on the high-performance multiprocessors like Cray T3E as well as on PC/workstation clusters. We carried out a simple experiment on the Cray T3E that compares timings between reconstruction and remote fetches. We used a Cray T3E-900 with 136 processors. The Cray T3E processing element (PE) includes a 450 MHz Alpha processor and 128 Mbytes local memory, and is connected by a high-bandwidth, low-latency bidirectional 3-D torus system interconnect network. In implementing our method, we used the Cray Shared Memory Access Library (SHMEM) which provides faster interprocessor communication than MPI and PVM do.

For a performance test, we have generated a $512 \times 512 \times 1440$ volume dataset from the original fresh CT data of the Visible Man, which takes up 720 Mbytes (Note that some portion of slices in the Legs section of the fresh CT are missing.). For rendering, it was compressed into a dataset of size 45.43 Mbytes, using the 3D wavelet compression scheme. This example data uses 7% of wavelet coefficients, and the rendered image quality is visually identical to that of the uncompressed data.

In this experiment, the entire Visible Man dataset in uncompressed form is partitioned into $n$ subvolumes of equal size where $n$ is the number of available processors. They are distributed into the processing elements, and are loaded into their local memory space. Each processor has a list of randomly generated voxel indices, and tries to access the corresponding voxel values. When a voxel is local, the processor reads it from its own local memory space. Otherwise, it remotely fetches a $4 \times 4 \times 4$ cell, containing the voxel, from remote memory space. Since there exists some degree of coherence in accessing voxels during ray traversal, we

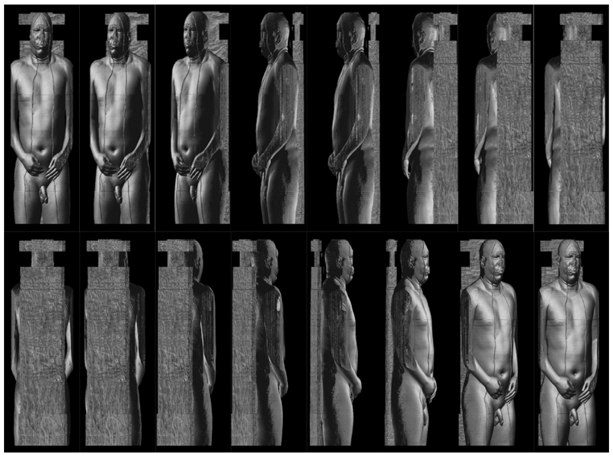


**Figure 4. A Sequence of Ray-Cast Images**

fetch a cell rather than an individual voxel so that the cell is put into a local cache for possible later use. Another simulation was performed to measure reconstruction overheads by taking timings for decompressing the $4 \times 4 \times 4$ cells that include voxels in the same access list.

Figure 3 compares the timing results. When voxels are reconstructed, there is no communication between processors. Hence we measured the timings on one processor. The row "All" shows the timings when the entire dataset is accessed. On the other hand, the row "Skin" is for the case where only the voxels, classified as skin, are decompressed or fetched. The table indicates that the communication overheads increase as more processors are added. In spite of the fast communication network of the Cray T3E, the data redistribution is one of the most negative factors that deteriorate parallel/distributed rendering performance. The preliminary test results on a PC/workstation cluster indicate that the redistribution cost is extremely high. This observation strongly implies that compression-based distributed rendering is one of the best solutions in the distributed environments especially when the data sizes are very large.

## 4.2. Performances on Cray T3E

We first implemented the compression-based volume ray-casting scheme on the Cray T3E. Timings were taken in seconds for both $512 \times 512$ and $512 \times 1024$ images as rotating the Visible Man by 20 degrees and averaging the rendering times (Figure 4). We have tested five different tile sizes, $2 \times 2$, $4 \times 4$, $8 \times 7$, $16 \times 16$, and $32 \times 32$, and was able to use up to 96 processors.

Figure 6 (a) to (e), and Figure 5 show the performance results that compare very favorably with existing results for direct volume rendering, say, [1, 7, 8, 11]. These timings do not include data replication and image display. We observe that efficiency higher than 89% and 97% are achieved on 96 processors for $512 \times 512$ and $512 \times 1024$ images, respectively, which surpasses most of the recently reported parallel volume rendering on the distributed-memory architectures. When $8 \times 8$ tiles were used for $512 \times 1024$ images, it took 64.1 seconds per frame on one processor, and 0.68 seconds on 96 processors. Considering that most portion of the $512 \times 1024$ image is opaque, and our ray-caster is a true volume renderer, the rendering speed appears to be very high. The primary reason for getting such good speedups is that our rendering scheme minimizes the data communication overheads during rendering. Breakdown of execution time for processors shows that the communication time for task assignment and image segment collection is negligible compared to the rendering time.

The graph (a) and (b) shows that the renderer performed best when the tile sizes are $4 \times 4$ and $8 \times 8$ for the image sizes $512 \times 512$ and $512 \times 1024$, respectively. There is a tradeoff between tile sizes and performances. The graph (c) illustrates how evenly the tasks are distributed for the various tile sizes when 16 processors are used. Here, the value for each processor is obtained by dividing the processor's time spent computing by the total execution time, and indicates how well the processor is utilized without communicating and idling. For a fixed image size, a fine-grained decomposition of the image screen achieves good dynamic load balancing and speedups as more processors join in computation. However, the larger number of smaller tasks increases overheads of communication and task management. At some point, there is no net gain from having the tiles smaller. In the graph (d) of the rendering times for $512 \times 1024$ images, we see the the tile size of $8 \times 8$ is optimal when more than 80 processors are used.

The graph (e) illustrates the rendering times over varying rotational angles, and indicates that the oblique views take more rendering time. When the Visible Man is rendered from the front view, it took only 0.62 seconds to generate the $512 \times 1024$ image on 96 processors. Our optimized ray-caster module produces much faster rendering compared to the previous result [2] in which 2.86 seconds was taken for the front view on the same number of processors.

## 4.3. Performances on PC/Workstation Cluster

We also implemented our rendering scheme on a PC/workstation cluster. Figure 6 (f) to (g) show the distributed rendering performances where the cluster consists of 8 400/450 MHz Intel Pentium II processors and two 195 MHz MIPS R10000 processors running Windows NT 4.0/Windows 98 and Irix 6.4, respectively. We generated both $512 \times 512$ and $512 \times 1024$ images using $16 \times 16$, $32 \times 32$, and $64 \times 64$ tiles. The computers in the cluster are connected through an Ethernet, and there was normal network traffic during test period. The Pentium II processors were added one at a time until all 8 PCs joined, then the two SGIs are added. Since the R10000 CPU turned out to be a little bit slower in running our ray-caster than the Pentium II processors, the speedup went a little worse when the 9th and 10th processors are added (graph (f) and (g)). Since the cluster is made of heterogeneous processors, the speedup curve is less meaningful to study the performance of our method. Nevertheless, we observe that the compression-based volume rendering scheme is also pretty effective on the PC/workstation clusters.

| | | 1 | 32 | 48 | 64 | 80 | 96 |
|---|---|---|---|---|---|---|---|
| $512 \times 512$ Images ($4 \times 4$ Tiles) | Ren. Times (sec's) | 34.53 | 1.16 | 0.78 | 0.59 | 0.48 | 0.40 |
| | Speedups | 1.0 | 29.7 | 44.4 | 58.9 | 72.0 | 86.1 |
| $512 \times 1024$ Images ($8 \times 8$ Tiles) | Ren. Times (sec's) | 64.09 | 2.09 | 1.40 | 1.05 | 0.81 | 0.68 |
| | Speedups | 1.0 | 30.7 | 45.9 | 61.0 | 78.6 | 93.8 |

**Figure 5. Rendering Times and Speedups with 96 CPUs**

When all the 10 PCs/SGIs were used to generate a $512 \times 1024$ image, it took 6.38 seconds on the cluster using $32 \times 32$ tiles. When only the 8 PCs were used, it took 7.68 seconds. This is, in fact, faster than the parallel rendering on Cray T3E using 8 processors that took 8.23 seconds. The current high-end processors of PCs and workstations are as fast as those of the expensive high-performance parallel machines. As mentioned, the major drawback of a PC/workstation cluster is that its communication network is relatively slower. Contrary to the Cray T3E which has a fast communication network, we observe that the time spent communicating and idling increases in the PC/workstation cluster environment, which deteriorates the overall performance (graph (h)). In our compression-based scheme, the time taken for communication is very small compared to the time taken for rendering computation. This property becomes crucial when our method is implemented on PC/workstation clusters with a slower Ethernet links. We were able to use up to only 10 processors due to the limitation of our environment. While achieving linear speedup in such a heterogeneous environment has been considered to be difficult, the simple computational structure of our scheme will allow to achieve very good speedups in distributed volume rendering when a large number of computers join the rendering computation.

## 5. Conclusions and Future Work

In this paper, we proposed a new compression-based parallel/distributed ray-casting scheme devised for the distributed environments, and showed that it can be used effectively for rendering very large volumes. Our current result may not be the fastest in terms of frame rates. For example, interactive rendering of iso-surfaces of Visible Woman was achieved on an SGI Reality Monster, which is a scalable shared-memory multiprocessor [16]. Our ray-casting scheme targets visualization of very large volume data on distributed systems, and attempts to achieve high performance by minimizing communications between processing elements during rendering through compression. Our scheme is more practical than the previous works in that it is also very appropriate for distributed systems with low bandwidth links such as easily available clusters of PCs and workstations, in which communications between processors are regarded as quite costly. We have been applying our ray-casting scheme to development of a volume navigation system which harnesses the computing power of clusters to interactively investigate very large volumes. We believe that utilizing idle CPU time of networked PCs and workstations in this way would be the most cost-effective way to visualize huge volume datasets.

**Acknowledgements** This paper was accomplished with the research fund provided by Korea Research Foundation under Support for Faculty Research Abroad.

## References

[1] M. Amin, A. Grama, and V. Singh. Fast volume rendering using an efficient, scalable parallel formulation of the shear-warp algorithm. In *Proceedings of the 1995 Parallel Rendering Symposium*, pages 7–14, Atlanta, October 1995.

[2] C. Bajaj, I. Ihm, G. Koo, and S. Park. Parallel ray casting of visible human on distributed memory architectures. In *Proceedings of VisSym '99 (Joint EUROGRAPHICS-IEEE TCCG Symposium on Visualization)*, pages 269–276, Vienna, Austria, May 1999.

[3] C. Bajaj, I. Ihm, and S. Park. 3D RGB image compression for interactive applications. Technical report, TICAM, University of Texas at Austin, U.S.A., November 1999.

[4] A. Beers, M. Agrawala, and N. Chaddha. Rendering from compressed texture. *Computer Graphics (Proc. SIGGRAPH '96)*, pages 373–378, 1996.

[5] A. Gersho and R. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[6] I. Ihm and S. Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18(1):3–15, 1999.

[7] P. Lacroute. Real-time volume rendering on shared memory multiprocessors using the shear warp factorization. In *Proceedings of the 1995 Parallel Rendering Symposium*, pages 15–22, Atlanta, October 1995.

[8] A. Law and R. Yagel. Multi-frame thrashless ray casting with advancing ray-front. In *Proceedings of Graphics Interface '96*, pages 70–77, Tronto, Canada, May 1996.

[9] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[10] M. Levoy and P. Hanrahan. Light field rendering. *Computer Graphics (Proc. SIGGRAPH '96)*, pages 31–42, 1996.

[11] P. Li, S. Whitman, R. Mendoza, and J. Tsiao. ParVox – a parallel splatting volume rendering system for distributed visualization. In *Proceedings of the 1997 Symposium on Parallel Rendering*, pages 7–14, Phoenix, U.S.A., October 1997.

[12] K.-L. Ma, J. Painter, C. Hansen, and M. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14(4):59–68, July 1994.

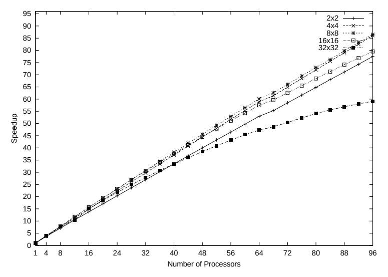
[13] U. Neumann. Communication costs for parallel volume-rendering algorithms. *IEEE Computer Graphics and Applications*, 14(4):49–58, July 1994.

[14] P. Ning and L. Hesselink. Fast volume rendering of compressed data. In *Proceedings of Visualization '93*, pages 11–18, San Jose, October 1993.

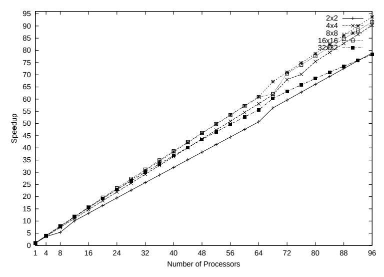
[15] NLM. $http://www.nlm.nih.gov/research/visible/visible\_human.html$, 1998.

[16] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, 1999.

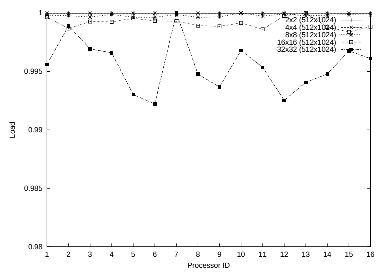
[17] T. M. Rhyne, editor. *Visualizing and examining large scientific data sets: a focus on the physical and natural sciences*. ACM SIGGRAPH, 1994. ACM SIGGRAPH '94 Course Notes.



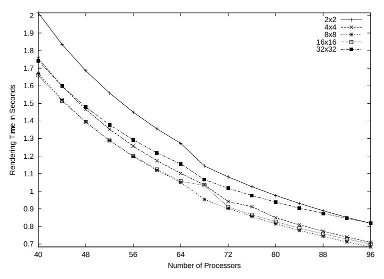
(a) Speedups I: $512 \times 512$



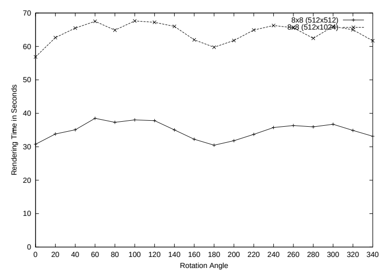
(b) Speedups II: $512 \times 1024$



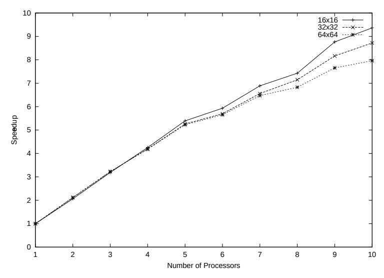
(c) Load Balancing on 16 Processors



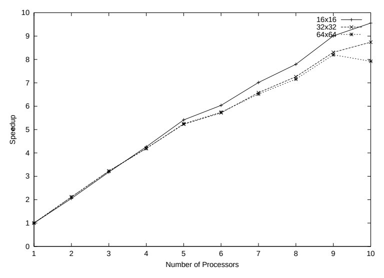
(d) Rendering Times: $512 \times 1024$



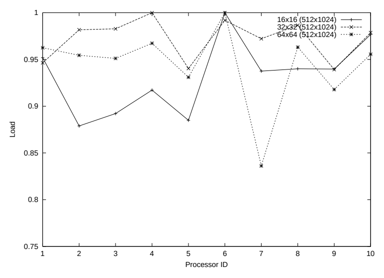
(e) Rendering Times for 18 Rotational Angles



(f) Speedup III: $512 \times 512$



(g) Speedup IV: $512 \times 1024$



(h) Load Balancing on 10 PCs/WSs

**Figure 6. Performances on PC/Workstation Cluster**