MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Ray Casting Architectures for Volume Visualization

Harvey Ray, Hanspeter Pfister, Deborah Silver, Todd A. Cook

TR99-17 April 1999

Abstract

Real-time visualization of large volume datasets demands high performance computation, pushing the storage, processing, and data communication requirements to the limits of current technology. General purpose parallel processors have been used to visualize moderate size datasets at interactive frame rates; however, the cost and size of these supercomputers inhibits the widespread use for real-time visualization. This paper surveys several special purpose architectures that seek to render volumes at interactive rates. These specialized visualization accelerators have cost, performance, and size advantages over parallel processors. All architectures implement ray casting using parallel and pipelined hardware. We introduce a new metric that normalizes performance to compare these architectures. The architectures included in this survey are VOGUE, VIRIM, Array Based Ray Casting, EM-Cube, and VIZARD II. We also discuss future applications of special purpose accelerators.

IEEE Transactions on Visualization and Computer Graphics

MERL – A MITSUBISHI ELECTRIC RESEARCH LABORATORY
http://www.merl.com

Ray Casting Architectures for Volume Visualization

Harvey Ray[*], Hanspeter Pfister[†], Deborah Silver[‡], Todd A. Cook[§]

TR-99-17   April 1999

Abstract

Real-time visualization of large volume datasets demands high performance computation, pushing the storage, processing, and data communication requirements to the limits of current technology. General purpose parallel processors have been used to visualize moderate size datasets at interactive frame rates; however, the cost and size of these supercomputers inhibits the widespread use for real-time visualization. This paper surveys several special purpose architectures that seek to render volumes at interactive rates. These specialized visualization accelerators have cost, performance, and size advantages over parallel processors. All architectures implement ray casting using parallel and pipelined hardware. We introduce a new metric that normalizes performance to compare these architectures. The architectures included in this survey are VOGUE, VIRIM, Array Based Ray Casting, EM-Cube, and VIZARD II. We also discuss future applications of special purpose accelerators.

To appear in IEEE Transactions on Visualization and Computer Graphics

[*] Harvey Ray is a Ph.D. student at Rutgers State University, Email: haray@caip.rutgers.edu

[†] Hanspeter Pfister is with Mitsubishi Electric Research, Email: pfister@merl.com

[‡] Deborah Silver is an associate professor at Rutgers State University, Email: silver@caip.rutgers.edu

[§] Todd Cook is a research and development engineer at Improv System Inc., Email: toddc@improvsys.com

Publication History:–

1. First printing, TR-99-17, April 1999

1 Introduction

Volume visualization is an important tool to view and analyze large amounts of data from various scientific disciplines. It has numerous applications in areas such as biomedicine, geophysics, computational fluid dynamics, finite element models, and computational chemistry. Numerical simulations and sampling devices such as magnetic resonance imaging (MRI), computed tomography (CT), satellite imaging, and sonar are common sources of large 3D datasets. These datasets are generally anywhere from 128^3 to 1024^3 and may be non-symmetric (i.e., $1024 \times 1024 \times 512$).

Volume rendering involves the projection of a volume dataset onto a 2D image plane. From Figure 1 we see that a volume dataset is organized as a 3D array of volume elements, or voxels[1].

Voxels represent various physical characteristics, such as density, temperature, velocity, and pressure. Other measurements, such as area and volume, can be extracted from the volume datasets. Volume data may contain more than a hundred million voxel values requiring a large amount of storage. In Figure 1, the voxels are uniform in size and regularly spaced on a rectilinear grid. Other types of volume data can be classified into curvilinear grids, which can be thought of as resulting from a warping of a regular grid, and unstructured grids, which consist of arbitrary shaped cells. This paper presents a survey of recent custom volume rendering architectures that seek to achieve interactive volume rendering for rectilinear datasets. A survey of other methods used to achieve real time volume rendering is presented in [1]. The motivation for custom volume renderers is discussed in the next section. Several other custom architectures exist [2, 3, 4, 5, 6, 7, 8, 9] but were not presented because they are either related to the architectures presented here or are not considered to be recent. Section 3 presents three parallel volume rendering algorithms that are implemented by the architectures in this paper. Major components of a volume rendering system are discussed in Section 4. Five specialized volume rendering architectures are surveyed in Section 5. A new metric is introduced in Section 6 to compare each architecture. A comparison of the surveyed architectures is presented in Section 7 and a discussion is presented in section 8. Future trends for specialized rendering architectures are presented in Section 9.

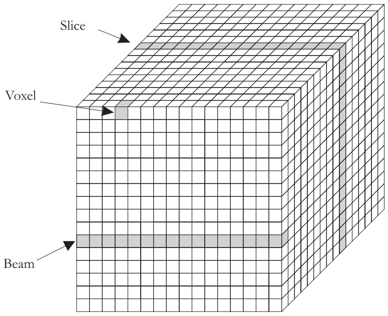


Figure 1: *Volume dataset.*

2 Need for Custom Visualization Architectures

A real-time volume rendering system is important for the following reasons [10]: 1) to visualize rapidly changing 4D (spatial-temporal) datasets, 2) for real-time exploration of 3D datasets (e.g., virtual reality), 3) for interactive manipulation of visualization parameters (e.g., classification), and 4) interactive volume graphics [11]. As the sampling rates of devices become faster, it will be possible to generate several 3D datasets at interactive rates; real-time volume rendering is required to visualize these dynamically changing

[1] Note, the term voxel has been used to refer to point samples and cubic volume elements. The papers surveyed here use both definitions for illustration purposes. Therefore, figures in this paper will use a point sample representation or a unit volume representation of a voxel as necessary.

datasets (e.g., for 3D ultrasound [12, 13]). It is often necessary to view the dataset from continuously changing positions to better understand the data being visualized; real-time volume rendering will enhance visual depth cues through motion and occlusion as the dataset is viewed from varying positions. Classification is important in correctly visualizing the dataset by configuring object properties (opacity, color, etc.) based on voxel values; classification is an iterative process which will benefit from real-time volume rendering; thus, scientists will be able to interactively manipulate opacity and color mappings. Volume graphics is an emerging area of research that produces synthetic datasets [11]. Volume graphics challenges the way 3D graphics is currently implemented. Traditional 3D graphics use polygonal meshes to model objects and these meshes are scan-converted into pixels inside the frame buffer. Alternatively, volume graphics models objects as a 3D discrete set of point samples (voxels). These voxels comprise the 3D dataset. The dataset is rendered using standard volume visualization techniques.

Real-time visualization of large 3D datasets places stringent computational demands on modern workstations, especially on the memory system. Table 1 estimates the memory bandwidth to render different size datasets at $30Hz$. It is assumed that the volume rendering algorithm accesses each voxel once per projection. The required memory bandwidth can not be sustained on most modern workstations and personal computers. The dataset must be partitioned among multiple memory modules to achieve the desired bandwidth and parallel processing must be used.

Table 1: Estimated memory bandwidth for real-time volume rendering.

Dataset Size	Frame Rate (Hz)	Memory Bandwidth
$128^3 \times 16$	30	120 MB/s
$256^3 \times 16$	30	960 MB/s
$512^3 \times 16$	30	7.5 GB/s
$1024^3 \times 16$	30	60 GB/s

Massively parallel processors and multiprocessors architectures [14, 15, 16, 17, 18, 19] have achieved image generation rates up to $30Hz$ on moderate sized datasets using algorithmic optimizations; however, the cost of these machines is prohibitive. In addition, the algorithmic optimizations are usually dataset dependent. Custom architectures have the potential to match or exceed the performance of other interactive visualization solutions at a lower cost and smaller size. Performance, cost, and size benefits are necessary for a desktop interactive visualization system.

3 Parallel Ray Casting

Volume rendering involves the direct projection of the entire 3D dataset onto a 2D display. Volume rendering algorithms can simultaneously reveal multiple surfaces, amorphous structures, and other internal structures of a 3D dataset [20]. These algorithms can be divided into two categories: forward-projection and backward-projection. Forward-projection algorithms iterate over the dataset during the rendering process projecting voxels onto the image plane. A common forward-projection algorithm is splatting [21]. Backward-projection algorithms iterate over the image plane during the rendering process re-sampling the dataset at evenly spaced intervals along each viewing ray. In general, ray casting algorithms traverse the dataset in a more random manner. All architectures surveyed in this paper implement ray casting, a common backward-projection algorithm [22]. The ray casting algorithm is capable of producing high-quality images and a large degree of parallelism can be exploited from the algorithm.

In ray casting, rays are cast into the dataset. Each ray originates at the viewing (eye) position, penetrates a pixel in the image plane (screen), and passes through the dataset. At evenly spaced intervals along the ray, sample values are computed using interpolation. The sample values are mapped to display properties such as opacity and color. A local gradient is combined with a local illumination model at each sample point to provide realistic shading of the object. Final pixel values are found by compositing color and opacity values along a ray. Composition models the physical reflection and absorption of light.

Because of the high computational requirements of volume rendering, the data needs to be processed in a pipelined and parallel manner. Parallel ray casting algorithms use one of the following processing strategies: object order, image order, or hybrid order [23]. This division describes the manner in which a dataset is processed. Figure 2 illustrates the three variations.

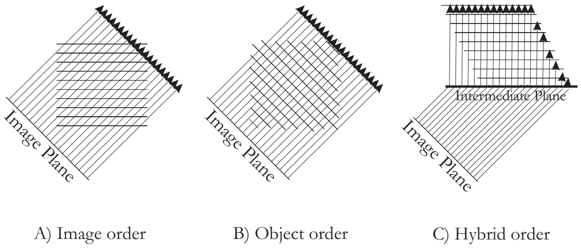


Figure 2: *Ray casting categories.*

A dataset in Figure 2 is organized as a set of parallel slices. Image order algorithms cast rays through the image plane and re-sample at locations along the ray (Figure 2A). They offer flexibility for algorithmic optimizations, but accessing the volume memory in a non-predictable manner significantly slows down memory performance. Object order algorithms require that the dataset be re-sampled so that the slices are aligned with the view direction (Figure 2B). A major advantage of object order algorithms is that accesses to the volume memory are predictable, thereby, leading to efficient memory bandwidth utilization. Hybrid order algorithms project the dataset to the face of the dataset most parallel to the image plane. This also allows for predictable memory accesses to the volume data where no more than one sample is taken per voxel. The intermediate 2D image is warped into the final image (Figure 2C). The shear-warp algorithm is an example of a hybrid order algorithm [24]. A summary of implementation tradeoffs for each parallel scheme is shown in Table 2.

Table 2: Tradeoffs for different parallelization methods.

	Image order	Object order	Hybrid order
Advantages	- Easy to implement algorithmic optimizations (e.g., early-ray termination)	- Regular memory access patterns	- Merge benefits of image order and object order algorithms
Disadvantages	- ”Random” memory access patterns - Non-uniform mapping of ray samples to voxels	- Difficult to implement algorithmic optimizations	- Perspective projections adversely affect performance - Additional 2D image warp required

4 Components of a Ray Casting System

The following components are needed for any ray casting implementation:

Memory system provides the necessary voxel values at a rate which ultimately dictates the performance of the architecture.

Ray-path calculation determines the voxels that are penetrated by a given ray; it is tightly coupled with the organization of the memory system.

Interpolation estimates the value at a re-sample location using a small neighborhood of voxel values.

Gradient estimation estimates a surface normal using a neighborhood of voxels surrounding the re-sample location.

Classification maps interpolated sample values and the estimated surface normal to a color and opacity.

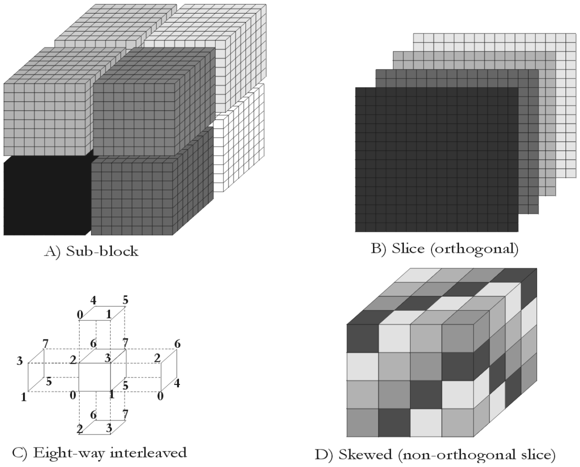
Shading uses gradient and classification information to compute a color that takes into account the interaction of light on the estimated surfaces in the dataset.

Composition uses shaded color values and opacity to compute a final pixel color for display.

4.1 Memory System

The memory system is the most important component of a visualization architecture. The memory system contains the dataset and is responsible for supplying the computational units with voxel values at a high bandwidth to support the target frame rate. Since the dataset will be visualized from various view positions, the throughput of the memory system should be as view independent as possible. Regardless of the parallel processing strategy, each ray casting algorithm requires simultaneous access to multiple voxels. Ideally, the memory system provides these voxels in a conflict-free manner; otherwise, the overall system may suffer performance degradation.

The architectures surveyed in this paper use four memory partitioning schemes shown in Figure 3 to achieve a high memory throughput. Sub-block partitioning (Figure 3A) divides the dataset into smaller volumes. Each sub-block is assigned to a different memory module. Orthogonal slice partitioning (Figure 3B) assigns each slice inside the dataset to a memory module. Each slice is perpendicular to one axis of the dataset. In this partitioning scheme, memory throughput is maximized for two of the three orthogonal viewing directions. The eight-way interleaved memory system (Figure 3C) assigns each voxel in a $2 \times 2 \times 2$ block to separate memory banks. The eight-way interleaved memory partition is limited to eight parallel memory accesses. As a result, it can be combined with sub-block partitioning when additional parallelism is necessary. The skewed (non-orthogonal) slice partitioning scheme (Figure 3D) assigns slices that make a $45°$ angle with each axis of the dataset to memory modules. In this partitioning scheme, the memory throughput is maximized for the three orthogonal viewing directions.


A) Sub-block

B) Slice (orthogonal)

C) Eight-way interleaved

D) Skewed (non-orthogonal slice)

Figure 3: *Common memory organization schemes.*

The maximum performance obtained by a volume rendering architecture is primarily determined by the degree of parallelism and the memory technology used. Recent memory devices use pipelining to accelerate linear accesses. Synchronous memories, such as Synchronous DRAM (SDRAM), can sustain memory accesses at $150MHz$. This is a three-fold speed up over previous alternatives. More recently, Rambus defined a high-speed interface that will allow sustainable bandwidths up to 800 MB/s using an 8-bit bus. Using a wider 16-bit bus (Direct Rambus), these devices are able to sustain 1.6 GB/s data throughput [25]. These advanced memories can potentially enhance performance for any given architecture. A metric that measures the ability of a volume rendering architecture to utilize available memory bandwidth is presented in Section 6.

4.2 Ray-Path Calculation

Calculating ray-voxel intersections is tightly coupled with the memory system design and is related to the type of ray casting algorithm used. The appropriate memory addresses for each voxel that a ray penetrates must be computed. These addresses are calculated by constructing a line (ray) between the viewing position and a pixel on the image plane and extending the line (ray) through the dataset. Based on the processing strategies from the previous section, it may be necessary to calculate a substantial number of memory addresses in parallel. Look-up tables (or templates) have been used to reduce the computation involved in calculating ray paths through the dataset [26]. For parallel projections and hybrid order architectures, templates only need to be generated once per projection because all rays have the same slope.

4.3 Interpolation

Estimating the sample value requires evaluation of the trilinear interpolation equation:

$$\begin{aligned} S(i,j,k) &= P_{000}(1-i)(1-j)(1-k) \\ &+ P_{100}i(1-j)(1-k) + P_{010}(1-i)j(1-k) \\ &+ P_{110}ij(1-k) + P_{001}(1-i)(1-j)k \\ &+ P_{101}i(1-j)k + P_{011}(1-i)jk + P_{111}ijk \end{aligned} \tag{1}$$

i, j, and k are fractional offsets of the sample position in the x, y, and z directions, respectively. These variables are between 0 and 1. P_{abc} is a voxel whose relative position in a $2 \times 2 \times 2$ neighborhood of voxels is (a,b,c). a, b, and c are the least significant bit of the x, y, and z sample position, respectively. From Equation 1, we see that a total of 24 multiplications are necessary and eight voxel values are required to compute each re-sample location. The number of multiplications can be reduced by approximately one-half if factors are re-used. If we assume that each $1 \times 1 \times 1$ unit volume in a $512 \times 512 \times 512$ dataset contains one re-sample location per projection, then more than 1.5 billion multiplications would be necessary for each projection. Multiple projections are needed per second for interactive projection rates, requiring an enormous amount of computational power. As few as eight multiplications are necessary if the interpolation weights are stored in a look-up table. Higher-order interpolation can be used to improve image-quality but it is typically not done in hardware because of its computational cost.

4.4 Gradient Estimation

The next step is the determination of gradients to approximate surface normals for classification and shading. x-, y-, and z-gradients may be computed using central differences:

$$\begin{aligned} G_x &= \frac{S(i+1,j,k)-S(i-1,j,k)}{\Delta x} \\ G_y &= \frac{S(i,j+1,k)-S(i,j-1,k)}{\Delta y} \\ G_z &= \frac{S(i,j,k+1)-S(i,j,k-1)}{\Delta z} \end{aligned} \tag{2}$$

$S(i,j,k)$ is the interpolated sample at the location (i,j,k) inside the dataset. Δx, Δy, and Δz is the spacing between samples in x, y, and z directions, respectively. The costly divisions are usually avoided because of the regular spacing between voxels inside the dataset. Two re-sample locations adjacent to the sample location in each direction are required to compute the gradient using central differences. Some algorithms use a larger neighborhood of voxels to generate images that appear smoother and/or to reduce temporal aliasing. In addition to the gradient vector components, the gradient magnitude and the normalized gradient vector may be required. Gradients can also be taken at neighboring voxels and interpolated to yield the gradient at the re-sample location.

In practice, several gradient estimation schemes exist [27, 20, 28, 29, 30]. A comprehensive consideration of these methods is beyond the scope of this survey. In general, high-quality gradient estimation requires additional computation and memory bandwidth (or on-chip storage) that may affect performance and cost.

4.5 Classification

Classification maps a color and opacity to sample values. Opacity values range from 0 (transparent) to 1.0 (opaque) [22]. Classification is typically implemented in hardware using look-up tables (LUTs). These LUTs are typically addressed by sample value and/or gradient magnitude, and they output sample opacity and color. It is desirable to be able to modify the information in these LUTs during the visualization process in real-time. If the architecture processes multiple re-sample locations in parallel, these LUTs must be duplicated to avoid contention.

4.6 Shading

The Phong shading algorithm [31], or variants, are often used in the shading subsystems of volume rendering architectures. This algorithm requires gradients, light and reflection vectors to calculate the shaded color for each re-sample location. The algorithm involves computationally expensive division, multiplication, and exponentiation that must be implemented in hardware. In practice, the shading algorithm is implemented in either arithmetic units for accuracy or reflectance LUTs for flexibility [32]. For color images, the Phong shading models may be applied to the red, green, and blue components. Also, additional computation may be necessary if multiple light sources are supported.

4.7 Compositing

The composition system is responsible for summing up color and opacity contributions from re-sample locations along a ray into a final pixel color for display [33]. The front-to-back formulation for compositing is:

$$\begin{aligned} C_{\mathrm{Acc}} &= (1.0 - \alpha_{\mathrm{Acc}}) * C_{\mathrm{sample}} + C_{\mathrm{Acc}} \\ \alpha_{\mathrm{Acc}} &= (1.0 - \alpha_{\mathrm{Acc}}) * \alpha_{\mathrm{sample}} + \alpha_{\mathrm{Acc}} \end{aligned} \tag{3}$$

C_{Acc} is the accumulated color, α_{Acc} is the accumulated opacity, C_{sample} is the samples color, and α_{sample} is the samples opacity. Two multiplies are needed to composite each re-sample location. Compositing in a front-to-back order allows for early ray termination if a desired opacity threshold has been reached. Back-to-front composition can be utilized to simplify the calculation; however, early ray termination is not possible. Color information produced from the compositing system is stored into a frame buffer for display.

5 Architecture Survey

This section presents five special purpose volume rendering architectures. A description of each architecture is given along with its performance. The following architectures are surveyed, in chronological order of their development: VOGUE, VIRIM, Array Based Ray Casting, EM-Cube, and VIZARD II. Each figure in this section were redrawn from their original publication.

5.1 VOGUE

The VOGUE architecture [34, 35] was developed at the University of Tübingen, Germany. One rendering engine provides high-quality, volume-rendered images with multiple light sources using four custom VLSI chips. A block diagram of the architecture is shown in Figure 4. The main goals of VOGUE are flexibility and compactness. VOGUE is capable of three rendering modes based on the gradient estimation method: a fast 8-voxel gradient, a slower intermediate quality 32-voxel gradient, and a higher quality 56-voxel gradient. VOGUE hardware consist of an Address SeQuencer (ASQ) for memory addressing, a volume memory (VoluMem) for dataset storage, a Reconstructor EXtractor (REX) for interpolation, a COLOSSUS unit for shading, and a COMET unit for composition. VOGUE implements an unrestricted Phong illumination model in addition to depth cueing.

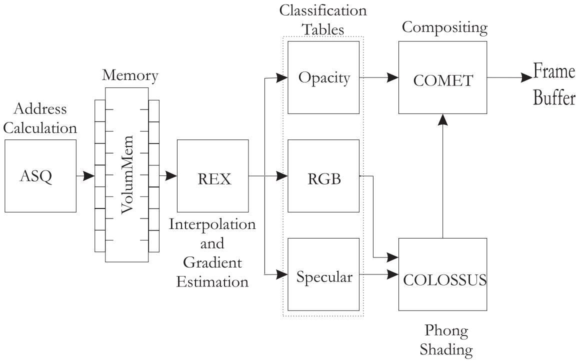


Figure 4: *VOGUE architecture.*

5.1.1 Description

The Volume Memory (VoluMem) is organized as an eight-way interleaved memory system (see Figure 3C) which allows eight voxels surrounding a trilinear re-sample location to be retrieved in parallel.

The ASQ unit provides necessary addresses for the VoluMem. It generates addresses for the voxels involved in re-sampling and gradient estimation. A ray's initial position and incremental values to the next re-sample location are computed by the host computer and passed to the ASQ where they are incremented to compute the address of the eight voxels surrounding the re-sample location.

The REX unit performs trilinear interpolation using the eight voxels from VoluMem to compute the re-sampled value. The REX contains three stages of linear interpolators. Adjacent voxels from the trilinear interpolation neighborhood are used in linear interpolations to compute edge-values, then pairs of edge-values are used in linear interpolations to compute face-values, and the last linear interpolation uses a pair of opposite face-values to compute the final sample value. The REX is a pipelined unit and produces one interpolation value per clock cycle.

In addition to interpolation, the REX unit also performs gradient calculation. Gradient calculation requires 1 memory accesses for the fastest gradient mode (8-voxel gradient), 4 memory accesses for the intermediate mode (32-voxel gradient), and 7 memory accesses for the highest quality gradient mode (56-voxel gradient). In the fastest gradient mode, opposite face-values computed during trilinear interpolation are used to compute gradients. The higher quality gradient modes require additional voxels and interpolation. The REX unit can produce one gradient vector and magnitude per clock cycle. The REX unit contains three pipelined square units and one square root unit to compute the gradient magnitude.

Classification information is stored in three LUTs: specular coefficient, color, and opacity. These tables are indexed using the sample value, gradient, and gradient magnitude. These values are subsequently used by the shading unit (COLOSSUS) and compositing unit (COMET).

The COLOSSUS shading unit implements the unrestricted Phong illumination model and depth cueing. The specular coefficient from the LUT along with the gradient vector, light vector, and ambient coefficient are passed to the COLOSSUS chip. The COLOSSUS chip internally converts operands to logarithms to reduce multiplication and division to simple addition and subtraction, respectively. The costly exponentiation operation required by the Phong illumination model is reduced to a multiply; however, fast logarithmic converters are necessary. These units are pipelined to achieve the desired system performance.

Shaded samples are composited in the COMET chip. The COMET chip requires an opacity, from a LUT, and color values from the COLOSSUS chip. These values are composited into a final pixel color that is passed to the frame buffer.

5.1.2 Performance

Estimated performance of one VOGUE module, containing the four VLSI units (ASQ, REX, COLOSSUS, and COMET), is 2.5 frames/second for 256^3 datasets using the fastest rendering mode. For higher performance, several rendering modules are connected to other modules in a ring network. To achieve larger memory throughput, a fully-parallel implementation uses sub-block partitioning to globally partition the dataset. Each sub-block is locally partitioned using the eight-way memory interleaving scheme and is stored into the VoluMem of a given rendering module. Boundary voxels are replicated among adjacent rendering modules to enhance performance.

VOGUE is capable of perspective projection and is able to utilize early ray termination. The estimated performance using the fastest 1-access gradient mode is $20Hz$ using eight modules for 256^3 datasets and using 64 modules for 512^3 datasets. VOGUE's highest quality gradient mode improves image-quality, however, performance is lowered to 2 frames/second.

5.2 VIRIM

The VIRIM architecture has been developed and assembled at the University of Mannheim [36] to achieve real-time visualization on moderate sized datasets $(256 \times 256 \times 128)$ with high image quality. VIRIM is an object order ray casting engine that uses the Heidelberg raytracing algorithm [37] discussed below. The VIRIM architecture is shown in Figure 5. It consist of a geometry unit and a ray casting unit. The geometry unit is responsible for interpolation and gradient calculation; the ray casting unit is responsible for implementing the actual ray casting algorithm.

5.2.1 Description

The rotation of the dataset occurs on dedicated rotation hardware called the Rotator Board (geometry unit in Figure 5). The Rotator Board aligns the dataset with the viewing position. The Rotator Board consists of the volume memory, a geometry processor, an interpolation processor, and a gradient processor.

The dataset is stored in an eight-way interleaved memory system. The dataset is rotated using backward mapping from the re-sample position and a weighted interpolation mask on an eight voxel neighborhood. Arbitrary (e.g., Gaussian) interpolation weights can be used in the 8-voxel neighborhood instead of trilinear interpolation. The geometry processor generates the addresses for the eight memory banks using a rotation matrix. Unlike other architectures, VIRIM does an interpolation on classified density values. The mappings are stored in eight LUTs that can be freely modified.

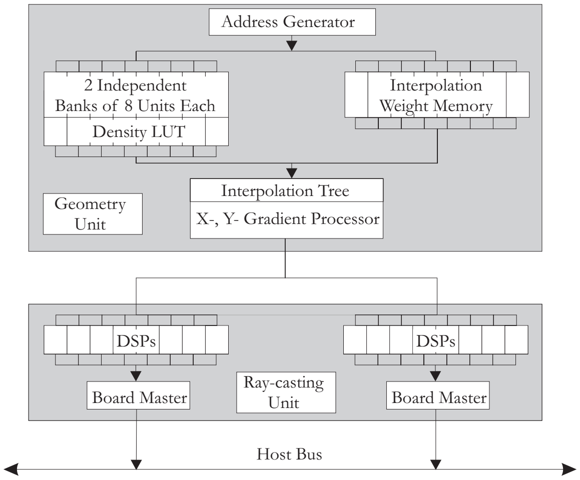


Figure 5: *VIRIM architecture.*

A modified 2D Sobel filter is used to estimate the X and Y components of the gradient vector in the re-sampled coordinate system. Because of this, the gradient is only two-dimensional and view dependent. The output of the rotator board are the density and gradient values for a sample location. These components are transferred to the Digital Signal Processor (DSP) boards (ray casting unit in Figure 5) using a specialized bus and stored into first-in first-out memories (FIFOs). The geometry units are much faster than the DSPs; therefore, the FIFOs are required to de-couple speed differences between the two units.

The DSP board implements ray casting with the Heidelberg illumination model. In the Heidelberg model, the dataset is rotated such that viewer looks along a major axis. Two light sources enter the volume. One light source is along the direction of the viewer and the other light source is 45° from the first light source. Light intensity is calculated slice-by-slice, and the final illumination value per sample is generated by the summation of all light intensity emitted in the viewers direction. The Heidelberg raytracing algorithm can account for reflection, absorption, emission of light, and is capable of producing shadows.

The DSP board contains eight DSP chips and a CPU. Floating point operations for the shading and visualization algorithm are performed by the DSPs. The DSPs are programmable and provide flexibility for the architecture to implement different volume rendering and shading algorithms.

5.2.2 Performance

VIRIM is capable of producing shadows and supports perspective projections. One VIRIM module with four boards has been assembled and achieves $2.5Hz$ frame rates for $256 \times 256 \times 128$ datasets. To achieve interactive frame rates, multiple rendering modules have to be used; however, dataset duplication is required. Four modules (16 boards) are estimated to achieve $10Hz$ for the same dataset size, and eight modules (32 boards) are estimated to achieve $10Hz$ for 256^3 datasets [23].

5.3 Array Based Ray Casting

The Array Based Ray Casting engine developed at the University of New South Wales [38] is an object order ray casting architecture. This architecture consists of two parallel pipelined arrays used to rotate the dataset and to cast rays, as illustrated in Figure 6. These rotation arrays are connected between n memory modules and $1.5n$ rendering pipelines, n is the resolution of the dataset. In the second array, intersections with voxels are determined by using nearest neighbor or zero order interpolation. Each rendering pipeline performs

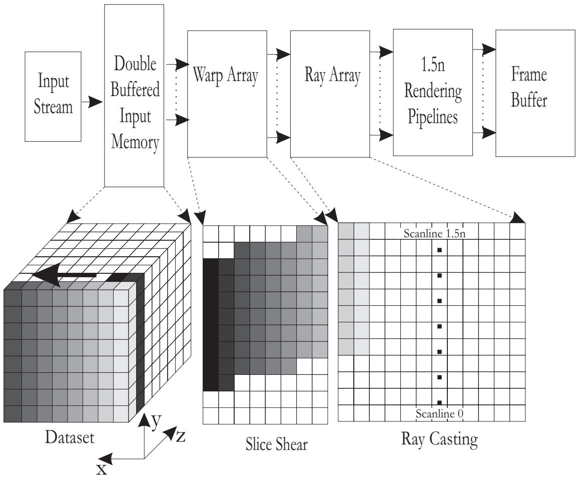


Figure 6: *Array Based Ray Casting architecture.*

shading and composition for a given scanline. In addition, the system is composed of a double-buffered input memory, memory swapping array, and a frame buffer.

5.3.1 Description

The volume dataset is stored in a double-buffered volume memory that allows the simultaneous loading of one dataset and visualization of another. The memory system uses orthogonal slice partitioning (see Figure 3B). The dataset is stored in memory in a view dependent manner using coordinate swapping. Using a spherical coordinate system, view positions are classified as being in one of eight primary octant regions. As the dataset is loaded, coordinate swapping occurs based on the view position to allow conflict-free access to beams. Note that coordinate swapping performs a partial rotation. Limited rotations about the X- and Y- axis occurs in the Warp Array and Ray Array, respectively. These three partial rotations allow general rotation of the dataset.

Vertical beams, indicated by similar shaded voxels in Figure 6, are loaded into the Warp Array in one clock cycle. The Warp Array rotates the volume by $\pm 45°$ around the X-axis by shearing slices in Y. The shear is accomplished in the Warp Array by shifting beams of voxels based on a comparison of the row coordinate and the beam's rotated Y-coordinate. The first column of the Warp Array computes these Y-coordinates for each voxel, and the remaining columns contain simple processing elements. These elements perform three basic functions: shift-right, shift-right-up, and shift-right-down. The rows in both the Warp Array and Ray Array correspond to a discrete Y-coordinate. However, explicit Y-coordinate information is only stored in the Warp Array.

Voxels in the rightmost column of the Warp Array proceed to adjacent processing elements in the leftmost column of the Ray Array. The Ray Array casts parallel rays into the sheared YZ-slices. As indicated in Figure 6, a row inside the Ray Array corresponds to a scanline of rays. Initializers compute X- and Z-coordinates for voxels during ray casting. Each ray's initial coordinate and increment vector is shifted into place inside the Ray Array before ray casting. The processing elements in the Ray Array implement a Compare-and-Shift-Right function. If a voxels coordinate matches a ray's current coordinate, a flag is set which proceeds through the remainder of the array with the voxel and coordinate data. The Ray Array implements nearest neighbor or zero order interpolation.

A one dimensional array of rendering pipelines classifies, shades, and composites the voxels along the discrete rays. To estimate gradients, each element in the Ray Arrays has additional registers to buffer voxel information. Voxels traverse a row inside the Ray Array three times. A 3×3 box filter is used in the

Rendering Pipelines to estimated gradients. The gradient estimation and shading algorithm uses a full 26-voxel neighborhood and creates smoothly shaded images [39].

5.3.2 Performance

If n is the dimension of the volume data, the size of the Warp Array and Ray Array are $1.5n \times n$ and $1.5n \times 1.5n$, respectively. For 256^3 datasets, this corresponds to approximately 212,992 processing elements. An additional column in each array contains the coordinate initializers. The architecture also contains $1.5n$ rendering pipelines. The Warp Array for this dataset dimension is estimated to fit inside a 5×5 array of FPGAs. Processing elements in the Ray Array are larger than those in the Warp Array and require more hardware. However, a smaller Ray Array (i.e., with fewer columns) can be used by time-multiplexing the Ray Array and stalling the Warp Array, thereby reducing throughput.

This architecture only supports parallel rendering and is capable of $15Hz$ frame rates for 256^3 datasets sharing the Ray Array 10 times. The architecture has undergone several changes since its publication and is now called VIZAR [40].

5.4 EM-Cube

EM-Cube is a commercial version of the high-performance Cube-4 [8, 10, 41] volume rendering architecture that was originally developed at the State University of New York at Stony Brook. EM-Cube is currently under development at Mitsubishi Electric Research Laboratory [42]. The Cube family of architectures are characterized by memory skewing. EM-Cube is a highly parallel architecture based on the hybrid order ray casting algorithm shown in Figure 2C. Rays are sent into the dataset from each pixel on the base plane, which is co-planar to the face of the dataset that is most parallel to the image plane. Because the image plane is typically at some angle to the base-plane, the resulting base-plane image is 2D 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 [10]. Within a slice, voxels are read from memory a beam of voxels at a time, in top to bottom order. This leads to regular, object order data access. The EM-Cube architecture utilizes memory skewing [43] on a block granularity for conflict-free beam access.

5.4.1 Description

EM-Cube will be implemented as a PCI card for Windows NT computers. The card will contain one volume rendering ASIC, 32 Mbytes of volume memory, and 16 Mbytes of local pixel storage. The warping and display of the final image will be done on an off-the-shelf 3D graphics card with 2D texture mapping. The EM-Cube volume rendering ASIC, shown in Figure 7, contains eight identical rendering pipelines, arranged side by side, and interfaces to voxel memory, pixel memory, and the PCI bus. Each pipeline communicates

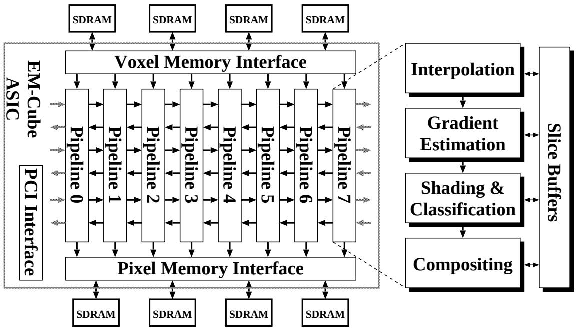


Figure 7: EM-Cube architecture with eight identical ray casting pipelines.

with voxel and pixel memory and the two neighboring pipelines. Pipelines on the far left and right are connected to each other in a wrap-around fashion (indicated by grey arrows in Figure 7). A main characteristic of EM-Cube 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.

EM-Cube is a parallel projection engine with multiple rendering pipelines. Each pipeline implements the ray casting algorithm. Samples along each ray are calculated using trilinear interpolation. A 3D gradient is computed using central differences between trilinear samples. The gradient is used in the shader stage, which computes the sample illumination according to the unrestricted Phong lighting model using a reflectance LUT [32]. Look-up 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 is organized as four 64-Mbit 16-bit wide SDRAMs for 32 Mbytes of volume storage. The volume data is stored as a $2 \times 2 \times 2$ blocks of neighboring voxels. Miniblocks are read and written in bursts of eight voxels using the fast burst mode of SDRAMs. In addition, EM-Cube uses linear skewing of these blocks. Skewing guarantees that the rendering pipelines always have access to four adjacent miniblocks in any of the three slice orientations.

5.4.2 Performance

EM-Cube is a parallel ray casting engine that implements a hybrid order algorithm; however, EM-Cube does not support perspective projections. Each of the four SDRAMs provides burst-mode access at up to $133MHz$, for a sustained memory bandwidth of $4 \times 133 \times 10^6 = 533$ million 16-bit voxels per second. Each rendering pipeline operates at $66MHz$ and can accept a new voxel from its SDRAM memory every cycle. Eight pipelines operating in parallel can process $8 \times 66 \times 10^6$ or approximately 533 million samples per second. This is sufficient to render 256^3 volumes at 30 frames per second.

5.5 VIZARD II

The VIZARD II architecture is being developed at the University of Tübingen to bring interactive ray casting into the realm of desktop computers [44]. This is the second generation of VIZARD systems [45, 46]. These image order architectures are characterized by methods to reduce memory bandwidth requirements for interactive visualization. While VIZARD uses a pre-shaded and pre-classified compressed dataset, VIZARD II only preprocesses gradients that are stored into a quantized gradient table. Using central differences as the underlying gradient filter, preprocessing the gradient filter requires only a few seconds and is only performed once per dataset. Gradient quantization potentially allows VIZARD II to implement global gradients. VIZARD II was designed to interface to a standard PC system using the PCI bus. The dataset is stored in four interleaved memory banks along with a pre-computed gradient index, segmentation index, and gradient magnitude for each voxel. The combination of pre-computed gradients, caching, and early ray termination reduces the bandwidth requirements of the memory system. Added flexibility is obtained by using a DSP and FPGAs as the implementation technology. This allows the VIZARD II card to perform other visualization task such as reconstruction, filtering, and segmentation.

5.5.1 Description

The VIZARD II architecture is illustrated in Figure 8. VIZARD II consists of four functional blocks: Control Unit, Memory Unit, Trilinear Interpolation Unit, and Shading/Compositing Unit. The Control Unit is determines intersections of the rays with the dataset and cut planes. The Memory Unit stores the dataset in four SDRAM modules, each with its own SRAM cache. The Trilinear Interpolation Unit is responsible for resampling the dataset. It also interpolates the gradients at off-grid locations using eight parallel lookups to the

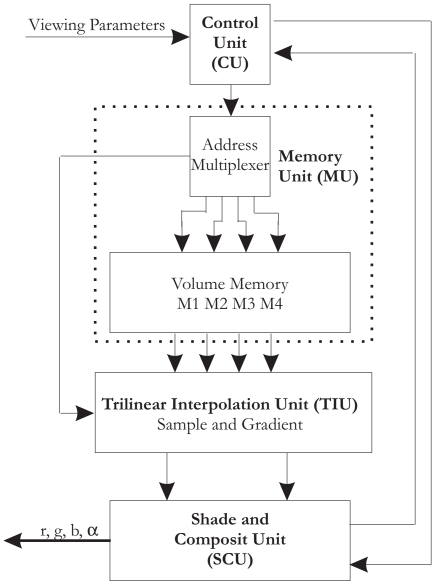


Figure 8: *VIZARD II architecture.*

quantized gradient table. The Shading and Compositing Unit supports look-up-based shading and multiple classification tables. Final pixels values are transferred through the PCI bus to the host computer.

VIZARD II implements an image order algorithm that utilizes early ray termination. The algorithm first preprocesses the dataset to compute gradient indices for each voxel. The gradient index contains 9 bits but is not limited to 9 bits. Alternatively, the full gradient component could be stored with the voxel memory. The 9-bit gradient index generates 512 table entries. Using 512 entries, the average error in the gradient computation is 2.3 degrees and the maximum error is 7.9 degrees [44]. Larger gradient tables can be used for greater accuracy. Four voxels (including gradient index) are simultaneously accessed in parallel. These voxels are four-way interleaved with respect to the YZ-plane. A burst memory access is used to fetch adjacent voxels in the X-direction. To access a $2 \times 2 \times 2$ trilinear neighborhood, four voxels (in the YZ-plane) are accessed in parallel from each bank, and a sequential burst memory access from each bank provides the remaining four values from an adjacent YZ-plane. These values are cached in separate cache banks to allow parallel access and re-use.

The Interpolation Unit uses the fractional $x, y,$ and z components and the trilinear interpolation neighborhood from the Memory Unit to re-sample the dataset. The gradient index is used to address the gradient look-up table. The resulting $x, y,$ and z gradient components are interpolated in a similar manner. The trilinear interpolation units can sustain samples at a rate up to four times faster than the rate of the memory system if the desired voxels reside inside the cache. The sample throughput is enhanced by supersampling the dataset because of a significant increase in cache hits.

The sample and gradient value are used in the Shading and Compositing Unit to classify and shade the sample. The classification table is chosen using a classification index and the architecture handles segmentation and multiple cut-planes. Phong shading is implemented using a look-up table and compositing uses the standard ”over” operator. Early ray termination is utilized to increase the frame rate.

5.5.2 Performance

VIZARD II supports multiple cut planes, segmentation, parallel, and perspective viewing. It is expected to sustain a frame rate of $10Hz$. However, its worst-case performance is approximately 1 frame per second.

Worst-case performance occurs for 1:1 mapping of samples to voxels and a transparent classification of the dataset. Using four $100MHz$ SDRAM devices, this architecture is capable of 14×10^6 samples per second worst-case performance and 56×10^6 samples per second best-case, assuming 4:1 mapping of samples to voxels.

6 Performance Metrics

The performance of a volume rendering architecture is determined by several factors:

Frame Rate is the number of images that can be generated per unit of time and is measured in frames per second (or Hz).

Samples Processed Per Second (SPPS) is the number of filtered samples that can be generated per unit of time. Unlike frame rate, SPPS is not sensitive to image and dataset resolution. SPPS is similar to trilinear interpolated samples per second that is commonly used in the specification of 3D texture mapping hardware.

Latency is the time between a change in dataset or viewing parameters and the display of the updated image.

Image quality is mainly a qualitative assessment that is related to the resolution of the generated images, interpolation filter, gradient filter, and illumination models used.

Scalability is the ability of an architecture to extend its performance by increasing the amount of computational and memory throughput. Ideally, a linear increase in the number of rendering modules should linearly increase an architecture's frame rate or maintain its frame rate for a linear increase in dataset size.

Although many architecture's primary goal is to achieve high frame rates and SPPS, image quality may be more desirable when visualizing static datasets. Frame rate and SPPS are indicators of the amount of acceleration that is provided by the rendering architecture. One drawback to these metrics is that both are proportional to the amount of bandwidth available to the architecture. This is undesirable since the architectures surveyed span several generations of memory technology. To address this problem, we introduce a simple model that measures the ability of an architecture to convert voxel throughput (memory bandwidth) into sample throughput. The model can be derived by accounting for changes in throughput along the *processing path* as a voxel is converted into a filtered sample. This leads to a relationship between peak memory bandwidth and effective sample throughput. Sample throughput can be given by the following equation:

$$S_{\frac{sample}{second}} = B_{\frac{voxel}{second}} \cdot \overbrace{U_{\%} \cdot \frac{1}{V_{\frac{voxel}{sample}}}}^{P} \tag{4}$$

where S is performance measured in SPPS, B is the peak bandwidth of the memory system, U is the memory bandwidth utilization (in percent), and V is the average number of voxels that need to be fetched per sample. In this survey, B is held constant to compensate for advances in memory technology and for varying degrees of parallelism among the different architectures.

U, memory bandwidth utilization, is the percentage of the peak memory bandwidth that is realized. The product of U and B is the sustained voxel throughput into the rendering components. For maximum performance, U should be 1.0. U accounts for any combination of random cycles, sequential cycles, and idle time on the memory bus. U is given by:

$$U = \frac{w}{r \cdot C + (1-r)} \tag{5}$$

where r is the percentage of all accesses that are random and w is the percentage of time that voxels are transferred to rendering components (bus utilization). C is the speedup of a memory device obtained by using sequential memory access (or burst access) instead of random access. C is a memory technology dependent term. For example, assume a $100MHz$ SDRAM memory ($10ns$ synchronous access time) has a $70ns$ random access time (including page faults). This leads to a C of 7.0 (i.e., 700%). In this case, if all memory accesses are random ($r = 1.0$) and the memory bus is fully utilized ($w = 1.0$), the memory bandwidth utilization, U, would be as low as 14.3%. To account for the fact that not all random accesses are page faults we assume that the memory bandwidth utilization is 20% when the bus is fully utilized ($w = 1.0$) with random accesses ($r = 1.0$). This is a conservative estimate for newer DRAM memories.

V is the average number of voxels that are fetched from memory per sample. V is related to the size of the gradient and interpolation filter. For example, an architecture that uses a 32-voxel gradient filter could have $V = 32$ using a brute force approach; however, if voxels and intermediate results are buffered efficiently, $V = 1$ because access to buffered (or cached) voxels can occur in parallel with new voxel accesses. Some architectures use algorithmic acceleration (e.g., early ray termination) to enhance performance. The reduction in fetched voxels due to algorithmic speedup is view and dataset dependent. In our comparison, we assume a reduction of V by a factor of 3 when early ray termination has been implemented [22].

We call the term, $P = \frac{U}{V}$ in Equation 4, *Sample Processing Efficiency*. It is an architecture specific measure of how peak memory bandwidth (B) translates to SPPS (S). If each voxel is accessed on average once per filtered sample ($V = 1.0$) and the memory system is fully utilized ($U = 1.0$), then the sample processing efficiency, P, will be 1.0. Sample processing efficiency may be larger than 1.0 if compression is used or if the volume dataset is super-sampled (i.e., multiple re-sample locations per unit volume). Sample processing efficiency, P, is related to frame rate, F, by the following formula:

$$F_{\frac{frame}{second}} = \frac{B_{\frac{voxel}{second}} \cdot P_{\frac{sample}{voxel}}}{T_{\frac{sample}{frame}}} \tag{6}$$

B is the peak bandwidth of the memory system and T is the number of samples needed to render the frame. Sample processing efficiency only measures the ability to convert memory bandwidth to processed samples; it is relatively independent of memory technology and can therefore be used as an objective measure of architecture efficiency. However, it does not measure other important performance metrics such as image-quality, cost, scalability, or latency. Since the architectures presented in this paper span several generations in VLSI technology, it is difficult to augment the comparison with normalized cost. Consequently, we do not explicitly compare cost or use a cost/performance ratio. However, it should be noted that these architectures will reach a higher performance/price ratio than most interactive volume rendering options currently available.

7 Comparison

Table 3 presents a comparison of the five architectures. The categories include (1) status - the present stage of development, (2) algorithm - type of ray casting algorithm used, (3) memory partitioning - the organization of the memory system, (4) interpolation hardware - size, type, and/or implementation of the interpolation filter, (5) gradient hardware - size, type, and/or implementation of the gradient filter, (6) shading - shading algorithm supported, (7) perspective support - the ability to handle perspective projections, (8) real-time data input - the potential to support real-time streamed input, (9) target technology - type of implementation, (10) performance limitations - bottlenecks in the system, (11) scalability - the ability to scale performance using multiple rendering pipelines, (12) algorithmic acceleration - early ray termination support, (13) sample processing efficiency - normalized acceleration metric that also accounts for algorithmic speedup, (14) published SPPS - performance numbers obtained from the respective publications, and (15) architectural highlights - features considered important for next generation volume rendering architectures.

The VOGUE architecture supports three rendering modes based on its gradient computation kernel (8, 32, and 56 voxels). Voxels are re-fetched on average 8, 32, and 56 times for mode 1, mode 2, and mode 3,

Table 3: Architecture comparison.

	VOGUE	VIRIM	Array Based Ray Casting	EM-Cube	VIZARD II
Started	1993	1994	1995	1997	1998
Status	Simulated	System built	Simulated	ASIC in development	Simulated
Algorithm	Image order	Object order	Object order	Hybrid order	Image order
Memory Partitioning	Eight-way	Eight-way	Orthogonal slice	Skewed block	Four-way
Interpolation Hardware	Trilinear	Programmable	Nearest neighbor	Trilinear	Trilinear
Gradient Hardware	Three Modes (8/32/56 voxels)	2D Sobel filter	26-voxel neighborhood	Central differences	Quantized gradient
Shading	Phong	Programmable	Programmable	Phong	Phong
Perspective Projections	Yes	Yes	No	No	Yes
Real-time Data Input	Moderately Difficult	Difficult	Easy	Easy	Difficult
Target Technology	VLSI	Off-the-shelf components	FPGA	VLSI	DSP/FPGA
Performance Limitation	Memory	Bus	Memory	Memory	Memory
Scalability	Moderate	Hard	Easy	Easy	Moderate
Early Ray Termination	Yes	No	No	No	Yes
P, Sample Processing Efficiency	mode 1: 0.075 mode 2: 0.01875 mode 3: 0.01071	0.2 $(B = 200 \frac{MVoxels}{second})$	0.1 $(m = 10)$ 1 $(m = 1)$	0.95	0.075 (1:1 sampling) 0.3 (4:1 supersampling)
Published SPPS (one module)	40×10^6	40×10^6	240×10^6 $(m = 10)$	533×10^6	56×10^6 (4:1 supersampling)
Architectural Highlight	High-quality parallel and perspective rendering	Algorithmic flexibility and shadows	Double-buffering	High performance	Good performance cost ratio

respectively. VOGUE can accommodate multiple point light sources with an unrestricted Phong illumination model. Each module is memory limited and capable of $40 \times 10^6 SPPS$ performance in the fastest rendering mode. Because of its random memory access pattern, VOGUE's memory bandwidth utilization is 20%. As a result, VOGUE's sample processing efficiency is between 0.00357 (mode 3) and 0.025 (mode 1) without algorithmic acceleration. Assuming that an algorithmic speedup of 3 is realizable due to early ray termination, the sample processing efficiency in Table 3 has been multiplied by 3.0. In large multi-module configurations, VOGUE's sample processing efficiency per module may decrease due to network overhead.

Image quality is the primary focus of the VIRIM architecture. This architecture is capable a flexible illumination model including shadows. VIRIM uses programmable DSPs that support other rendering, shading, and interpolation algorithms. VIRIM is capable of $40 \times 10^6 SPPS$ performance. Multiple engines can be used to increase performance; however, each engine must duplicate the entire dataset. The object order algorithm leads to no random memory accesses ($r = 0.0$). However, sample processing efficiency is limited by a global bus between the re-sampling hardware and the rendering hardware. The sample processing efficiency, P, is the ratio of the sustainable sample throughput of the bus between the Geometry and Ray Casting Units and the peak voxel throughput of the memory system. $P = 0.2$ for $\frac{40MSample}{second}$ bus bandwidth (VME bus) and assuming the eight memory modules can sustain $200\frac{MVoxel}{second}$. A new memory architecture was introduced in [47] that enhances the memory bandwidth utilization; however, the voxel re-fetch factor V is increased. This new memory system uses buffers and a pre-fetch mechanism to achieve a sample processing efficiency of 0.125.

The Array Based Ray Casting architecture strives for high performance. It uses slice-by-slice processing to render the dataset. The two arrays used in this architecture are a 2D array of processing elements. In FPGA implementations, feedback paths in the larger Ray Array limit the maximum clock speed. The double-buffered memory offers support for real-time data input. The architecture is fully pipelined and parallel and uses only local communication. However, the hardware required to implement the two arrays scales with $O(n^2)$, where n is the dimension of the volume data. This architecture does not use interpolation leading to lower image quality and does not support perspective projections. The performance of this architecture is memory limited. In the full implementation, the Ray Array contains $1.5n \times 1.5n$ processing elements. In this configuration, memory bandwidth utilization, U, is 1.0. However, one drawback is the size and cost to implement the $O(n^2)$ Ray Array. One solution is to shorten the Ray Array (reduced columns) and to re-use them multiple times per projection. In this smaller configuration, the memory bandwidth utilization is inversely proportional to the number of times the Ray Array is shared. The sample processing efficiency, P, is $\frac{1.0}{m}$, where m is the number of times the Ray Array is re-used. Theoretically, m can be as large as $1.5n$, where n is the dataset resolution, greatly reducing the size of the ray array and severely limiting the performance of the architecture. m can be assumed to lie be between 1 and 10 for practical implementations. Using this assumption, this architecture has a sample processing efficiency of 1 (best-case) and 0.1 (worst-case). m is a design parameter for this architecture based on target cost, size, implementation technology, and performance.

EM-Cube is a highly optimized parallel rendering architecture and it was designed to render high-resolution datasets at real-time frame rates on a PC or workstation. The skewed memory system and slice-parallel processing allows EM-Cube to sustain a large memory bandwidth. EM-Cube uses memory skewing on a block granularity. This reduces chip pin-out and communication costs. This architecture can sustain $533 \times 10^6 SPPS$ using four $133MHz$ SDRAMs. During perspective projections certain view points may adversely affect performance for hybrid order algorithms. As a result, EM-Cube does not support perspective projections. EM-Cube uses central differences for gradient estimation which requires 32 voxels; however, no performance penalty is incurred since EM-Cube uses on-chip storage to buffer sample values. Only local communication between processing pipelines are used, therefore, EM-Cube is highly scalable. EM-Cube's memory utilization is 1.0 because it can sustain synchronous memory accesses for each of its voxels. By processing the dataset in sections, EM-Cube is able to significantly reduce on-chip storage [42]. Consequently, EM-Cube's average voxel re-fetch is increases slightly because some voxels on the boundary of a section must be re-fetched from the memory system. EM-Cube's average voxel re-fetch is 1.05 for eight sections on a $256 \times 256 \times 256$ dataset. As a result, EM-Cube's sample processing efficiency is 0.95.

VIZARD II uses three methods to reduce memory bandwidth requirements for interactive visualization: 9-bit quantized gradient, caching, and early ray termination. VIZARD II supports perspective viewing, multiple cut planes, and segmentation. The quantized central difference gradient requires preprocessing and may degrade image quality when compared to traditional central difference gradients. In general, gradient quantization can be used to support a wide range of gradient filters and the size of the gradient table can be increased to enhance accuracy (i.e., image quality). The VIZARD II architecture has been simulated and is still at the research stage. A maximum performance of $56 \times 10^6 SPPS$ performance can be achieved assuming 4:1 supersampling of the dataset. Maximum performance is limited by the memory system. A worst-case performance of $14 \times 10^6 SPPS$ occurs for 1:1 sampling of the dataset. The performance of this architecture is memory limited. Since VIZARD II uses early ray termination, we assume that its algorithmic speedup is 3.0. VIZARD II's average voxel re-fetch is 8. The quantized central difference gradient prevents the voxel re-fetch from being larger than the size of the trilinear interpolation neighborhood. Since VIZARD II's memory system is clocked at 20% of its maximum rate, its memory efficiency factor is 0.2 assuming worst-case (1:1 sampling). The cache memory and Trilinear Interpolation Units are clocked four times faster than the rate of the memory system. When the dataset is 4:1 super-sampled or higher, a substantial increase in cache hits increases the memory bandwidth utilization to 0.8 best-case. This architecture has an overall sample processing efficiency of 0.075 (1:1 sampling) and 0.3 (4:1 sampling) taking into consideration a threefold speedup due to early ray termination.

8 Discussion

In each architecture, maximum performance is limited by either the memory system or global communication bottlenecks, such as buses or networks. Memory limitations are inherent to each architecture. Global buses and networks arise because of the need to communicate voxel data or intermediate values over a common medium. The volume rendering algorithm and memory organization determine whether these potential bottlenecks affect performance. Architectures that are only memory limited tend to be more scalable.

There is a tradeoff in most of the architectures between performance, quality, and hardware cost. VOGUE's different rendering modes present performance-quality tradeoffs. Practical implementations of the Array Based Ray Casting and EM-CUBE architectures use slightly modified configurations from a fully parallel design leading to performance-cost tradeoffs. On the other hand, VIZARD II trades performance versus image quality by using a 9-bit quantized central difference gradient. VIRIM's dataset duplication in the fully parallel implementation illustrates performance-cost tradeoffs. Furthermore, we see general trade-offs between the different types of parallel rendering algorithms. Image order algorithms exhibit greater flexibility, object order algorithms typically have higher sample processing efficiencies, and hybrid order algorithms can have characteristics of both. The primary objectives of the volume rendering architect is to balance these tradeoffs based on the target applications.

Of the surveyed architectures, EM-CUBE and VIZARD II are the only architectures still actively being developed. The primary goal of these architectures is an interactive visualization system that will augment a low-cost workstation or desktop computer. EM-Cube and VIZARD II will fit on a PCI card for a standard PC. As such, both architectures are also designed to be low-cost.

Note that two of the three architectures (Array Based Ray Casting and EM-Cube) do not support perspective projections. Although perspective projections may have less importance in medical and scientific visualization, perspective projections are necessary in virtual reality, volume graphics, and stereoscopic rendering applications. A general purpose volume rendering solution must support both types of projections.

Volume rendering is useful in both scientific visualization and volume graphics. These two paradigms do not have the exact same requirements. Scientific visualization is primarily parallel rendering with simple illumination models; whereas, volume graphics requires greater rendering flexibility. EM-Cube addresses many issues related to interactive scientific visualization. Although VIZARD II has lower performance, it is a more general rendering architecture with support for perspective projections, multiple cut planes, and segmentation.

9 Future of Special-Purpose Accelerators

Recently, interactive image generation rates have been achieved on medium resolution (i.e., $512 \times 512 \times 64$) datasets using 3D texture mapping (a forward-projection algorithm) [14]. One advantage of texture hardware is that volumetric primitives can be mixed with geometric primitives. However, texture mapping hardware does not readily support all aspects of volume visualization, such as per-sample gradient computation and per-sample illumination. In addition, texture memory is usually limited. Volume rendering hardware encompasses texture mapping by including 3D filtering (trilinear interpolation and gradient filters) and very high-bandwidth memory; thus, future volume visualization accelerators are likely to be used for high-performance texture mapping. Texture memory is typically four-way (eight-way) interleaved (similar to VIZARD II's and VOGUE's memory system) to allow conflict-free 2D (3D) re-sampling. A potential area of research is an efficient memory organization that readily supports both volume rendering, texture mapping, and next generation memory technology. In addition, the ability to render both voxel and polygonal primitives in a single projection for virtual reality should be considered.

The architectures surveyed represent the first generation of custom architectures that implement the ray casting algorithm. These architectures primarily focus on high output frame rates. We believe the second generation architectures will address dynamic datasets or large input rates. Fast sampling devices and interactive volume graphics will help promote this trend. Future interactive visualization systems may soon consist of a special purpose accelerator card connected to a real-time data acquisition subsystem. In volume graphics applications, it is expected that volume updates, volume animation, and voxelization will require input frame rates comparable to output frame rates. Consequently, it may be necessary to parallelize voxel input into the volume memory and the memory partitioning scheme in a given architecture may not necessarily be the optimal partitioning scheme for both parallel voxel input and output. Also, double buffered volume memory (see Array Based Ray Casting) will be necessary to eliminate rendering artifacts and performance loss when simultaneously loading and visualizing dynamic datasets. Furthermore, 3D double-buffering follows the development of the traditional graphics pipeline (i.e., double-buffered frame buffer). These are areas for additional research.

If volume graphics is to offset traditional polygonal graphics, volumetric raytracing [48] and perspective projections must be considered. Raytracing is capable of producing photo-realistic images using shadows, reflection, refraction, etc. For instance, VIRIM is capable of shadows. Secondary rays during raytracing have less coherence than primary rays, therefore, volumetric raytracing may be better suited for image order architectures. Ray casting architectures that rely on lock-step coherence between cast rays (i.e., Array Based Ray Casting and EM-Cube) may have difficulty incorporating raytracing.

Memory throughput increased an order of magnitude (e.g., Direct Rambus) since most of the architectures in this paper were first proposed. Advances in semiconductor technologies towards deep-submicron processes will continue to promote higher logic speeds, higher memory density, as well as lower memory access times. In addition, the ability to embed large amounts of memory on-chip with computational units will further enhance memory throughput. All of these trends will simplify future volume rendering architectures, increase their speed, and lower their cost. Furthermore, as these special-purpose accelerators evolve, software and application-program interfaces (APIs) will be defined and developed [49, 50, 51]. They will provide the user with more flexibility and additional features, such as stereoscopic views. The availability of low-cost real-time hardware and industry strength APIs will increase the acceptance of volume visualization and volume graphics. This will certainly lead to the development of new and exciting applications.

Acknowledgments

Special thanks to Meena Bhashyam for help reviewing the manuscript, and Michael Doggett, Christof Reinhart, Michael Meißner, and Gunter Knittel for their useful insight and correspondence regarding their architectures. This work was supported by the Office of Naval Research under grant number N00014-92-J-1252 and CAIP research center at Rutgers State university.

References

[1] R. Yagel, "Towards Real Time Volume Rendering," in *Proceedings of GRAPHICON 1996 Saint-Petersburg, Russia*, July 1996, vol. 1, pp. 230–241.

[2] I. Bitter and A. Kaufman, "A Ray-Slice-Sweep Volume Rendering Engine," in *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, Aug. 1997, pp. 121–138.

[3] M. C. Doggett and G. R. Hellestrand, "Video Rate Shading for Volume Data," in *Australian Pattern Recognition Society Digital Image Computing : Techniques and Applications*, Dec. 1993, pp. 398–405.

[4] S. M. Goldwasser, R. A. Reynolds, and T. Bapty, "Physician's Workstation with Real-Time Performance," *IEEE Computer Graphics and Applications*, vol. 5, no. 12, pp. 44–57, Dec. 1985.

[5] D. Jackel, "The graphics PARCUM system: A 3D Memory Based Computer Architecture for Processing and Display of Solid Models," *Computer Graphics Forum*, vol. 4, no. 4, pp. 21–32, 1985.

[6] S. Juskiw, N. Durdle, V. Raso, and D. Hill, "Interactive Rendering of Volumetric Data Sets," *Computer and Graphics*, vol. 19, no. 5, pp. 685–693, 1995.

[7] A. Kaufman and R. Bakalash, "CUBE - An Architecture Based on a 3D Voxel Map," *Theoretical Foundations of Computer Graphics and CAD*, pp. 689–700, 1988.

[8] H. Pfister, *Architectures for Real-Time Volume Rendering*, Ph.D. thesis, State University of New York at Stony Brook, Computer Science Department, Stony Brook, NY 11794-4400, 1996, MERL Report No. TR-97-04.

[9] H. Pfister, A. Kaufman, and T. Chiueh, "Cube-3: A Real-Time Architecture for High-Resolution Volume Visualization," in In *1994 Workshop on Volume Visualization*, Washington, DC, Oct. 1994, pp. 75–83.

[10] H. Pfister and A. Kaufman, "Cube-4 - A Scalable Architecture for Real-Time Volume Rendering," in *Volume Visualization Symposium Proceedings*, Oct. 1996, pp. 47–54.

[11] A. Kaufman, D. Cohen, and R. Yagel, "Volume Graphics," *IEEE Computer*, vol. 26, no. 7, pp. 51–64, July 1993.

[12] S. W. Smith, H. G. Pavy, and O. T. von Ramm, "High-Speed Ultrasound Volumetric Imaging System – Part I: Transducer Design and Beam Steering," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 38, no. 2, pp. 100–108, 1991.

[13] O. T. von Ramm, S. W. Smith, and H. G. Pavy, "High-Speed Ultrasound Volumetric Imaging System – Part II: Parallel Processing and Image Display," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 38, no. 2, pp. 109–115, 1991.

[14] B. Cabral, N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware," in *1994 Workshop on Volume Visualization*, Washington, DC, Oct. 1994, pp. 91–98.

[15] T. J. Cullip and U. Neumann, “Accelerating Volume Reconstruction with 3D Texture Mapping Hardware,” in *Technical Report TR93-027*, Department of Computer Science at the University of North Carolina, Chapel Hill, 1993.

[16] B. M. Hemminger, T. J. Cullip, and M. J. North, “Interactive Visualization of 3D Medical Image Data,” in *Technical Report TR94-027*, Department of Radiology and Radiation Oncology at the University of North Carolina, Chapel Hill, 1994.

[17] P. Lacroute, “Analysis of a Parallel Volume Rendering System Based on the Shear-Warp Factorization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 3, pp. 218–231, Sept. 1996.

[18] P. Schröder and G. Stoll, “Data Parallel Volume Rendering as Line Drawing,” in *1992 Workshop on Volume Visualization*, Boston, MA, Oct. 1992, pp. 25–31.

[19] T. S. Yoo, U. Neumann, H. Fuchs, S. M. Pizer, T. Cullipand J. Rhoades, and R. Whitaker, “Direct Visualization of Volume Data,” *IEEE Computer Graphics and Applications*, pp. 63–71, July 1992.

[20] A. Kaufman, *Volume Visualization*, IEEE Computer Society Press, 1991.

[21] L. A. Westover, *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*, Ph.D. thesis, The University of North Carolina at Chapel Hill, Department of Computer Science, jul 1991.

[22] M. Levoy, “Display of Surfaces From Volume Data,” *IEEE Computer Graphics and Applications*, vol. 5, no. 8, pp. 29–37, May 1988.

[23] J. Hesser, R. Männer, G. Knittel, W. Straßer, H. Pfister, and A. Kaufman, “Three Architectures for Volume Rendering,” in *Proceedings of Eurographics ’95*, Maastricht, The Netherlands, Sept. 1995, European Computer Graphics Association, vol. 14, No. 3.

[24] P. Lacroute and M. Levoy, “Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation,” in *Proceedings of SIGGRAPH ’94 (Orlando, Florida, July 24-29, 1994)*, Andrew Glassner, Ed. July 1994, Computer Graphics Proceedings, Annual Conference Series, pp. 451–458, ACM Press.

[25] R. Crisp, “Direct Rambus Technology: The Next Main Memory Standard,” *IEEE Micro*, vol. 17, no. 6, Nov. 1997.

[26] R. Yagel and A. Kaufman, “Template-Based Volume Viewing,” *Computer Graphics Forum*, vol. 11, no. 3, pp. 153–167, Sept. 1992.

[27] K. Höhne, M. Bomans, A. Pommert, M. Riemmer, C. Schiers, U. Tiede, and G. Wiebecke, “3D Visualization of Tomographic Volume Data Using The Generalized Voxel Model,” *The Visual Computer*, vol. 6, no. 1, pp. 28–36, Feb. 1990.

[28] B. Lichtenbelt M. Bentum and T. Malzbender, “Frequency Analysis of Gradient Estimators in Volume Rendering,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 3, pp. 242–254, Sept. 1996.

[29] T. Möller, R. Machiraju, K. Mueller, and R. Yagel, “A Comparison of Normal Estimation Schemes,” in *IEEE Visualization Proceedings 1997*, Oct. 1997, pp. 19–26.

[30] K. J. Zuiderveld, *Visualization of Multimodality Medical Volume Data using Object-Oriented Methods*, Ph.D. thesis, Utrecht University, Mar. 1995.

[31] B. Phong, “Illumination for Computer Generated Pictures,” *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, 1975.

[32] J. van Scheltinga, J. Smit, and M. Bosma, "Design of an On-Chip Reflectance Map," in *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, Maastricht, The Netherlands, Aug. 1995, pp. 51–55.

[33] T. Porter and T. Duff, "Compositing Digital Images," *Computer Graphics*, vol. 18, no. 3, July 1984.

[34] G. Knittel, "VERVE: Voxel Engine for Real-Time Visualization and Examination," *Computer Graphics Forum*, vol. 19, no. 3, pp. 37–48, Sept. 1993.

[35] G. Knittel, "A Scalable Architecture for Volume Rendering," *Computer and graphics*, vol. 19, no. 5, pp. 653–665, 1995.

[36] T. Günther, C. Poliwoda, C. Reinhart, J. Hesser, R. Männer, 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 Hardware Workshop*, 1995, vol. 19, No. 5, pp. 705–710.

[37] H.-P. Meinzer, K. Meetz, D. Scheppelmann, U. Engelmann, and H. J. Baur, "The Heidelberg Ray Tracing Model," *IEEE Computer Graphics and Applications*, pp. 34–43, Nov. 1991.

[38] M. C. Doggett, "An Array Based Design for Real-Time Volume Rendering," in *10th Eurographics Workshop on Graphics Hardware*, Aug. 1995, pp. 93–101.

[39] M. C. Doggett and G. R. Hellestrand, "A Hardware Architecture for Video Rate Smooth Shading of Volume Data," in *Eurographics Hardware Workshop*, Sept. 1994, pp. 95–102.

[40] M. C. Doggett, *Vizar : A Video Rate System for Volume Visualization*, Ph.D. thesis, University of New South Wales, 1996.

[41] H. Pfister, A. Kaufman, and F. Wessels, "Towards a Scalable Architecture for Real-Time Volume Rendering," in *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, Maastricht, The Netherlands, Aug. 1995, pp. 123–130.

[42] 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*, Los Angeles, CA, Aug. 1997, pp. 131–138.

[43] A. Kaufman and R. Bakalash, "Memory and Processing Architecture for 3D Voxel-based Imagery," *IEEE Computer Graphics and Applications*, vol. 8, no. 6, pp. 10–23, Nov. 1988.

[44] M. Meißner, U. Kanus, and W. Straßer, "VIZARD II: A PCI-Card for Real-Time Volume Rendering," in *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, Lisbon, Portugal, Aug. 1998, pp. 61–67.

[45] G. Knittel, "A PCI-based Volume Rendering Accelerator," in *In Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, Aug. 1995, pp. 73–82.

[46] G. Knittel and W. Straßer, "VIZARD-Visualization Accelerator for Realtime Display," in *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, Aug. 1997, pp. 139–146.

[47] M. de Boer, A. Gröpl, T. Günther, C. Poliwoda, C. Reinhart J. Hesser, and R. Männer, "Latency-Free and Hazard-Free Volume Memory Architecture for Direct Volume Rendering," in *Proceedings of the 11th Eurographics Hardware Workshop*, Poitiers, France, Aug. 1996, pp. 109–118.

[48] R. Yagel, D. Cohen, and A. Kaufman, "Discrete Ray Tracing," *IEEE Computer Graphics and Applications*, vol. 12, no. 9, pp. 19–28, Sept. 1992.

[49] R. Grzeszczuk, C. Henn, and R. Yagel, "Advanced Geometric Techniques for Ray Casting Volumes," in *SIGGRAPH 98 Course Nbr. 4*, Orlando, FL, 1998.

[50] B. Lichtenbelt, "Design of A High Performance Volume Visualization System," in *Siggraph/Eurographics Hardware Workshop*, Aug. 1997, pp. 111–119.

[51] B. Lichtenbelt, R. Crane, and S. Naqvi, *Introduction to Volume Rendering*, Hewlett-Packard Professional Books. Prentice Hall PTR, 1998.