Distributed Volume Rendering on a Visualization Cluster

Susan Frank and Arie Kaufman
Center For Visual Computing and Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400, USA

Abstract

We describe the rendering of massive volumes on a volume visualization cluster. We present our data distribution scheme and introduce an algorithm which reduces the memory requirement with no loss of accuracy. The volume is automatically cropped and partitioned into small volume blocks. The bounding boxes of these volume blocks are used at run-time for flexible partitioning of the volume across the network. We present results of rendering the full Visible Male color dataset, seismic data, and several large micro-CT scanned fossil and teeth datasets.

1. Introduction

Recently, volume resolutions have increased dramatically as data acquisition devices have become more sophisticated. At the same time, photographic volumetric datasets such as the Visible Male and Visible Female datasets from the National Library of Medicine [10, 16], have become more prevalent. The size of the data grows substantially due to multichannel information. Several cluster-based visualization architectures that support parallel rendering algorithms have been proposed or developed [4, 5, 12, 17]. The advantage of such systems is that they are easy to build from commodity components.

The main contribution of this work is the development of *Region of Interest Cropping*, an out-of-core algorithm for pre-processing massive volumetric data, and a scheme for distribution of this data on a volume rendering cluster. For the datasets studied, memory requirements are reduced by an average of 68% without any loss of data resolution.

Our block-level space leaping has been used to increase frame rates for direct volume rendering [2]. Samanta et al. [15] use a view dependent, hybrid 2D image and 3D polygon data partitioning scheme. Li et al. [7] achieve space skipping by partitioning volumes based on voxel attributes. Our block-level space leaping is achieved by *Volume Distribution Manager*, which uses a heuristic for the tradeoff between increasing empty space with larger subvolumes and overhead cost per subvolume. We are able to render very large volumes interactively without downsampling.

The rest of this paper is organized as follows. We present our Region of Interest Cropping algorithm and our memory management scheme in Section 2. In Section 3 we describe our test environment and some implementation issues. Results are given in Section 4.

2 Memory Management Scheme

Massive volumes suitable for distribution across a cluster do not fit within the paradigm of seeded region growing [1]. We introduce instead an incremental slab-to-slab version, where we read one small z-slab at a time. Slabs of approximately 20 slices work well. These are each segmented using seeded region growing. For each detected region a volume *block* is created, which is cropped to the region's axis-aligned bounding box. The block's position and size are retained for the Volume Distribution Manager.

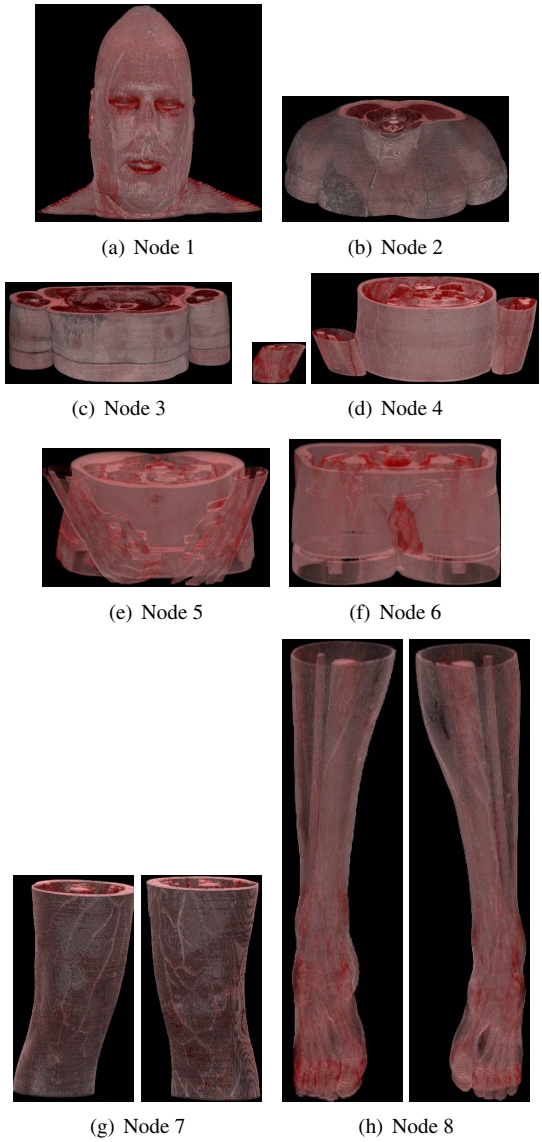
By default, a volume is distributed over the minimum number of nodes required to fit it. Alternatively, the user may request that a particular number of nodes be used. Each node may render one or more subvolumes. Each subvolume requires a rendering pass. In addition, one or more slices of voxels must be replicated between adjacent subvolumes for correct interpolation and shading during rendering.

We use a heuristic approach to balance this overhead with the increase in the per slice cost required to include slices with differing bounding rectangles in a subvolume. In our approach, the volume is broken into small, segmented non-overlapping volume blocks, which are recombined into larger subvolumes as described in Section 2.2.

2.1 Region of Interest Cropping Algorithm

The region grow algorithm is run on each distinct region of interest within a slab. A threshold range and the initial seed for each separate region within the whole volume are provided by the user. A mask volume is produced during region growing. The last slice of this mask volume, the *mask slice*, is used for seeds in region growing on the next slab. If this slice contains more than one distinct region, we say that the volume has a *split* within that slab, as in the arm

in Figure 1(d). We must insure that a separate region grow process is initiated for each distinct region of interest in the next slab which is connected to this mask's region of interest. If there is only one region, it means that the two regions *merge* in the next slab. In this case, we must be careful not to produce duplicate copies of the same region. Thus, before using a seed voxel from the mask slice on slab *s* we verify that it has not already been included in a segmented region for *s*.



(a) Node 1 (b) Node 2 (c) Node 3 (d) Node 4 (e) Node 5 (f) Node 6 (g) Node 7 (h) Node 8

Figure 1: Subvolumes for cluster rendering nodes.

As volume blocks are created they are grouped together. A *block group* is a consecutive group of volume blocks which have common faces, have contiguous nonempty voxels, and do not have a natural breaking point, such as a local minimum slice or empty voxel region. Each block group bounding box and volume block file list is maintained for the Volume Distribution Manager.

A block group is started when a block is encountered which does not belong with an existing block group, and it is *complete* when the volume defined by that block group encounters a local minimum x-y slice. If regions merge within a slab, then the block groups from each branch are completed and a new one is started. An active block group list is maintained which contains all block groups which are not complete. Slab *s* is processed as follows:

- Use seeded region grow segmentation on *s*.
- Create cropped volume block *b* for each region in *s*.
- If *b* belongs with any active group *g* add *b* to *g*.
- Else start a new group.
- Close active groups that have no new blocks.
- Use the final slice mask as seeds for next slab.

2.2 Volume Distribution Manager

The Volume Distribution Manager creates subvolumes from volumetric data and distributes these across a cluster in a manner that results in a good load balance. Additional subvolumes require additional rendering stages and replicated boundary slices. Fewer subvolumes result in more memory being wasted on empty voxels, especially when there is a wide range of bounding rectangle areas for the volume slices. We use a heuristic to balance this tradeoff. Memory calculations are all rounded to the nearest block.

The input to our algorithm is a list of volume block groups (as defined in Section 2.1), a list of node memory sizes, and a *useAllNodes* flag. Each slice in a volume which has not been preprocessed into block groups is treated as a single block which belongs to a single block group.

First, a feasibility check is made to assert that the smallest possible data size can fit into the specified nodes. This would be a single subvolume where the bounding boxes of all blocks line up in the *x* and *y* dimensions. Although this alignment is unlikely to occur for cropped subvolumes, it serves as a useful lower bound. The size is $\sum B_b$, for all volume blocks, where B_b is the size of block *b*.

Next, a two-phase distribution process is used: the volume is initially distributed coarsely by block groups, then these are merged as the final subvolumes are created. The lower bound of memory for blocks *b* assigned to node *i*, and not yet included in a subvolume of *i*, is $L_i = \sum B_b$. Block groups are distributed evenly between nodes, splitting up a group if adding its blocks to L_i exceeds $memory_i$, the capacity of node *i*. In the case that the volume is to be spread across more nodes than the minimum, then $memory_i$ is set to $1/n$ of the total required to hold all of the block groups.

During the second phase, blocks from node *i* groups are gathered into subvolumes until $memory_i$ is reached. The

memory used by all subvolumes *s* currently assigned to node *i* is $M_i = \sum S_s$, where S_s is the size of subvolume *s*. The minimum memory required by node *i* with the current assignment is $M_i + L_i$. When block *b* is merged into subvolume *s*, L_i decreases by B_b. The slice size of *s* may increase, so S_i increases by *at least* B_b. We call the new sum MERGE(b,s). If MERGE(b,s) increases $S_i + L_i$ by more than an ϵ tolerance, or exceeds $memory_i$, then *b* is instead used to start a new subvolume. The assignment of the Visible Male data across our cluster, with $\epsilon = 15\%$, is shown in Figure 1. Our algorithm proceeds as follows:

Distribute Groups

- For each group *g*, if *g* fits in node *i* assign *g* to *i*.
- Else break off the blocks from *g* that don't fit into into *i* and send them to node *i+1*, and proceed to subvolume creation phase for *i*.

Create Subvolumes

- Init subvolume *s* to block 0.
- For each subsequent block *b*
- If $MERGE(b, s) < (1 + \epsilon) * (S_i + L_i)$ and $MERGE(b, s) <= memory_i$, merge *b* with *s*.
- Else use *b* to start a new subvolume.

3 Implementation

We have implemented our algorithm on top of our Open Volume Library, OpenVL [6]. OpenVL's extensible plugin-based support for image and volume file reading and writing is well suited for a universal rendering system. For example, the Visible Male data contains non-interleaved RGB data. An incremental change to the OpenVL raw file I/O plugin allowed us to create an interleaved RGB volume from this data. Another small change enabled us to use a single channel for the region grow algorithm, rather than an extremely slow three channel segmentation.

The software maintains depth information, which may change from frame to frame as the camera position moves. Sepia-2a Alpha Blending [9] used for compositing of subvolume images, requires the relative depth of each node.

3.1 Stony Brook Visual Computing Cluster

The Stony Brook Visual Computing Cluster consists of 33 dual-boot compute nodes connected with Gigabit Ethernet connections. One node serves as a front-end node to the cluster. Each node is an HP dual-processor Pentium Xeon 2.4GHz with 2.5 GB memory. Each node is also equipped with a VolumePro1000 volume rendering accelerator [13] with 1 GByte of memory and a GeforceFX5800 GPU with

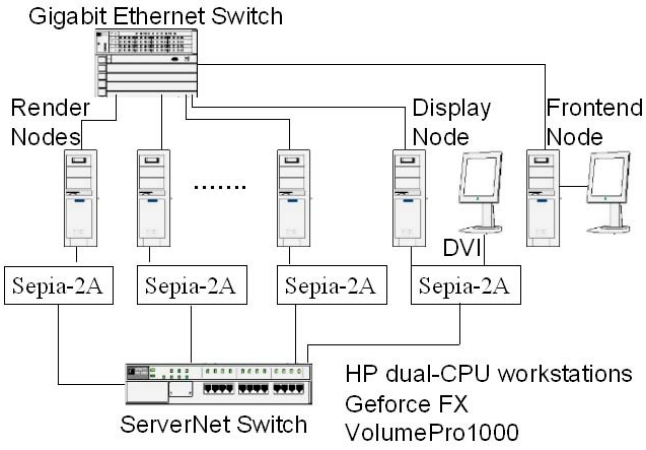


Figure 2: Stony Brook Visual Computing Cluster.

128MB memory. In addition, nine of these nodes are configured as a HP Market Development System (MDS) Visualization System based on the prototype Sepia-2a compositing architecture. Since the completion of this project we have upgraded the cluster to 66 nodes.

3.2 Sepia-2a Architecture

Compositing has been performed traditionally by graphics hardware accelerators [3, 11]. The HP Sepia architecture allows images to be composited at interactive rates by avoiding the frame buffer readback bottleneck [8]. Composition order is not static and compositing operations are not required to be communicative.

A single Sepia PCI board and its associated daughter board is added to each graphics workstation. The Sepia board gets its graphical image data through the DVI display port of a commodity OpenGL graphics card. The Sepia board connects to a ServerNet II high-bandwidth, low latency network. By communicating through the network, a series of such boards carries out the image computations resulting in a sort-last architecture in real time.

Our MDS Visualization System has eight render nodes and one display node with Sepia-2a boards. The Sepia architecture combines the images produced from individual data subsets and supports user mouse and keyboard interactive rendering. Figure 2 shows a block diagram of our cluster.

The Sepia architecture consists of two types of nodes: render nodes and display nodes. At the display location, a Sepia card configured for display mode, receives the final display-ready image from the network and transmits it through a display projector, flatpanel, high-resolution CRT monitor, or other display device. The Sepia-2a daughter board allows DVI output to the display board further in-

creasing speed-up. Each render node composes the stream from the previous node with the results of its own rendering and transmits the results to the next node.

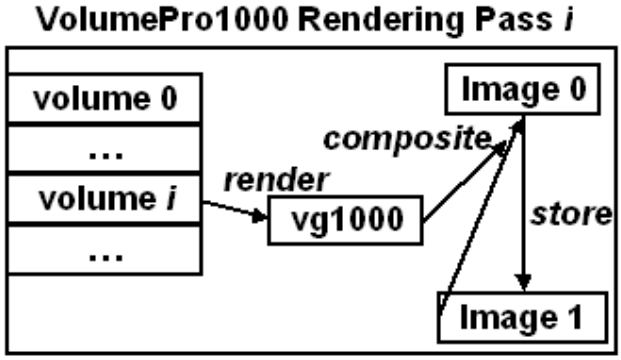


Figure 3: Multipass rendering on the VolumePro1000. Prior to rendering pass *i*, *Image* 1 buffer has image composited from all prior rendering passes.

3.3 Rendering on VolumePro

For the results presented here volumes were rendered on the VolumePro1000. Each render node has a VolumePro1000 with 1GB of memory of which approximately 850MB can be used for volume data. We leverage the VolumePro1000 board capability of rendering multiple volumes in a single context with on-board image compositing. Each subvolume is rendered as a separate pass, allowing subvolumes to be tightly cropped. The resulting images are accumulated into an image buffer which is located in the VolumePro board memory (see Figure 3). The end result is sent to the GPU frame buffer for acquisition by the Sepia-2a card at the end of each frame. The volume correction matrix is used to place each subvolume in its correct position within the whole volume. VolumePro rendering on an MDS cluster proceeds as follows:

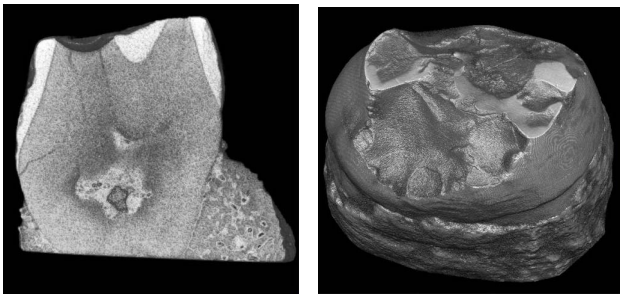
- Local processor updates viewing parameters.
- VolumePro1000 renders, composites subvolumes.
- Graphics card renders image to video memory.
- Sepia simultaneously receives pixels from local video memory and from upstream node.
- Sepia combines pixels, transmits to next node.
- Display node receives fully composited image.

4 Results

The effectiveness of the cluster is demonstrated by rendering and compositing several very high resolution CT scanned teeth and tooth fossils, Visible Male photographic

(a) Cebus Occlusal (b) Homo Sapiens

Figure 4: Rendered Teeth Images (image resolution 1280×1024): (a) One-channel Cebus Occlusal $2048 \times 2048 \times 663$; (b) Homo Sapiens $2048 \times 2048 \times 1589$.



(a) Baboon Fossil (b) Diademodon Fossil

Figure 5: Rendered Fossil Images (image resolution 1280×1024): (a) Baboon Fossil $2048 \times 2048 \times 1570$; (b) Diademodon Fossil $2048 \times 2048 \times 2028$.

image data, and seismic data. Region of Interest Cropping was used to reduce all except the seismic data. The Visible Male data was used to stress the MDS Visualization System, requiring all available VolumePro memory.

The very high resolution CT scanned teeth (see Figure 4) and tooth fossils (see Figure 5) consist of micro CT scanned images of 2048 pixels by 2048 pixels, each with a slice thickness of around 10 microns. We rendered a cebus apella tooth, a diademodon fossil, a baboon fossil, and a homo sapiens molar. The data sizes for the cebus apella, diademodon, baboon, and homo sapiens specimens consist of 663, 2028, 1570, and 1589 slices, respectively.

The seismic data is 8.3 GB, consisting of $1834 \times 1382 \times 783$ floating point with four bytes/voxel. Following standard practice of the Society of Exploration Geophysicists, it has been reduced to a 2.1 GB greyscale data set. It was rendered on three VolumePro1000 boards and composited on Sepia-2a boards (see Figure 6). The end-to-end render time was 0.45 seconds.

The full Visible Male data set, from the Visible Human

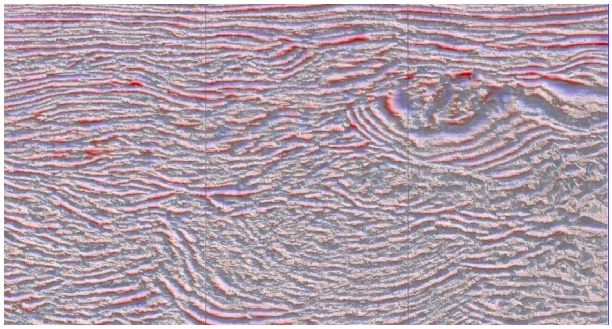


Figure 6: Composited seismic data $1834 \times 1382 \times 783$.

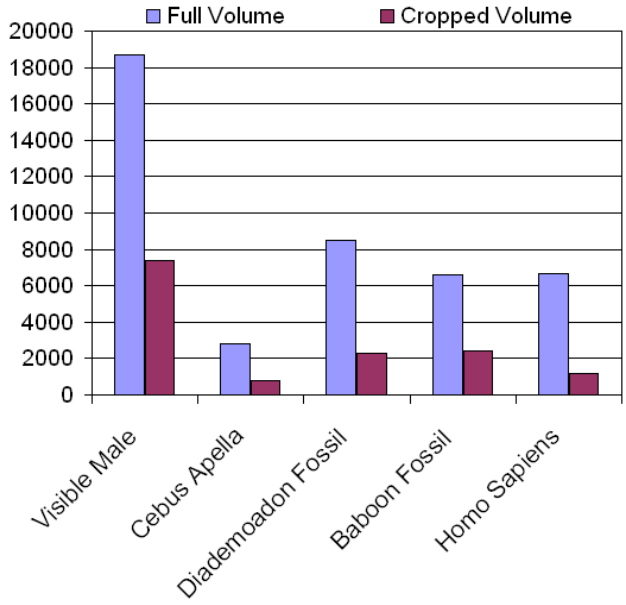


Figure 7: Data storage reduction after cropping and data distribution.

Project, is a sequence of axial anatomical images of 2048 pixels by 1216 pixels at 1 mm slices. The Visible Male color dataset has 1879 slices. With the addition of an alpha channel the data set is 18.7GB, more than double the capacity of our MDS System VolumePro1000 boards.

Region of Interest Cropping is used to reduce volume sizes for the full Visible Male set and several CT scanned teeth and tooth fossils. In our experiments, data is reduced by an average of 68% (see Figure 7).

We render and composite the full resolution Visible Male using eight render nodes (see Figure 8). The cropped full-scale Visible Male data fit within the VolumePro1000 memory of these nodes. On some render nodes multiple volumes are rendered within a single VLI context with VolumePro1000 multipass rendering (see Figures 1(d), 1(g) and 1(h)). Figure 9 shows frame rates for a varying number of nodes, each rendering a full screen 1280×1024 resolution

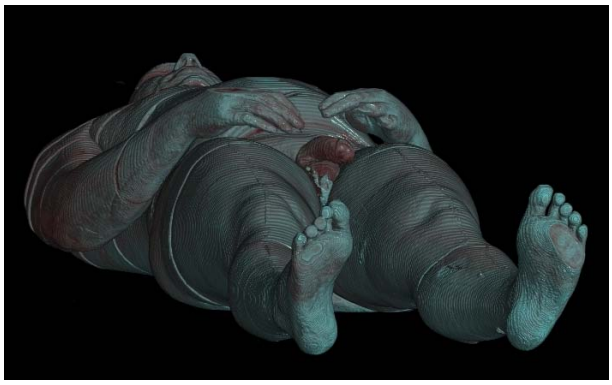


Figure 8: Composited full resolution four-channel Visible Male $2048 \times 1214 \times 1879$. Image resolution 1280×1024.

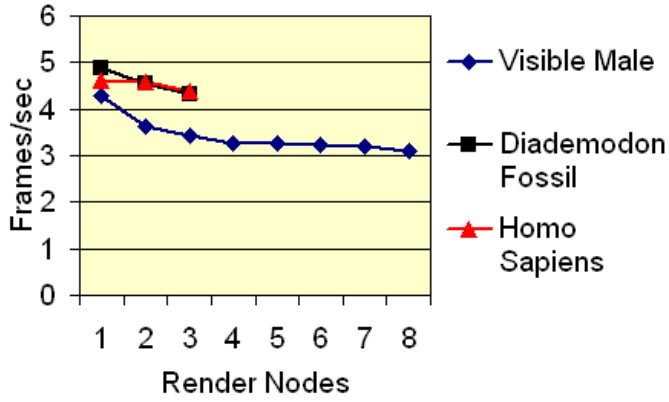


Figure 9: Rendering frame rates for 1280×1024 image resolution and approximately 1GB data per node.

image of volumetric data which is nearly the same size as the VolumePro1000 memory capacity. Volume sizes, preprocessing times, and volume distribution times are given in Table 1.

5 Conclusions and Future Work

We have developed a technique for out-of-core region of interest segmentation. By breaking the volume into cropped and pre-segmented blocks, we have made substantial reductions in memory requirements. This technique is most appropriate for volumes with substantial empty space.

Data acquisition typically results in corrupt data with ill-defined problems including noise, misaligned slices and empty slices. We allow blank slices and misaligned slices to appear in the rendered volume. An alternative would be to use methods developed for smoothing slices [14].

The results presented here are limited by the number of Sepia-2a composite boards in our cluster. VolumePro rendering with GPU compositing will allow us to use all avail-

Table 1: Volume sizes (MB), preprocessing (sec), and distribution times (sec).

	4-Channel RGBA	1-Channel CT			
	Visible Male	Cebus Apella	Diademodon Fossil	Baboon Fossil	Homo Sapiens
Resolution	2048x1216x1879	2048x2048x663	2048x2048x2028	2048x2048x1570	2048x2048x1589
Full volume size	18,717	2,780	8,506	6,585	6,665
Preprocessing	4605	2,515	7,800	6,240	2,398
Volume Distribution	1.76	0.38	1.26	0.45	0.47

able nodes. We also plan to implement parallel Region of Interest Cropping and GPU acceleration.

Acknowledgements

This work has been partially supported by NSF grant CCR-0306438. We thank HP for providing the Sepia-2a hardware and TeraRecon for supplying us with the Volume1000 hardware and the seismic data. In addition, we would like to thank the Stony Brook Anthropology Department for the teeth and fossil data, and the National Library of Medicine for the Visible Male data.

References

[1] R. Adams and L. Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis & Machine Intelligence, Volume 16*, pages 641–647, 1994.

[2] F. Dachille and A. Kaufman. GI-Cube: An architecture for volumetric global illumination and rendering. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 119–128, Aug. 2000.

[3] A. Ghosh, P. Prabhu, A. Kaufman, and K. Mueller. Hardware assisted multichannel volume rendering. *Computer Graphics International*, pages 2–7, July 2003.

[4] A. Heirich and L. Moll. Scalable distributed visualization using off-the-shelf components. *Proceedings of Symposium on Parallel & Large-Data Visualization and Graphics*, pages 55–59, Oct. 1999.

[5] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner, and J. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *Proceedings of SIGGRAPH*, pages 693–712, Aug. 2002.

[6] S. Lakare and A. Kaufman. OpenVL: The open volume library. *Third International Workshop on Volume Graphics*, pages 69–78, July 2003.

[7] W. Li, K. Mueller, and A. Kaufman. Empty space skipping and occlusion clipping for texture-based volume rendering. *Proceedings of Visualization*, pages 317–324, Oct. 2003.

[8] S. Lombeyda, L. Moll, M. Shand, D. Breen, and A. Heirich. Scalable interactive volume rendering using off-the-shelf components. *Symposium on Parallel & Large-Data Visualization and Graphics*, pages 115–121, Oct. 2001.

[9] L. Moll, A. Heirich, and M. Shand. Sepia: scalable 3D compositing using PCI pamette. *IEEE Symposium on Field Programmable Custom Computing Machines*, pages 146–155, April 1999.

[10] C. Morris and D. S. Ebert. Direct photographic volume rendering using multi-dimensional color-based transfer functions. *Proceedings of Eurographics/IEEE TCGVG Symposium on Visualization, VisSym*, pages 115–124, 2002.

[11] S. Muraki, E.Lum, K. Ma, M. Ogata, and X. Liu. A pc cluster system for simultateous interactive volume modeling and visualization. *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 95–102, Oct. 2003.

[12] H. Nishimura, T. Endo, T. Maruyama, J. Saito, and P. H. Christensen. Parallel rendering and the quest for realism: The KILAUEA massively parallel ray tracer. *SIGGRAPH Course Notes*, pages 1–59, Aug. 2001.

[13] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volumepro real-time ray-casting system. *Proceedings of SIGGRAPH*, pages 251–260, Aug. 1999.

[14] B. Reddy and B. Chatterji. An FFT-based technique for translation, rotation and scale-invariant image registration. *IEEE Transactions on Image Processing, Volume 5*, pages 1266–1271, 1996.

[15] R. Samanta, T. Funkhouser, K. Li, and J. P. Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of PCs. *SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 97–108, 2000.

[16] V. M. Spitzer, M. Ackerman, A. Sherzinger, and D. Whitlockand. The visible human male: A technical report. *Journal of the American Medical Informatics Association*, pages 118–130, Oct. 1996.

[17] G. Stoll, M. Eldridge, D. Patterson, A. Webb, S. Berman, R. Levy, C. Caywood, M. Taveira, S. Hunt, and P. Hanrahan. Lightning-2: A high-performance display subsystem for PC clusters. *Proceedings of SIGGRAPH*, pages 141–148, Aug. 2001.