Architectures for Real-Time Volume Rendering

Hanspeter Pfister

TR99-18 April 1999

Abstract

Over the last decade, volume rendering has become an invaluable visualization technique for a wide variety of applications. This paper reviews three special-purpose architectures for interactive volume rendering: texture mapping, VIRIM, and VolumePro. Commercial implementations of these architectures are available or underway. The discussion of each architecture will focus on the algorithm, system architecture, memory system, and volume rendering performance.

Future Generation Computer Systems, Vol. 15, 1999

Architectures for Real-Time Volume Rendering

Hanspeter Pfister
Mitsubishi Electric Research, 201 Broadway
Cambridge, MA 02139, U.S.A.
pfister@merl.com

TR-99-18 April 1999

Abstract

Over the last decade, volume rendering has become an invaluable visualization technique for a wide variety of applications. This paper reviews three special-purpose architectures for interactive volume rendering: texture mapping, VIRIM, and VolumePro. Commercial implementations of these architectures are available or underway. The discussion of each architecture will focus on the algorithm, system architecture, memory system, and volume rendering performance.
Color pictures are available at http://www.elsevier.nl/locate/future.

Appeared in Future Generation Computer Systems, Vol. 15, 1999

Accepted at FGCS 9 September, 1998

1 Introduction

Visualization of scientific, engineering or biomedical data is a growing field within computer graphics. In many cases, the objects or phenomena being studied are volumetric, typically represented as a three-dimensional grid of volume elements, or voxels. Examples of volume data include 3D sampled medical data (CT, MRI), simulated datasets from computational fluid dynamics, or computed finite element models. One of the key advantages of volumetric data is that, unlike surface-based representations, it can embody interior structure of the objects. Additionally, operations such as cutting, slicing, or tearing, while challenging for surface-based models, can be performed relatively easily with a volumetric representation [Gib97].

Volume rendering generates images directly from the volume data without intermediate surface models. It allows the display of internal structures, including amorphous and semi-transparent features. Voxels are either processed in image-order or object-order to generate an image. Image-order algorithms iterate over all pixels of the output image and determine the contributions of voxels towards each pixel. Ray-casting is the most commonly used image-order technique [Lev88]. Rays are cast from the viewpoint into the volume and the contributions of voxels along each ray are used to determine pixel colors. Object-order algorithms iterate over the volume data and determine the contribution of each voxel to the pixels. A typical object-order technique is splatting, which convolves every voxel with a 3D reconstruction filter and accumulates the voxels contribution on the image plane [Wes91].

While volume rendering is a very popular visualization technique, the lack of interactive frame-rates has limited its widespread use. Volume rendering is very memory and computation intensive. To render one frame typically takes several seconds. Highly optimized software techniques for volume rendering try to address this problem by using pre-processing to compute, for example, object illumination or regions of the volume that can be skipped during rendering [LL94, Lev90]. However, pre-processing prohibits immediate visual feedback during parameter changes. Each time rendering parameters such as voxel transparency or illumination are changed, the lengthy pre-processing must be repeated before the new image is displayed. Furthermore, the pre-computed values typically increase the storage requirements by a factor of three to four.

To overcome these limitations, several Universities (e.g., University of Mannheim [GPR+94], University of Tübingen [KS94, KS97], State University of New York at Stony Brook [PK96, BK97]) and companies (e.g., Mitsubishi Electric [OPL+97] and Japan Radio Corporation) have undertaken the development of special-purpose hardware for volume rendering. Hardware accelerators aim to provide real-time frame rates, typically defined to be between 10 and 30 frames per second. Real-time visual feedback allows for interactive experimentation with different rendering parameters. In addition, because hardware does not require pre-processing, it allows visualization of dynamically changing volume data, such as data from interactive tissue cutting during surgical simulation, or continuous data input from 3D ultrasound.

The next section presents general issues related to the design of high-performance volume rendering architectures. The focus is on the design of high-bandwidth memory systems which are the basis for all architectures presented in this paper. Sections 3 through 5 present the architectures of texture mapping, VIRIM, and VolumePro, respectively. Section 6 concludes the paper with a brief summary of the main features of these architectures.

2 Architectural Challenges

The computational demands of volume rendering require the use of a high degree of hardware parallelism. Pipelining and unit replication are the two main forms of parallelism found in most high-performance architectures. A pipeline consists of a sequence of stages through which a computation and data flow. New data is input at the start of the pipeline while other data is being processed throughout the pipeline. Unit replication refers to using multiple processing units, each working on a different stream of data. Combining pipelining and unit replication by using parallel processing pipelines achieves a high level of parallelism and performance. Important issues are the interconnections and data paths between stages of different pipelines.

Because volume rendering is memory intensive, the design of the memory system is critical in volume rendering architectures. The memory systems includes a suitable memory hierarchy, fast memory caches, and memory bus architecture. A determining factor of the maximum achievable memory bandwidth is the performance of dynamic random-access memory (DRAM), the fundamental building block of all memory systems. In the last 30 years the speed of microprocessors has increased 1000-fold whereas the speed of DRAM has only increased by a factor of 20 [Sak97]. As shown in Figure 1, recent architectural improvements to DRAM modules have increased their sequential access bandwidth [Kat97]. However, their random access performance remains low and approximately constant across different DRAM technologies.

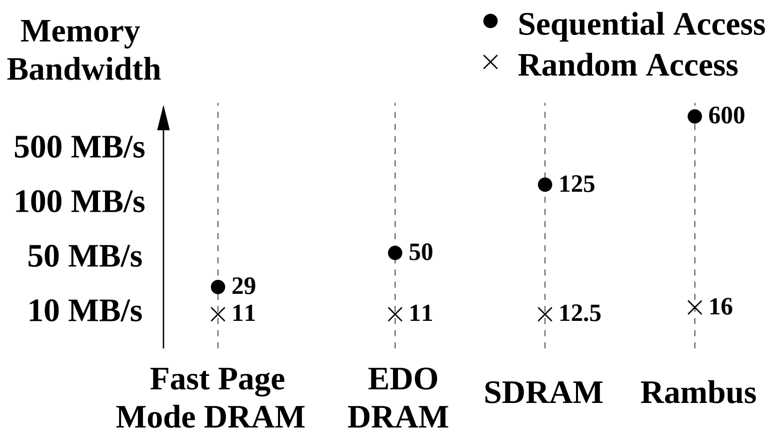


Figure 1: *This graph plots memory bandwidth (in MBytes per second, logarithmic scale) for different DRAM technologies.* Fast page mode DRAM *allows fast access to an entire row (called a page) of the internal memory array.* Extended data out DRAM *(EDO DRAM) includes an extra pipeline stage in the output buffer.* Synchronous DRAM *(SDRAM) has a high-speed synchronous interface and multiple internal memory banks.* Rambus DRAM *uses a high-speed packet-type memory interface.*

A common method to increase memory bandwidth for regular, systematic access to data is a technique called interleaving. The idea is to subdivide the data into smaller subsets that can be distributed uniformly among different physical memories. The simplest and most common form of interleaving is called low-order interleaving [Fly95]. It assigns successive memory addresses to distinct memory modules. For m memory modules, enumerated from 0 to $(m-1)$, memory address a is assigned memory module number $k = a \bmod m$.

Figure 2 shows the resulting partitioning of the address space across memory modules. The index i into the memory module is calculated as $i = \lfloor \frac{a}{m} \rfloor$, where $\lfloor\rfloor$ indicates the floor, or the next lower integer, of the expression. The number of memory modules in an interleaved memory system is called the degree of interleaving. Alternatively, a memory system is said to be m-way interleaved.

Notice in Figure 2 that voxels are stored successively in rows across the memory. Low-order interleaved memory performs best for consecutive row access, but performance breaks down for column access. Accesses against the storage order may occur frequently in volume rendering, for example, in object-order algorithms. One solution is to store the dataset three times, once for each main axis [LL94]. However, data duplication increases the high storage requirements of volume rendering.

The architectures surveyed in this paper implement interleaved memory systems that support high-bandwidth access to volume data without pre-processing and data duplication. The remainder of this paper describes texture mapping hardware, the Virtual Reality in Medicine (VIRIM) system, and Mitsubishi's VolumePro system. We discuss the algorithms these systems implement, their system architectures with emphasis on the memory system, and their volume rendering performances.

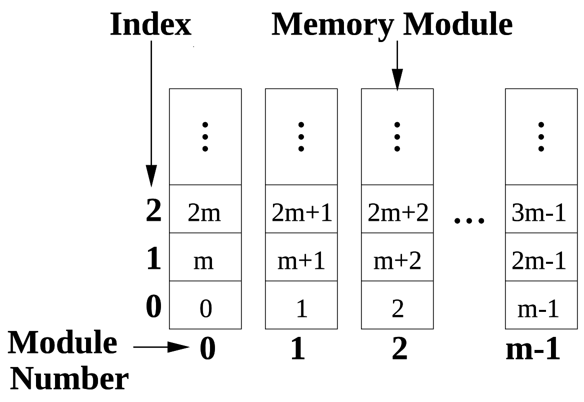


Figure 2: *Low-order interleaved memory system with m memory modules. Memory address a is assigned to memory module k according to $k = a \bmod m$.*

3 Texture Mapping Hardware

Texture mapping hardware is a common feature of modern 3D polygon graphics accelerators. It can be used for volume rendering by applying a method called *planar texture resampling* [CCF94]. The volume is stored in 3D texture memory and resampled during rendering by extracting textured planes parallel to the image plane (see Figure 3). Lookup tables map density to RGBA color and opacity. The resulting texture images

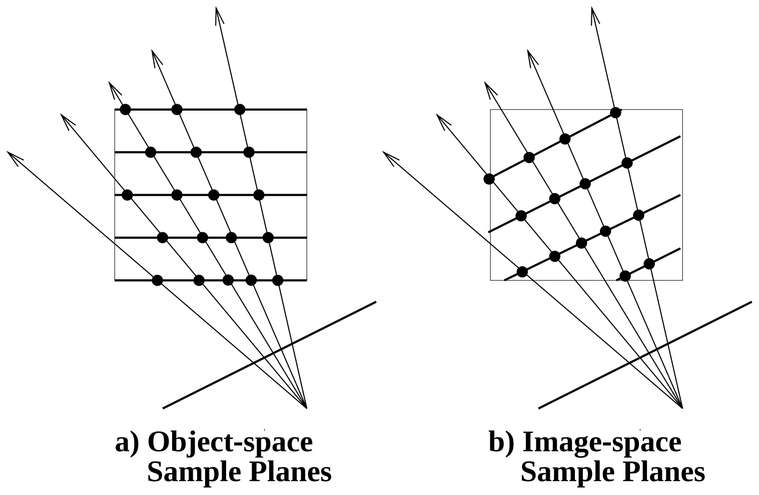


Figure 3: *Planar texture resampling.*

are combined in back-to-front visibility order using compositing [PD84]. Each texture sample is assigned an opacity, called the alpha component. An alpha of 1.0 implies a fully opaque sample, and an alpha value of 0.0 implies a fully transparent sample. When two texture images are combined, the background color is blended with the foreground color using the alpha component to linearly interpolate between the colors.

High-quality shading effects of the volume object require gradients which reflect the rate and direction of change in the volume data. Most volume rendering algorithms use the central difference gradient, which is computed by local differences between voxel values in all three dimensions [HB86]. Using these gradients, a local illumination model is applied to each texture sample to shade the volume object.

Figure 4 shows several texture mapped images of a wrist rendered with different gradient methods.

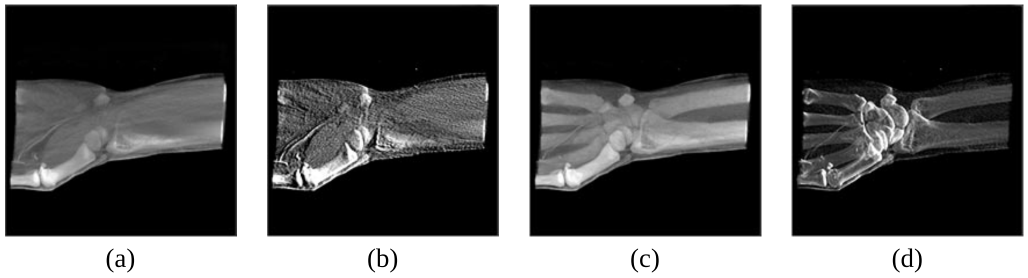


Figure 4: *Several renderings of a wrist dataset (*$256 \times 256 \times 53$*) using texture mapping hardware and multi-pass gradient estimation. a) 1 pass: No shading. b) 2 passes: Slices of gradients are calculated by subtracting co-planar texture slices, one shifted in direction of the light source. c) 3 passes: Gradients are resampled from a separate edge filtered volume. d) 10 passes: Gradients are estimated using central differences between axis aligned texture slices in all three dimensions. Images courtesy of David Heath, Johns Hopkins University.*

Texture engines do not directly support estimation of gradients or per-sample illumination in hardware. Gradients are typically pre-computed and stored in an additional volume or are computed on-the-fly using multi-pass methods. However, multi-pass rendering and shading in software greatly reduce the achievable frame rate. An alternative is to use a shading approximation by pairwise subtracting co-planar texture slices, one shifted in direction of the light source [PAC97].

System Architecture

Texture mapping hardware is part of the raster graphics rendering pipeline shown in Figure 5 [FvDFH90]. During planar texture resampling, the geometry processor computes the texture coordinates of each vertex of

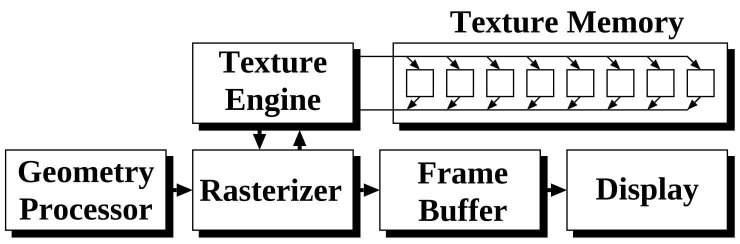


Figure 5: *Texture mapping system architecture.*

a resampling slice. The rasterizer is connected to the texture engine which resamples the texture data stored in texture memory using tri-linear interpolation. The resampled texture slices are then accumulated into the frame buffer using compositing [PD84].

High-performance 3D graphics engines, such as the SGI Reality Engine [Ake93], use eight-way interleaved texture memory. Tri-linear interpolation requires a cell of eight adjacent voxels. Eight-way memory interleaving allows the texture engine to fetch any tri-linear cell in one memory access. A voxel with address $a = [zyx]$ is stored in memory module k at index i as follows:

$$\begin{aligned} k &= (x \mod 2) + 2\,(y \mod 2) + 4\,(z \mod 2), \\ i &= \lfloor \frac{x}{2} \rfloor + 2\,\lfloor \frac{y}{2} \rfloor + 4\,\lfloor \frac{z}{2} \rfloor. \end{aligned} \quad (1)$$

Figure 6 shows the resulting assignment of voxels to memory modules in three dimensions.

Texture Performance

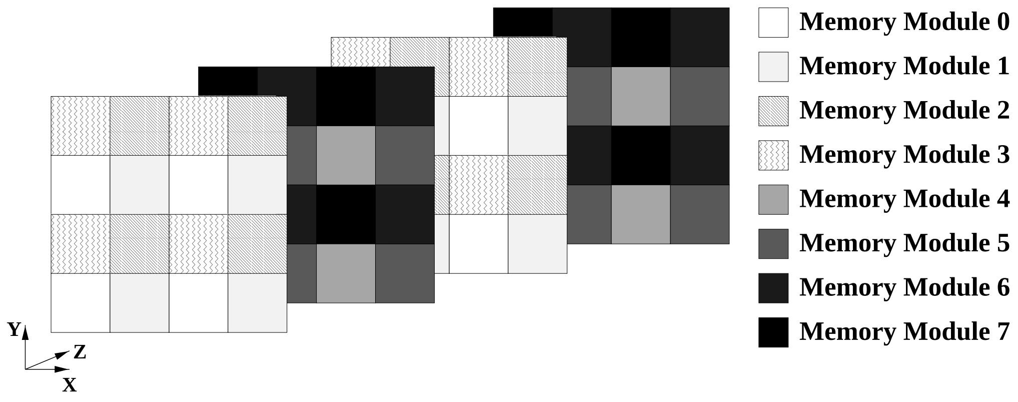


Figure 6: *Assignment of voxels to memory modules in an 8-way interleaved texture memory.*

The best published texture rendering performances have been achieved on SGI Reality Engine 2 systems with multiple texture engines or Raster Managers (RMs). Hemminger et. al [HCN94] report a series of results with two RMs, rendering a $256 \times 256 \times 32$ volume at 12 frames per second. Cabral et al. [CCF94] use four RMs to render a $512 \times 512 \times 64$ dataset at 10 frames per second. These results translate roughly into a maximum performance of 160 million unshaded tri-linear samples per second for four RMs. However, these published results do not consider gradient estimation or shading.

4 Virtual Reality in Medicine (VIRIM)

VIRIM is an object-order volume rendering engine developed and built at the University of Mannheim [GPR+94] and commercially distributed by the Volume Graphics GmbH, Heidelberg, Germany. VIRIM implements *projection plane ray-casting.* In this method, the rays belonging to a particular scanline of the image reside on the same projection plane (see Figure 7). Planes of consecutive scanlines are resampled in

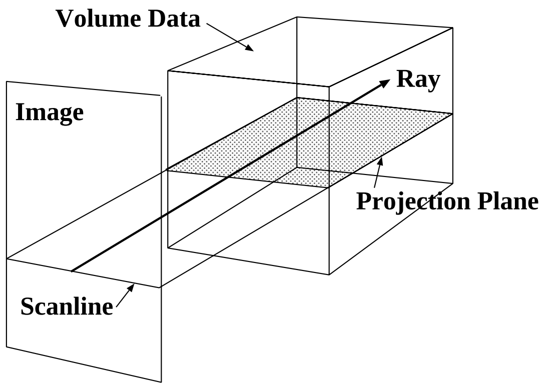


Figure 7: *Projection plane ray-casting.*

top-to-bottom order from the volume data. The image is produced by casting rays in viewing direction inside projection planes.

System Architecture

The hardware of VIRIM is divided into a geometry unit for volume resampling and a ray-casting unit for image generation (see Figure 8). Resampling of the dataset is performed by tri-linear interpolation. The

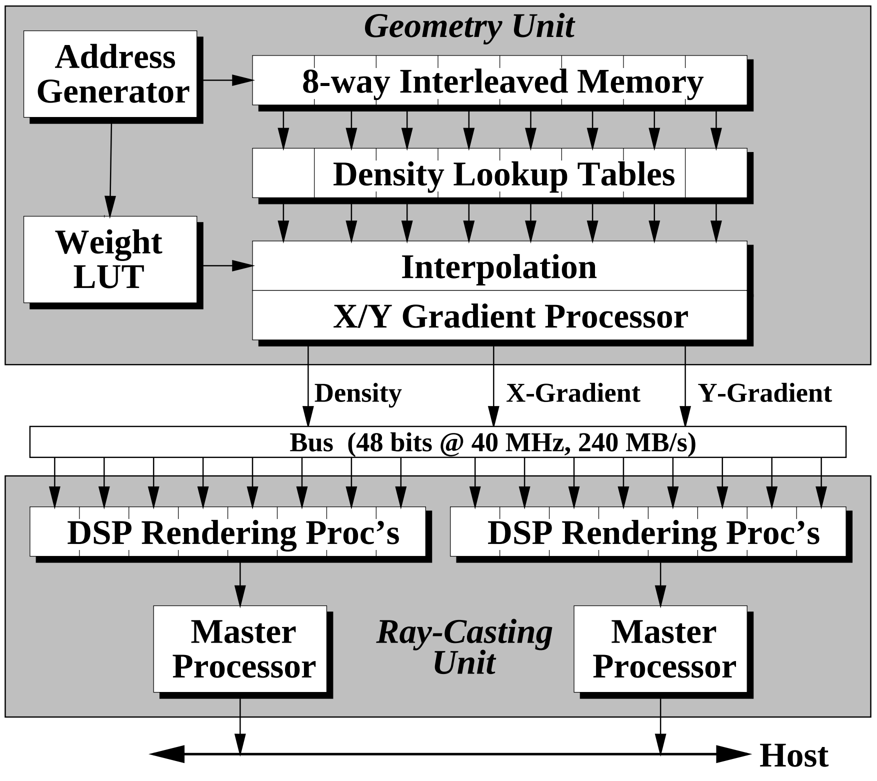


Figure 8: *VIRIM Architecture.*

volume data is stored in a dedicated 8-way interleaved memory system, identical in organization to the 3D texture memory discussed in Section 3. Before interpolation, the voxel values are mapped onto density values using a density lookup table). Only the X- and Y-component of the gradient are estimated using a 2D gradient operator. The sample density and the X- and Y-gradient components are transmitted to the ray-casting unit over the geometry bus which has a peak transfer rate of 240 MBytes per second.

Using the sample and gradient values of the resampled dataset, the ray-casting unit generates the final image. In order to allow maximum flexibility, VIRIM uses 16 programmable digital signal processors (DSPs). This allows the system to implement high-quality illumination models, such as the Heidelberg ray-tracing algorithm which includes shadowing [MMSE91]. A local master processor on the ray-casting board collects all scan lines of the final image from the DSP memories and transfers the results to the host system.

VIRIM Performance

The maximum dataset size for 16-bit voxels is 256^3, and the maximum read-out rate from volume memory is 640 MBytes per second. VIRIM achieves up to 2.5 frames per second for $256 \times 256 \times 128$ datasets, although higher performance may be achieved by duplicating the dataset across multiple VIRIM engines [HMK+95]. The VIRIM geometry unit generates a maximum of 36 million tri-linear samples per second.

5 VolumePro

The VolumePro system is based on the Cube-4 volume rendering architecture developed at SUNY Stony Brook [PK96]. Mitsubishi Electric licensed the Cube-4 technology and developed the Enhanced Memory Cube-4 (EM-Cube) architecture [OPL+97]. The VolumePro system, an improved commercial version of EM-Cube, is currently underway at Mitsubishi Electric.

VolumePro is a highly parallel architecture based on the *template-based ray-casting* algorithm shown in Figure 9 [YK92, SS92]. Rays are sent into the dataset from each pixel on a base plane, which is co-planar to

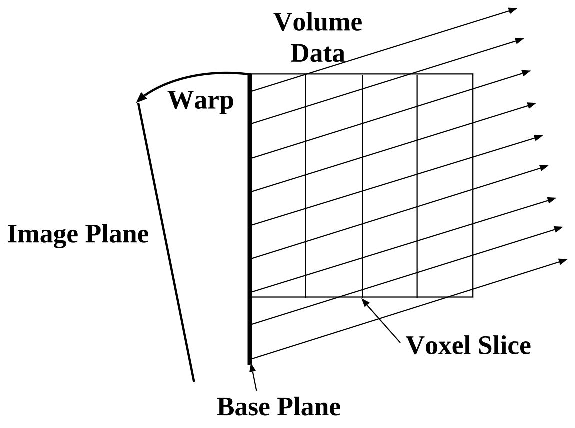


Figure 9: *Template-based ray-casting.*

the face of the volume data that is most parallel and nearest to the image plane. Because the image plane is typically at some angle to the base-plane, the resulting base-plane image is warped onto the image plane.

The main advantage of this algorithm is that voxels can be read and processed in planes of voxels (so called slices) that are parallel to the base-plane. Within a slice, voxels are read from memory a scanline of voxels at a time, in top to bottom order. This leads to regular, object-order data access.

VolumePro System Architecture

VolumePro will be implemented as a PCI card for Windows NT computers. The card will contain one volume rendering ASIC (called the *vg500*) and 32, 64, or 128 MBytes of volume memory. The warping and display of the final image will be done on an off-the-shelf 3D graphics card with 2D texture mapping. The vg500 volume rendering ASIC, shown in Figure 10, will contain four identical rendering pipelines, arranged side by side, running at 133 MHz each. The vg500 also contains interfaces to voxel memory, pixel memory, and the PCI bus. Each pipeline communicates with voxel and pixel memory and two neighboring pipelines.

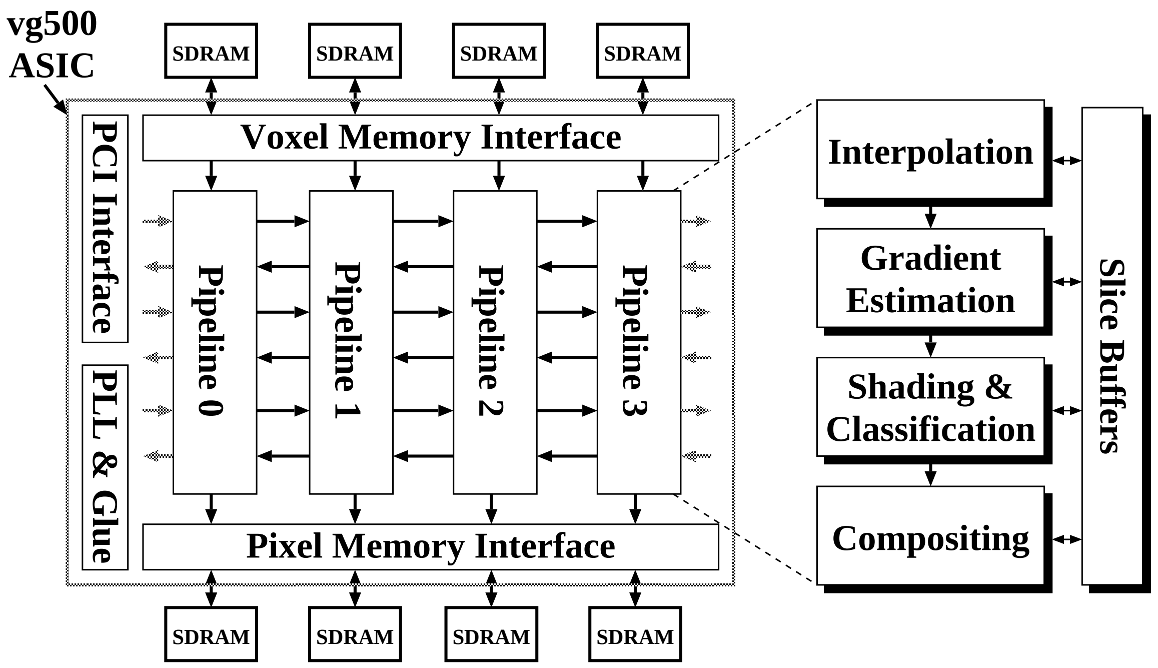


Figure 10: *The vg500 volume rendering ASIC with four identical ray-casting pipelines.*

Pipelines on the far left and right are connected to each other in a wrap-around fashion (indicated by grey arrows in Figure 10). A main characteristic of VolumePro is that each voxel is read from volume memory exactly once per frame. Voxels and intermediate results are cached in so called slice buffers so that they become available for calculations precisely when needed.

Each rendering pipeline implements ray-casting and sample values along rays are calculated using tri-linear interpolation. A 3D gradient is computed using central differences between tri-linear samples. The gradient is used in the shader stage, which computes the sample intensity according to the Phong illumination model. Lookup tables in the classification stage assign color and opacity to each sample point. Finally, the illuminated samples are accumulated into base plane pixels using front-to-back compositing.

Volume memory uses 16-bit wide synchronous DRAMs (SDRAMs) for up to 128 MBytes of volume storage. $2 \times 2 \times 2$ cells of neighboring voxels, so called miniblocks, are stored linearly in volume memory. Miniblocks are read and written in bursts of eight voxels using the fast burst mode of SDRAMs. In addition, VolumePro uses a linear skewing of miniblocks [KB88]. Skewing guarantees that the rendering pipelines always have access to four adjacent miniblocks in any of the three slice orientations. A miniblock with position $[xyz]$ in the volume is assigned to the memory module k as follows:

$$k = (\lfloor \frac{x}{2} \rfloor + \lfloor \frac{y}{2} \rfloor + \lfloor \frac{z}{2} \rfloor) \bmod 4. \tag{2}$$

VolumePro Performance

Each of the four SDRAMs provides burst-mode access at up to 133 MHz, for a sustained memory bandwidth of $4 \times 133 \times 10^6 = 533$ million 16-bit voxels per second. Each rendering pipeline operates at 133 MHz and can accept a new voxel from its SDRAM memory every cycle. Over 500 million tri-linear samples per second is sufficient to render 256^3 volumes at 30 frames per second.

6 Summary

Table 1 lists the main characteristics of the volume rendering accelerators that have been presented in this paper. Figure 11 shows texture rendered images, courtesy of David Heath, Johns Hopkins University. Figures 12 shows images created on VIRIM at the University of Mannheim, Germany. And Figure 13 shows images generated by a C++ simulation of the VolumePro system. Color versions of these pictures are available at http://www.elsevier.nl/locate/future.

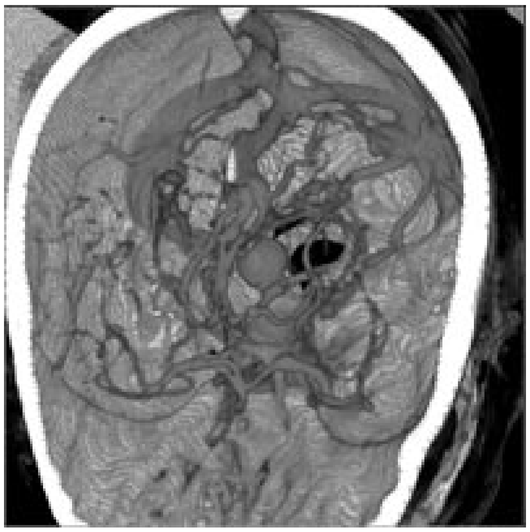
	Texture Mapping	**VIRIM**	**VolumePro**
Algorithm	Planer texture resampling. Gradients with multi-pass rendering, shading in software.	Projection plane ray-casting. 2D gradients in hardware, shading using DSPs.	Template-based ray-casting. 3D gradients and Phong illumination in hardware.
Memory	8-way interleaved. Volume size limited by texture memory only	8-way interleaved. 256^3 volumes, 16 bits per voxel.	Burst mode and linear skewing of miniblocks. 256^3 volumes, 16 bits per voxel.
Parallelism	Pipelining.	Pipelining and 16 parallel ray-casting DSPs.	Four parallel rendering pipelines.
Implementation	Part of 3D graphics engines, available now.	Large deskside machine, available now.	PCI card for Windows NT, available in 1999.
Performance	160M samples/sec	36M samples/sec	533M samples/sec

Table 1: Comparison of architectural features.

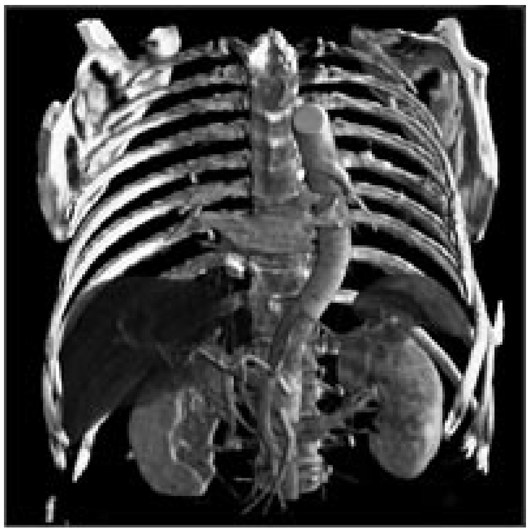
Texture hardware offers a natural integration of volume datasets into traditional polygon graphics accelerators. This provides combined rendering of volume data and embedded geometry. The availability of texture systems is increasing with the advent of 3D graphics in PC systems. However, current texture systems do not fully address the needs of volume rendering applications. The lack of gradient estimation and directional shading in hardware prohibits high-quality rendering at real-time frame rates.

VIRIM and VolumePro use high quality ray-casting, and both architectures implement gradient estimation and per-sample illumination in hardware. The image quality of both architectures is very good. The main disadvantages of VIRIM are its large physical size and the comparatively low frame rates. In contrast, VolumePro contains a single ASIC, capable of rendering a data set of 256^3 voxels at 30 frames per second. VolumePro is targeted to PC class computers running Windows NT and will be available at PC price levels. However, both VIRIM and VolumePro do not allow arbitrary intermixing of volumes and polygons.

The similarities between texture mapping and volume rendering will ultimately lead to an integration of these approaches into a unified architecture. The lessons learned from special-purpose volume rendering hardware will lead to texture systems that implement all elements of the volume rendering pipeline in hardware. Alternatively, volume rendering architectures will include support for embedded geometry and advanced texturing. The architectures surveyed in this paper are a first and important step in this direction.

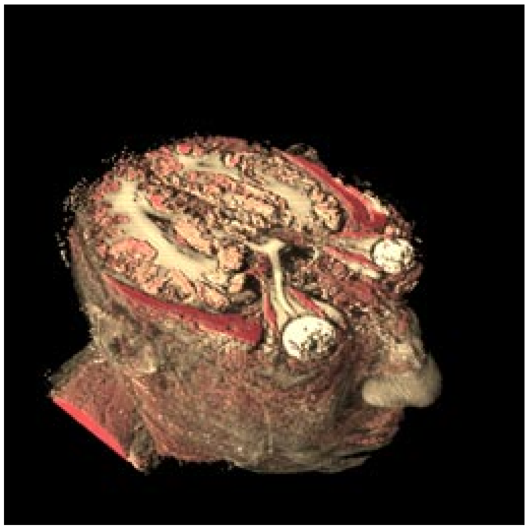


(a) Aneurysm, 9 frames/sec

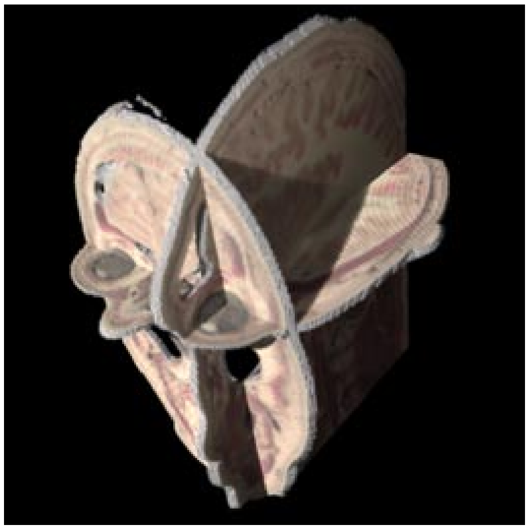


(b) Thorax, 1 frame/sec

Figure 11: *Texture mapped images of medical CT data. a) Aneurysm (*$256 \times 256 \times 104$*), no shading. b) Human thorax (*$256 \times 256 \times 11$*), central difference gradients and shading. Performance numbers are for a dual-processor SGI Onyx with 256 MB main memory, Infinite Reality Engine graphics, and two RM-64 texture systems. Images courtesy of David Heath, Johns Hopkins University.*



(a) Visible human head, 10 seconds/frame



(b) Cut planes and shadows, 10 seconds/frame

Figure 12: *Data from the visible human project rendered on VIRIM. a) Head, downsampled to* 256^3 *voxels. b) Same as a), but rendered with three cut-planes and shadows. Images courtesy of University of Mannheim, Germany, and Volume Graphics GmbH, Heidelberg, Germany.*

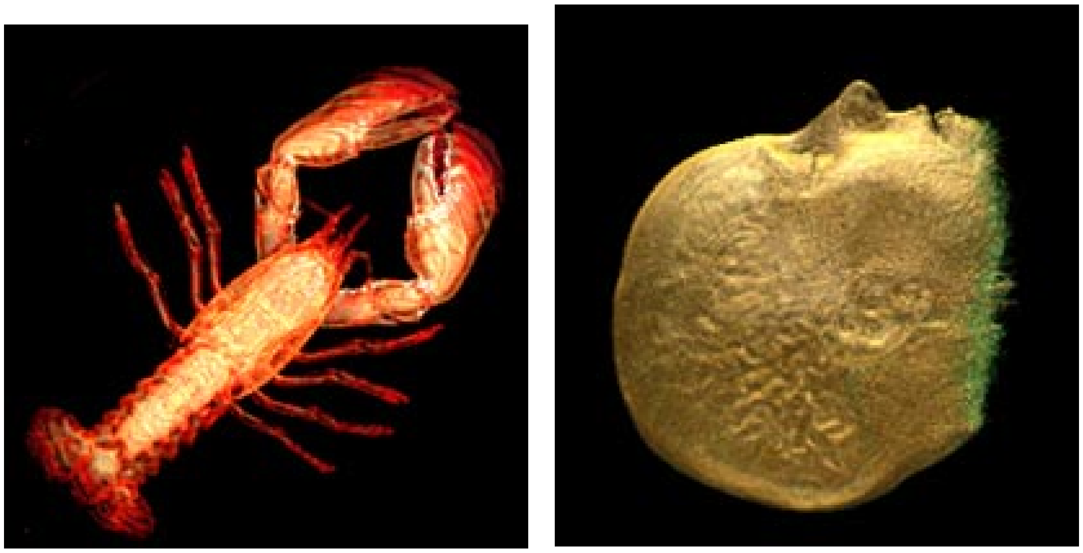

(a) Lobster (b) MRI head

Figure 13: *Images from a bit-accurate C++ software simulation of the VolumePro system. a) CT scan of a lobster (*$320 \times 320 \times 34$*). b) UNC MRI head (*$256 \times 256 \times 109$*).*

References

[Ake93] K. Akeley. RealityEngine graphics. In *Computer Graphics*, Proceedings of SIGGRAPH 93, pages 109–116, August 1993.

[BK97] I. Bitter and A. Kaufman. A ray-slice-sweep volume rendering engine. In *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, pages 121–130, Los Angeles, CA, August 1997.

[CCF94] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *1994 Workshop on Volume Visualization*, pages 91–98, Washington, DC, October 1994.

[Fly95] M. J. Flynn. *Computer Architecture – Pipelined and Parallel Processor Design*. Jones and Bartlett Publishers, 1995.

[FvDFH90] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, 2nd edition, 1990.

[Gib97] S. Gibson. Linked volumetric objects for physics-based modeling. Technical Report TR97-20, MERL – A Mitsubishi Electric Research Lab, November 1997.

[GPR$^+$94] T. Guenther, C. Poliwoda, C. Reinhard, J. Hesser, R. Maenner, H.-P. Meinzer, and H.-J. Baur. VIRIM: A massively parallel processor for real-time volume visualization in medicine. In *Proceedings of the 9th Eurographics Workshop on Graphics Hardware*, pages 103–108, Oslo, Norway, September 1994.

[HB86] K. H. Höhne and R. Bernstein. Shading 3D-images from CT using gray-level gradients. *IEEE Transactions on Medical Imaging*, MI-5(1):45–47, March 1986.

[HCN94] B. M. Hemminger, T. J. Cullip, and M. J. North. Interactive visualization of 3D medical image data. Technical report, University of North Carolina at Chapel Hill, Department of Radiology and Radiation Oncology, 1994. TR 94-027.

[HMK$^+$95] J. Hesser, R. Männer, G. Knittel, W. Straßer, H. Pfister, and A. Kaufman. Three architectures for volume rendering. In *Proceedings of Eurographics '95*, pages C–111–C–122, Maastricht, The Netherlands, September 1995. European Computer Graphics Association.

[Kat97] Y. Katayama. Trends in semiconductor memories. *IEEE Micro*, 7(6):10–17, November 1997. Special issue on Advanced Memory Technology.

[KB88] A. Kaufman and R. Bakalash. Memory and processing architecture for 3D voxel-based imagery. *IEEE Computer Graphics & Applications*, 8(6):10–23, November 1988.

[KS94] G. Knittel and W. Strasser. A compact volume rendering accelerator. In *Volume Visualization Symposium Proceedings*, pages 67–74, Washington, DC, October 1994. ACM Press.

[KS97] G. Knittel and W. Strasser. Vizard – visualization accelerator for realtime display. In *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, pages 139–146, Los Angeles, CA, August 1997.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.

[Lev90] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[LL94] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transform. In *Computer Graphics*, Proceedings of SIGGRAPH 94, pages 451–457, July 1994.

[MMSE91] H.-P. Meinzer, K. Meetz, D. Scheppelmann, and U. Engelmann. The Heidelberg ray tracing model. *IEEE Computer Graphics & Applications*, page 34, November 1991.

[OPL$^+$97] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami. EM-Cube: An architecture for low-cost real-time volume rendering. In *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, pages 131–138, Los Angeles, CA, August 1997.

[PAC97] M. Peercy, J. Airey, and B. Cabral. Efficient bump mapping hardware. In *Computer Graphics*, Proceedings of SIGGRAPH '97, pages 303–306, August 1997.

[PD84] T. Porter and T. Duff. Compositing digital images. *Computer Graphics*, 18(3), July 1984.

[PK96] H. Pfister and A. Kaufman. Cube-4 – A scalable architecture for real-time volume rendering. In *1996 ACM/IEEE Symposium on Volume Visualization*, pages 47–54, San Francisco, CA, October 1996.

[Sak97] K. Sakamura. Advanced DRAM technology. *IEEE Micro*, 7(6):8–9, November 1997. Special issue on Advanced Memory Technology.

[SS92] P. Schröder and G. Stoll. Data parallel volume rendering as line drawing. In *1992 Workshop on Volume Visualization*, pages 25–31, Boston, MA, October 1992.

[Wes91] L. A. Westover. *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*. PhD thesis, The University of North Carolina at Chapel Hill, Department of Computer Science, July 1991. Technical Report, TR91-029.

[YK92] R. Yagel and A. Kaufman. Template-based volume viewing. *Computer Graphics Forum, Proceedings Eurographics*, 11(3):153–167, September 1992.