# Parallel Ray Casting of Visible Human on Distributed Memory Architectures

Chandrajit Bajaj[1], Insung Ihm[2], Gee-bum Koo[2], and Sanghun Park[2]

[1] Dept. of Computer Sci., Univ. of Texas at Austin, U.S.A.
[2] Dept. of Computer Sci., Sogang Univ., Seoul, Korea

**Abstract.** This paper proposes a new parallel ray-casting scheme for very large volume data on distributed-memory architectures. Our method, based on data compression, attempts to enhance the speedup of parallel rendering by quickly reconstructing data from local memory rather than expensively fetching them from remote memory spaces. Furthermore, it takes the advantages of both object-order and image-order traversal algorithms: It exploits object-space and image-space coherence, respectively, by traversing a min-max octree block-wise and using a run-time quadtree which is maintained dynamically against pixels' opacity values. Our compression-based parallel volume rendering scheme minimizes communications between processing elements during rendering, hence is also very appropriate for more practical distributed systems, such as clusters of PCs and/or workstations, in which data communications between processors are regarded as quite costly. We report experimental results on a Cray T3E for the Visible Man dataset.

## 1 Introduction

A few years ago, the National Library of Medicine (NLM) of the U.S.A. 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 [12]. 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 data set 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 data sets amount to 15 Gbytes and 40 Gbytes, respectively.

Visualizing such very large volume data requires a great deal of computing time and memory space. In particular, ray-casting such volume data is one of the most compute- and memory-intensive tasks for volume rendering, while the ray-casting algorithm produces the highest quality of rendered images. The motivation for this work is to develop an effective parallel ray-casting scheme for visualization of very large volume data on distributed systems. In this article, we are particularly concerned with parallel ray-casting of the Visible Human datasets, on a Cray T3E, a distributed-memory parallel computer (For the previous works on parallel rendering of the Visible Human, such as MPIRE, refer to

[12].). Our new method tries to achieve high performance by minimizing communications between processing elements during rendering through compression, hence is also very appropriate for more practical distributed systems, such as clusters of PCs and/or workstations, in which data communications between processors are regarded as quite costly.

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, 4, 13], Ihm et al. developed a new compression method, based on 3D wavelets, that provides very fast random access ability to compressed volume data. Most parallel rendering algorithms for very large volumes 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 efficiency, and this often causes data redistribution between processing elements. The data redistribution, or remote memory fetch, when implemented carelessly, is one of the most serious factors that deteriorate the speedup of parallel volume rendering, especially when the data is very large [11].

In our implementation, the whole CT dataset of the Visible Man is compressed, and is replicated at each processing element. Since the entire dataset that is necessary for generating image segments, is available at the local memory, no data communication is needed between processors for data redistribution. As briefly explained in Section 2, the compression method we use, guarantees very quick random access which is faster than remote data fetch, hence produces a better speedup than the previous methods based on data redistribution.

## 2 Wavelet-Based 3D Compression of Volume Data

Our parallel ray-casting algorithm requires a volume data compression scheme which has the following properties: High compression ratio, minimal distortion in the reconstructed images, and fast random access ability. First, very large volume dataset (say, several hundred mega bytes to a few giga bytes) should be compressed into smaller sizes (say, 64 to 128 mega bytes) that can fit into local memory spaces. Second, the contents of the images should be retained as best as possible after reconstruction. Lastly, when an individual voxel in the compressed data is accessed in a random fashion, the data item should be reconstructed quickly during run-time.

Ihm et al. [3, 4] have compromised between these factors, and developed a wavelet-based 3D compression scheme for interactive visualization of very large volume data, and the timing performance has been enhanced further in [13]. Table 1 and 2 brief the performances of the compression method described in [13]. The experiments are performed 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 ratios of wavelet coefficients that are used after wavelet transforms. Table 1 shows the compression ratios and quality of the reconstructed images. When more than 7% of wavelet coefficients

are used, the ray-cast images are virtually identical with those generated from the uncompressed dataset.

Table 2 indicates how fast the compression scheme reconstructs an individual voxel from compressed data. 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 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 faster when the voxels are accessed from compressed data. This is because most of the null detail coefficients are not even traversed in reconstruction.

**Table 1.** Experimental Results on Compression Quality

| | | Desired Ratio of the Wavelet Coef's Used | | | |
|---|---|---|---|---|---|
| | | 3% | 5% | 7% | 10% |
| Compression Performance | Compressed Data Size (MB) | 24.59 | 35.06 | 45.43 | 60.50 |
| | Compression Ratio | 29.27 | 20.54 | 15.58 | 11.90 |
| Errors in Voxel Values | SNR (dB) | 22.64 | 25.94 | 28.57 | 31.99 |
| | PSNR (dB) | 44.49 | 47.79 | 50.41 | 53.84 |

**Table 2.** Experimental Results on Voxel Reconstruction Time

| | Uncompressed | Desired Ratio of the Wavelet Coef's Used | | | |
|---|---|---|---|---|---|
| | | 3% | 5% | 7% | 10% |
| Pure Random | 2.78 | 3.33 | 3.49 | 3.62 | 3.85 |
| Cell-Wise All | 18.88 | 3.88 | 4.88 | 5.99 | 7.52 |
| Cell-Wise Skin | 6.50 | 2.28 | 2.91 | 3.53 | 4.36 |

## 3 Compression-Based Parallel Ray-Casting

### 3.1 Image-Order and Object-Order Volume Rendering

Volume ray-casting is an *image-order* volume rendering algorithm that is most popularly used since it produces the best rendering quality [8]. Various opti-

mization methods have been proposed for ray-casting. The early ray termination technique allows to stop sampling along the rays as soon as accumulated opacities reach a pre-specified threshold value [9]. Hierarchical data structures such as octrees and pyramids, and k-D trees, have been applied to exploit object-space coherence inherent in volume data [9, 2, 15]. The data access pattern during parallel ray-casting is very irregular, and that makes such data structures less natural than the *object-order* algorithms.

In the object-order algorithms such as splatting, on the other hand, the data are traversed in a regular manner, hence, the data coherence can be exploited very easily [16]. While the object-order algorithm is more amenable to parallelization, it has the disadvantage that it is difficult to apply the optimization techniques for image-order algorithms, such as early ray termination. As described below, our *object-order* ray-casting exploits the advantages from both algorithms.

### 3.2 Our Parallel Ray-Casting Scheme

In our parallel rendering scheme, image screen is divided into regular spaced pixel tiles of small sizes, and these form a pool of tasks. During run-time, processors are assigned tiles from the pool of tasks waiting to be executed. The processors perform ray-casting repeatedly on tiles until the task pool becomes empty. Load balancing is carried out dynamically during rendering.

As mentioned previously, the entire CT data of the Visible Man are replicated at each local memory space in a compressed form. For compression, the volume data are partitioned into $16 \times 16 \times 16$ subblocks, called *unit block*, and unit blocks become the basic unit for encoding and decoding. Each compressed unit block is associated with the min-max values of voxels within it, which prevents from reconstructing voxels of no interest. The processor computes image segments corresponding to tiles using ray-casting in the object-order fashion. When a tile is assigned, its view volume is computed, and the unit blocks that intersect with the view volume are listed, on demand, in the front-to-back order by traversing an octree with a proper height, constructed from min-max unit blocks. Each unit block is, then, projected into the tile on the image plane, and the rays for the pixels within the projected area are advanced simultaneously through the unit block by accumulating colors and opacities.

The block-by-block access to the volume during ray-casting is more amenable to exploiting the data coherence in object space than the ray-by-ray access. In most previous methods, non-local voxels are fetched from remote memory on demand, and are usually cashed locally. When the cache is not large enough, the system starts to thrash. Since it results in a poor speedup, it is important to utilize voxels maximally once fetched. In [6], Law et al. also ray-cast volumes block-wise, and showed the object-order traversal reduces the costs for data redistribution. In our method, the entire volume data is available at local memory, but only at the cost of reconstruction. Although our compression method reconstructs voxels very fast, it is also important to minimize the number of reconstruction operations for the same voxels. Contrary to ray-by-ray traversal,

block-wise traversal guarantees each voxel in the view volume is reconstructed at most once.

A problem with ray-casting based on the object-order traversal, is that it is no longer natural to apply the early ray termination technique which allows significant savings in computation. In our implementation, we solved this problem by using an image-space quadtree that exploits the image-space coherence as in [7]. To efficiently determine the visibility of regions in a tile, we construct a quadtree in image space as follows. Each leaf node of the quadtree, corresponding to each pixel in the tile, has value 1 when the pixel has been made opaque enough, that is, the opacity has reached a threshold value, and 0 otherwise. The four adjacent values at each level are then combined into one value at the next, coarser level by adding them. When a non-terminal node in the tree has value 4, it indicates that the corresponding region in the image plane is opaque, and one is recursively added to its parent node. Otherwise, its region is not opaque, and more ray sampling operations are necessary.

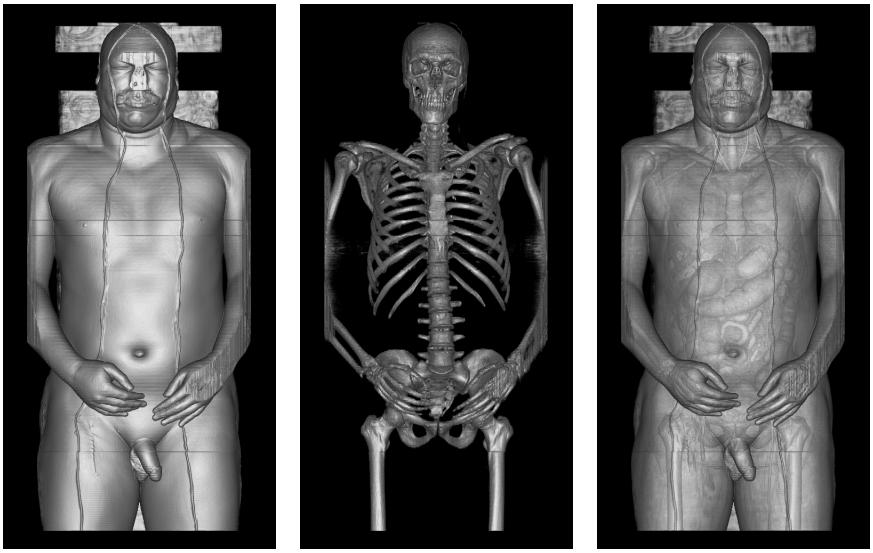
When a unit block is projected into the tile, the opaque regions are quickly rejected by recursively traversing the quadtree against the projected area, and the resampling operations are performed only on the transparent rays. When the entire tile gets opaque, that is the quadtree's root have value 4, traversing the list of unit blocks stops. In this way, we can simulate early ray termination in the object-order traversal algorithm.

### 3.3 Experimental Results

We implemented our parallel scheme on 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.

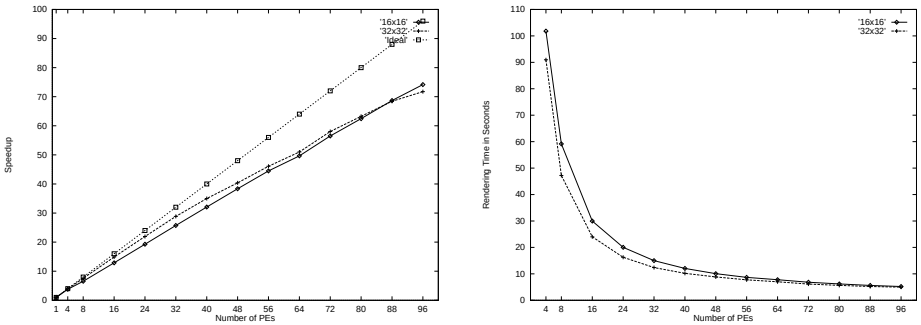
For a performance test, we have generated a $512 \times 512 \times 1440$ volume data set from the original fresh CT data of the Visible Man, which takes up 720 Mbytes (Note that some portion of slices in the Legs section of the fresh CT are missing.). For rendering, it was compressed into a dataset of 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 from the uncompressed data.

Timings were taken in seconds for generating $512 \times 1024$ images (Skin) using $16 \times 16$ and $32 \times 32$ tiles (Figure 1). Figure 2 shows the performance results that compare very favorably with existing results for direct volume rendering, say, [1, 5, 6, 10]. These timings do not include data replication and image display. When $32 \times 32$ tiles were used, it took 357 seconds per frame on one processor, and 4.9 seconds on 96 processors. We observe that higher than 80% efficiency is achieved for up to 80 processors, which surpasses most of the recently reported parallel implementations for direct volume rendering.



(a) Skin (b) Bone (c) Trans. Skin

**Fig. 1.** Parallel Ray-Cast Visible Man




(a) Speedup (b) Rendering Time

**Fig. 2.** Speedup and Rendering Time

The primary reason for getting the good speedup is that our compression-based parallel ray-casting scheme minimizes the data communication overheads during rendering. Only communication for task assignment and image segment collection is necessary, and breakdown of execution time for processors shows that the time taken for communication is very small compared to the time taken for rendering computation. For instance, when 64 processors are used, the average ratio of communication time to rendering time is less than 0.0001, which is negligible. This property becomes crucial when our method is implemented on such other platforms as PC/workstation clusters with slower Ethernet links, in which data communication is usually very expensive.

Notice that the performance depends on the tile sizes: As more processors join in computation, smaller tiles achieve better dynamic load balancing. However, the larger number of tasks increases overheads of managing tasks. At some point, there is no net gain from having the tiles smaller. The extra overheads such as task assignment and partial image collection, are very small in our scheme, and we observe that smaller task sizes, such as, $16 \times 16$ or $32 \times 32$ tiles, produce a better performance. Figure 3 shows how evenly the tasks are distributed among processors when 16 processors are used. Currently, we are modifying the task assignment strategy so that the task sizes vary dynamically according to various computing parameters.

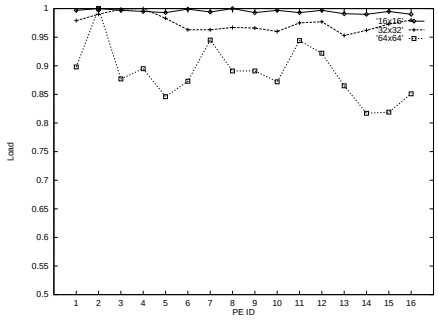



**Fig. 3.** Load Balancing

## 4 Conclusions and Future Work

In this paper, we proposed a new compression-based parallel ray-casting scheme aimed at the distributed-memory architecture, and showed that it can be used very effectively for 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 [14]. Our parallel scheme targets visualization of very large volume data on distributed systems, and tries 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, as well as distributed-memory multiprocessors, such as easily available clusters of PCs and workstations, in which communications between processors are regarded as quite costly.

We have been optimizing our parallel volume render to enhance its performance. Currently, a $16 \times 16 \times 16$ unit block is the basic unit for compression

and decompression. Sometimes, for example, when the normal and gradient at a voxel is to be approximated using central differences, unnecessary voxels may have to be decoded. We are testing with $4 \times 4 \times 4$ cells as the basic unit, and a preliminary experiment with the optimized renderer shows almost two times faster rendering. There are very few works to compare with our results on parallel direct volume rendering of Visible Human. We are trying to achieve one frame per second on 96 processors for the Skin classification (Figure 1 (a)). Considering that most portion of the $512 \times 1024$ image is opaque, and our ray-caster is a true volume renderer, the goal appears to be high enough.

Our scheme also needs to be modified for huge data sets that do not fit into local memory spaces even after compression. We believe that a rendering scheme based on data partition and compression, will also perform very well, and make it possible to handle huge volume data more effectively.

**Acknowledgements** We wish to thank the ETRI supercomputing center in Korea for access to the CRAY T3E-900.

## 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 Parellel Rendering Symposium*, pages 7–14, Atlanta, October 1995.

[2] D. Cohen and Z. Sheffer. Proximity clouds - an acceleration technique for 3D grid traversal. *The Visual Computer*, 11:27–38, 1994.

[3] I. Ihm and S. Park. Wavelet-based 3D compression scheme for very large volume data. In *Proceedings of Graphics Interface '98*, pages 107–116, Vancouver, Canada, June 1998.

[4] I. Ihm and S. Park. Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 1999. *To appear.*

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

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

[7] R. Lee and I. Ihm. On enhancing the speed of splatting using both object- and image-space coherence. Submitted for publication, 1998.

[8] M. Levoy. Display of surface from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

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

[10] 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.

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

[12] NLM. *http://www.nlm.nih.gov/research/visible/visible_human.html*, 1998.

[13] S. Park, G. Koo, and I. Ihm. Wavelet-based 3D compression schemes for the Visible Human dataset and thier applications. In *CD-ROM Proceedings of Visible Human Project Conference '98*, Maryland, USA, October 1998.

[14] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings of IEEE Visualization '98*. IEEE, 1998.

[15] K. Subramaanian and D. Fussell. Applying space subdivision techniques to volume rendering. In *Proceedings of Visualization '90*, pages 150–159, 1990.

[16] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, 1990.