Hardware Assisted Multichannel Volume Rendering

Abhijeet Ghosh* Poojan Prabhu† Arie E. Kaufman* Klaus Mueller*

Center for Visual Computing (CVC) and Computer Science Department
Stony Brook University
Stony Brook, NY 11794-4400, USA

Abstract

We explore real time volume rendering of multichannel data for volumes with color and multi-modal information. We demonstrate volume rendering of the Visible Human Male color dataset and photo-realistic rendering of voxelized terrains, and achieve high quality visualizations. We render multi-modal volumes utilizing hardware programmability for accumulation level mixing, and use CT and MRI information as examples. We also use multi-board parallel/distributed rendering schemes for large datasets and investigate scalability issues. We employ the VolumePro 1000 for real time multichannel volume rendering. Our approach, however, is not hardware-specific and can use commodity texture hardware instead.

Keywords: Multichannel, post-classification, transfer functions, multi-modal, volume mixing, Visible Human, terrain rendering, parallel rendering, distributed rendering, image compositing, VolumePro 1000, graphics hardware.

1. Introduction

Many volumetric datasets in medical and scientific applications have multichannel information. Examples include the Visible Human Male (VHM) dataset from the National Library of Medicine, which has CT, MRI as well as color information from photographs. Other examples are aerial or satellite color ortho-photographs of terrains (used in height-field visualization systems), and multi-spectral (remote-sensing) satellite data. This type of data offers exciting possibilities for photo-realistic volume visualization as well as better understanding of the data due to the multichannel information. In this paper, we explore hardware assisted volume rendering of multichannel data with examples taken from the VHM and terrain rendering applications.

Direct volume rendering is expensive in terms of computation and memory in comparison with surface based rendering. Volume rendering is preferred for medical visualization because of its ability to explore the internal structures such as bones and tissues. Polygon based rendering has been traditionally the choice for terrain visualization as terrains are mostly flat surfaces which can be modelled well with polygons and real time rendering can be achieved with texture mapping hardware using various LOD techniques [7]. Volume rendering still has many advantages for terrain visualization as the elevation maps can be easily voxelized into very high-resolution 3D volumes. Also, the voxel-based model is scene complexity independent, and offers a better representation for amorphous phenomena such as clouds, haze and fire. Therefore, volume rendering has been successfully applied to terrain visualization [3, 18].

Interactive rendering speed is required for the ease of exploration of large volumetric datasets and had been restricted for some time to high-end graphics workstations, due to the requirement of trilinear interpolations for image quality. Direct volume rendering with commodity 2D texture mapping [1, 5, 16] as well as 3D texture mapping have been successfully exploited for volume rendering. However, most PC platform texture mapping hardware have a limited on-board texture memory (typically 64-128MB) and slow texture-memory access rates, especially for 3D textures, limiting the size of volumes that can be rendered interactively in a single pass.

The above factors make special purpose hardware attractive for direct volume rendering of large volumes. The special purpose volume rendering hardware VolumePro 500 [13], which evolved out of our Cube-4 architecture [14], supports real time frame rates (30 frames/sec) for a 256^3 volume. The second generation hardware, the VolumePro 1000, is able to associate multiple channels of information (vector) with a voxel, not just a scalar density. It also has the capability of rendering super-volumes (i.e., volumes larger than the on-board memory) in real time with multi-pass rendering, and its pipeline is programmable, similar to commodity hardware. Given that this hardware is in its initial years of development, we hope that with future developments, the price/performance ratio will be more attractive.

*{abhijeet, poojan, ari, mueller}@cs.sunysb.edu
†now at Frounhofer Institute of Computer Graphics, Germany

In this paper, we demonstrate a post-classified single pass volume rendering of multichannel data (e.g., RGBα color volumes and multi-modal rendering of CT and MRI data) with graphics hardware, and multi-board real-time rendering for large datasets. We employ the VolumePro 1000 for real time rendering of multi channel data. However, our approach is not specific to VolumePro and can be implemented on commodity texture hardware as well.

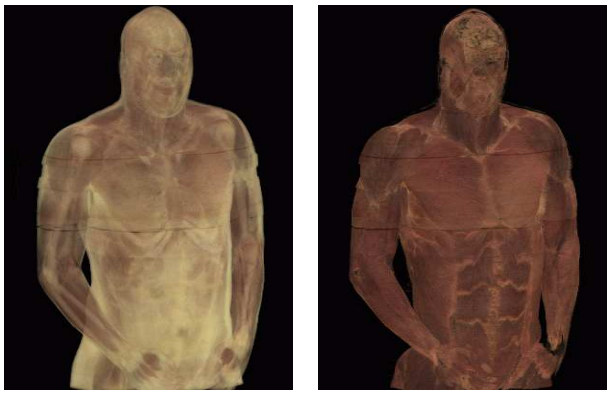
Section 2discusses the modeling and rendering of the VHM color volume and issues with transfer functions and gradients for multichannel data. We also explore multi-modal visualization of the CT and MRI datasets of the VHM in Section 3. Multi-board rendering for large datasets is discussed in Section 4. In Section 5, we briefly discuss the rendering pipeline of the VolumePro 1000 and some implementation issues. Alternative texture mapping hardware rendering schemes are also proposed. Finally, we discuss realistic terrain modeling and navigation as an application in Section 6.

2. Multichannel Color Volume Rendering

Color photographic volumes greatly simplify the task of creating realistic volume rendered images as the appropriate color for each voxel is already known from the photographs. Photographic volumes however offer a new challenge: determining the opacity for each voxel in the dataset. In traditional volume rendering, the design of effective color (1D to 3D mapping) and opacity (1D to 1D mapping) transfer functions for meaningful exploration of the data has been challenging [15]. In contrast, volume rendering from photographic data requires an opacity transfer function from the vector color fields (3D to 1D) and which is complicated by the non-linear nature of color spaces.

A way to solve this problem is to associate opacity values for the color voxels from an auxiliary density volume, such as CT, if available. There are two problems with this approach: a modality such as CT is good for external boundaries but does not capture internal details such as soft tissues well and the density volume has to be registered with the photographic volume. Registration of a large volume such as the VHM is difficult to automate.

Hence, we approach the direct volume rendering of the VHM color volume from the photographic information itself. We use the CIE L^* u^* v^* color space to obtain a perceptually uniform representation of the color volume [4]. The L^* component corresponds to the linear lightness of color, and hence has high values for bright areas of the volume (e.g., bones, skin and light colored organs such as the brain) (see Fig. 1a). The u^* component captures the chromatic changes in the red-green colors. Hence, it is sensitive to changes in the "redness" of tissues (e.g., muscle to bone) (see Fig. 1b). Our hardware rendering of the VHM is able



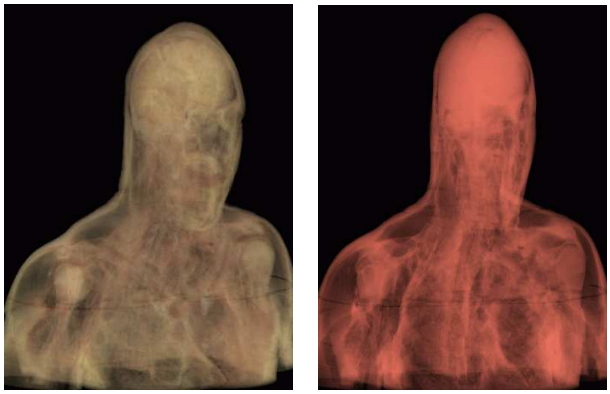
(a) (b)

Figure 1. Multichannel volume rendering: (a) L^* color component used as the density. (b) u^* color component used as the density.

to accurately simulate these properties of the color space. Currently, graphics hardware can store information in up to four fields, typically meant for RGBα. We assign the RGB information from the slices and the density information from the corresponding L^* or u^* color component that we calculate in a pre-process.

We propose a way to incorporate color difference gradients for shading in hardware rendering. Gradients can be calculated in hardware based on the voxel information. Hardware programmability allows us to access information of the neighboring voxels, which we can use for various kinds of gradient calculation. The changes in color values along *X*, *Y* and *Z*, expressed as the triple (*grad.x*, *grad.y*, *grad.z*), is the color difference gradient vector. We implement color difference gradient magnitude calculation with hardware (see Fig. 2a) by using the difference of the information in the Red channel across a voxel along the *X* direction as *grad.x*, and likewise for the other two color channels. The perceptual difference between two colors is to a good approximation proportional to the Euclidean distance between them. This is specially true for the CIELUV space. For color difference gradients in the CIELUV space, we sacrifice the true color volume rendering due to the limited number of color channels and render only with pseudo colors from an RGBα transfer function (see Fig. 2b).

We also incorporate gradient boundary enhancement by using first and second order directional derivatives along the gradient direction [11]. The first derivative in the direction of the gradient is the gradient magnitude itself and the second directional derivative can be well approximated by the gradient of the gradient magnitude. The first directional derivative can be currently calculated in hardware rendering. For the second directional derivative, we pre-calculate the first directional derivative and store them in separate channels. Then, using the existing algorithm, calculate the



(a) (b)

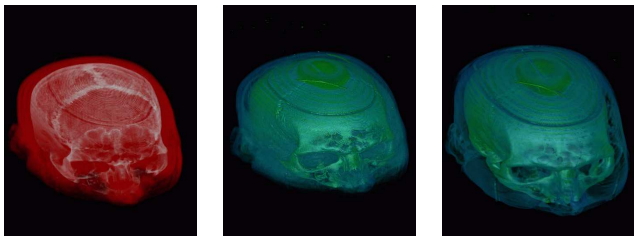
Figure 2. Multichannel volume rendering with hardware assisted gradients: (a) Color difference gradients of RGB color space. (b) Gradients of LUV space with second derivative along gradient direction.

gradient based on the information in these channels, resulting in second directional derivative on the fly (see Fig. 2b).

3. Multi-Modal Volume Rendering

In this section, we describe real-time multi-modal volume rendering. Various levels (image, accumulation and illumination) of volume mixing and rendering pipelines have been proposed for multi-modal rendering [2]. Image level intermixing is not efficient as it requires one rendering pass per volume and an additional merging, and also does not produce good quality visualization. Though accumulation level mixing produces good quality visualization, it is a slow process (mixing happens on a per sample basis along the ray) and our aim is to accelerate this using hardware. In our scheme, each modality is associated with a color channel for single-pass rendering in hardware. This scheme also ensures a color coding scheme for information from the different modalities. For the purpose of demonstration, we use information from two kinds of volumes, CT and MRI of the VHM dataset. However, our scheme is generic enough to be used for other multi-modal datasets.

A simple but lossy scheme for mixing data from two modalities such as CT and MRI is to sample from both using a density threshold, assigning value to the voxel from CT if the corresponding CT density is above the threshold (high density for bones and surfaces) and from MRI otherwise (low density for soft tissues). Hence, we get the best of both modalities. We use red color for MRI (density information replicated to Red and α channels) and gray for CT (density information replicated to all four RGBα channels) (see Fig. 3a) as our color coding scheme.

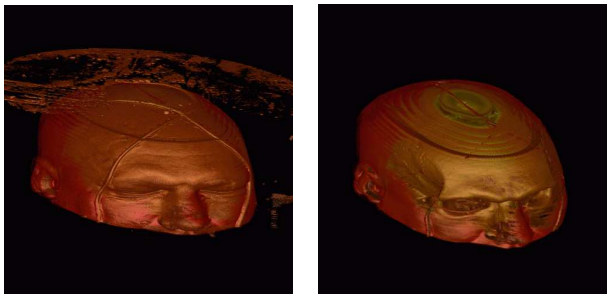


(a) (b) (c)

Figure 3. (a) Simple multi-modal volume mixing of CT and MRI data by sampling based on density threshold. Mixing with arithmetic operation: (b) gradients specified on CT while MRI is rendered as a point cloud, and (c) rendering with gradient modulation.

The above approach has the drawback of information being lost due to the thresholding during sampling. A better way to implement mixing would be to let the user select the threshold while exploring the data. We achieve this by employing the arithmetic and logical operation supported in hardware. We store the CT and MRI information in separate fields (color channels in hardware) of the voxel. Then, we compute CT - MRI and MRI - CT simultaneously and output the results of this Diff operation (with negative values clamped) on two separate color channels (green and blue in our case). The result is that for every voxel, only the greater of the two components of the voxel contributes to the final accumulation along the ray. This scheme is not lossy and the segmentation can be dynamically altered using transfer functions for data exploration. One issue that arises is that gradients need to to be calculated across the different modalities separately since they represent different volumes and this may not be supported in hardware. Hence, we specify gradients to be calculated only from one modality (voxel field) at a time, and not on the entire (multi-modal) voxel (see Fig. 3b). The above mixing scheme is used for the color channels only (for color coding), while we use Max(CT, MRI) for the α channel for well defined iso-surfaces using gradient modulation (see Fig. 3c).

Another approach to mixing is inclusive opacity, where all the volumes contribute to the voxel final color and/or opacity. A convenient way to identify voxels overlapped by both CT and MRI is by rendering them using a third orthogonal color (red in our case). We render the multi-modal volume with the function 1 - (CT - MRI) - (MRI - CT) for the red channel corresponding to voxels with information from both modalities. We still use the Diff operator for the green and blue channels and the Max operator for the α channel (see Fig. 4). More sophisticated operations such as conditionals should be soon available in graphics hardware adding flexibility to the volume mixing schemes.



(a) (b)

Figure 4. Multi-modal volume rendering with inclusive opacity: (a) Voxels with information from both CT and MRI; (b) Voxels with more CT.

4. Multi-Board Rendering

We utilize multiple units for rendering large volumetric datasets interactively. Many such visualization systems exist, ranging from those using multiple CPUs and graphics pipes in parallel [6] to cluster-based systems using special-purpose hardware such as VolumePro 500 [8, 12]. Here, we investigate the efficient parallelization of multiple graphics hardware boards on the PC platform.

We implement image-partitioned rendering by loading the entire volume on all the available boards, but restricting the range of the image and depth buffers to be filled by each board. This way, we distribute the rendering task to the boards uniformly. A limitation of this approach is that the size of the volume that can be rendered is limited by the memory size of the each board. Therefore, super-volumes have to be rendered using object-partitioned parallelism. A super-volume is divided into multiple sub-volumes, each of which can fit on one on-board memory. Each board renders its sub-volume and the resulting images have to be composited in order of depth. Hence, object-ordered parallelism seems more useful for large datasets.

We investigated distributed rendering on multiple PCs with one rendering engine per node as PC architecture does not commonly support multiple graphics boards. We setup one node as the master (control) and the others as rendering slaves. The master distributes the viewing parameters per frame to the slaves and collects the rendered images from them for compositing and display on a per frame basis. The slaves render their corresponding sub-volume after receiving the viewing parameters and send the rendered image to the master once every frame. This setup requires a high bandwidth low latency interconnect and network interface technology, such as Myrinet or Gigabit Ethernet, to support the sustained heavy network traffic.

Image compositing of rendered sub-volumes can be a computationally expensive task as the compositing operator has to be applied over every pixel of the image. We utilize the register combiner units on the texture mapping hardware (GeForce2) to carry out this per-pixel compositing and display at the master node in an efficient hardware accelerated fashion (see Fig. 5). The available sustained network bandwidth as well as the loading of the image data into texture memory are the overhead in this distributed rendering scheme and cost a few frames in performance.

(a) (b) (c)

Figure 5. Multi-board volume rending: (a)-(b) rendered sub-volumes; and (c) image composited on texture mapping hardware

We can also use multiple boards to cache large volumes ahead of rendering and at run-time only render the desired sub-volume. This is useful for terrain visualization (see Section 6) where the terrain may be large but only portions are visible at any point. Caching has been utilized for walk-throughs for reusing previously rendered images in subsequent frames [17]. We divide the terrain volume into slabs that can fit in the on-board memory. This way, we maintain real time frame rates for a fly-through by rendering only relatively small slabs at a time. This scheme also helps overcome the loading-unloading latency during fly-through as all required slabs are already on-board.

5. Implementation with VolumePro 1000

VolumePro 1000 supports real-time volume rendering of up to 513^3 volumes. The hardware performs the following basic operations on data at voxel or sample points: (1) Gradient estimation (central difference), (2) Classification/Interpolation (trilinear), (3) Illumination, and (4) Compositing. Alpha correction, accumulation, and early ray termination are included in the compositing process. The voxels in VolumePro memory are composed of up to a maximum of 4 fields. The classification function of VolumePro consists of transfer functions for color and α lookup and arithmetic and logical units.

We implement multichannel volume rendering with VolumePro 1000, utilizing calls of the Volume Library Interface (VLI) (C++ API for VolumePro). We render the color and multi-modal volumes on the VolumePro by assigning 8 bits per color channel or modality to a separate field of

the voxel. These fields can also be used to tag segmented data. Mapping of the voxel fields through transfer function LUTs is supported in VolumePro. This can be implemented with commodity texture hardware using multi-texturing and dependent texture lookups. The ALU units of VolumePro provide additional programmability, similar to the register combiner units of commodity graphics hardware.

We down-sampled the original VHM photographs from a resolution of 2048×1216 pixels to 512×256 pixels. Our color space conversion was performed by first converting RGB to CIE XYZ space using the *XYZitu601-1 (D65)* standard conversion matrix, and then converting to CIE L^* u^* v^* color space.

For multi-modal volume rendering, we modulated each voxel field with a separate transfer function and used the ALU units for implementing the Diff and Max operation for volume mixing. CT slices from the VHM are of 512×512 resolution and the MRI slices are 256×256. Hence, we first interpolate the MRI slices to 512×512 resolution. Also, the MRI data is available for every fourth slice of the CT. Therefore, we interpolate between slices with a large filter support to smooth out the staircase effect.

Though we experimented with multi-board VolumePro 1000 rendering on a single machine, we discovered it to be inefficient due to the PC architecture. We solve this issue by switching to distributed rendering with object ordered parallelism. Our test bed is a dual node PC cluster with one VolumePro per node. The nodes are connected with a high-speed 1 Gbps Myrinet point-to-point connection and we use the low level API GM for packet transfer over the Myrinet.

(a) (b)

Figure 6. (a) Original ortho-photograph (512×512) of the terrain; (b) High quality multichannel volume rendering of terrain ($1024 \times 64 \times 1024$) with embedded polygonal F-15.

Figure 7. Multichannel volume rendering of volumetric terrain with amorphous clouds

6. Applications and Results

We create realistic visualization of volumetric terrain as another application of multichannel color volume rendering on VolumePro 1000. One of the primary sources of data in terrain visualization is the terrain height field and the corresponding texture of the terrain. The latter is typically obtained from an aerial or satellite ortho-photograph. We construct the terrain dataset by first voxelizing the height-field data, creating an iso-surface of voxels. We scale all heights in our height-field data to a maximum height of 64 voxels as terrains do not generally have sharp variations in altitude. The terrain dataset is constructed as an RGBα volume, with one channel per field. At location (*x, y, z*), we obtain RGB information from the corresponding (*x, z*) location of the terrain ortho-photograph (texture) (see Fig. 6a). For the α channel we assign a constant high value in order to make the surface opaque.

Finally, we utilize VolumePro 1000 ability to embed polygonal objects within volumes to create the visual effect of an F-15 flying over volumetric terrain (see Fig. 6b). Though great for visualization, this is a costly process. We also produce high quality rendering of terrain with amorphous phenomenon such as clouds (see Fig. 7) and achieve interactivity with hardware volume rendering.

Our experiments are conducted using VolumePro 1000 boards with 1GB of on-board memory. Some of this memory is allocated for on-board image and depth buffers. Hence, we restrict the largest 32-bit RGBα color volume in a single pass to 950MB. For the visible Human Male, this evaluates to 512×256 resolution per slice and a total of 1871 slices. We divide the terrain volume into slabs, and restrict the slab size to $1024 \times 64 \times 1024$. We cache three such slabs on one board at a time for a total volume size of $3072 \times 64 \times 1024$.

For our parallel rendering experiments, we have experimented with up to four boards in parallel PCI (33MHz-32bit) slots of a Pentium III 1GHz machine, as well as in parallel PCI (66MHz-64bit) slots of an Athlon 1.2 GHz machine (see Table 1). Higher bandwidth of the 66MHz-64bit PCI bus gives consistently better performance. Unfortunately, the PCI slots on these machines share the same PCI bus and hence the performance does not scale well with the number of boards. Our distributed rendering experi-

Table 1. Frame rates for multi-board parallel rendering of VHM on one PC

PCI bus	Single Board	2 Boards on 1 PC	4 Boards on 1 PC
33MHz-32bit	8	14	18
66MHz-64bit	14	19	24

ment with VHM was configured using two nodes each with $512 \times 256 \times 936$ sub-volume and one or two boards. The frame rates for a 66MHz-64bit PCI bus are 25 for 1 board and 33 for 2 boards per node. Tiling smaller slabs of size $1024 \times 64 \times 1024$ within each board for volumetric terrain visualization provides real-time performance of 32 and 48 frames/sec for 32MHz-32bit and 66MHz-64bit PCI bus, respectively.

7. Conclusions and Future Work

We have demonstrated high-quality visualization of multichannel data with volume rendering through proper modeling of the data and hardware assisted single pass rendering. We achieve various mixing schemes for single pass multi-modal volume rendering using hardware units. Interactive speed has been achieved for super-volumes using multi-board rendering. Though we have used special purpose VolumePro 1000, our approach is generic and can be realized on commodity texture hardware too.

We are in the process of setting up a visualization PC cluster, initially with 12 rendering nodes and a display client. One issue that we need to resolve in this cluster is that of fast image composition, maybe using the efficient binary swap composition strategy [9]. We intend to overcome the slow reads of image data from the rendering unit using special-purpose compositing hardware [10, 12], in order to match the compositing speed with that of the rendering.

Acknowledgements

This work is supported by ONR grant N000140110034. We thank Andy Vesper and TeraRecon for information on the VolumePro 1000, and thank Kevin Kreeger, Wei Li and Huamin Qu for valuable discussions.

References

[1] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proc. ACM Symp. on Volume Visualization*, pages 91–98, 1994.

[2] W. Cai and G. Sakas. Data intermixing and multi-volume rendering. *Computer Graphics Forum*, 18(3):359–368, September 1999.

[3] D. Cohen-Or, E. Rich, U. Lerner, and V. Shenkar. A real-time photo-realistic visual flythrough. *IEEE Trans. on Visualization and Computer Graphics*, 2(3):255–265, September 1996.

[4] D. Ebert, C. Morris, P. Rheingans, and T. Yoo. Designing effective transfer functions for volume rendering from photographics volumes. *IEEE Trans. on Visualization and Computer Graphics*, 8(2):183–197, April-June 2002.

[5] K. Engel, M. Kraus, and T. Ertl. High quality pre-intergrated volume rendering using hardware-accelerated pixel shading. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware 2001*, pages 9–16, 2001.

[6] J. Kniss, P. McCormick, A. McPherson, J. Ahrens, J. Painter, A. Keahey, and C. Hansen. Interactive texture-based volume rendering for large data sets. *IEEE Computer Graphics and Applications*, 24(4):52–61, July-August 2001.

[7] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. In *Proc. SIGGRAPH 96*, pages 109–118, August 1996.

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

[9] K. 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-August 1994.

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

[11] C. Morris and E. Ebert. Direct volume rendering of photographic volumes using multi-dimensional color-based transfer functions. In *EUROGRAPHICS - IEEE TCVG Symp. on Visualization*, pages 115–124, 2002.

[12] S. Muraki, M. Ogata, K. Ma, K. Koshizuka, K. Kajihara, X. Liu, Y. Nagano, and K. Shimokawa. Next generation supercomputing using PC clusters with volume graphics hardware devices. In *Proc. IEEE Supercomputing Conference*, November 2001.

[13] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volumepro real-time ray-casting system. In *Proc. SIGGRAPH 99*, pages 251–260, August 1999.

[14] H. Pfister and A. Kaufman. Cube-4: A scalable architecture for real-time volume rendering. In *Proc. IEEE Symp. on Volume Visualization*, pages 47–54, October 1996.

[15] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Shroeder, L. Avila, K. Martin, R. Machiraju, and J. Lee. The trasfer function bakeoff. *IEEE Computer Graphics and Applications*, 21(3):16–22, May-June 2001.

[16] C. Rezk-Salama, K. Engel, M. Bauer, G. Griener, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using muti-textures and multi-stage rasterization. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware 2000*, pages 109–118, 2000.

[17] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proc. SIGGRAPH 96*, pages 75–82, August 1996.

[18] M. Wan, H. Qu, and A. Kaufman. Virtual flythrough over a voxel-based terrain. In *IEEE Virtual Reality Conference*, pages 53–60, 1999.