Three Architectures for Volume Rendering

Jürgen Hesser and Reinhard Männer, University of Mannheim, Germany*,
Günter Knittel and Wolfgang Straßer, University of Tübingen, Germany†,
Hanspeter Pfister and Arie Kaufman, State University of New York at Stony Brook, U.S.A.‡

Abstract

Volume rendering is a key technique in scientific visualization that lends itself to significant exploitable parallelism. The high computational demands of real-time volume rendering and continued technological advances in the area of VLSI give impetus to the development of special-purpose volume rendering architectures. This paper presents and characterizes three recently developed volume rendering engines which are based on the ray-casting method. A taxonomy of the algorithmic variants of ray-casting and details of each ray-casting architecture are discussed. The paper then compares the machine features and provides an outlook on future developments in the area of volume rendering hardware.

Keywords: scientific visualization, volume rendering, ray-casting, graphics hardware architecture

1 Introduction

Volume visualization is concerned with the representation, manipulation and display of volumetric data, typically represented by a 3D grid of scalar values. Volume visualization has become a key factor in the understanding of the large amounts of scientific data generated in a variety of disciplines. Examples include sampled data from biomedical and geophysical measurements, and simulated data from finite element models or computational fluid dynamics (see [7] Chapter 7). Another source of 3D data are volumetric geometrical objects synthesized with volume graphics techniques [9]. Direct volume rendering algorithms are employed to reveal the internal structure of the data [7]. However, their high computational expense limits interactivity and real-time frame rates. The main computational aspects of volume rendering are the massive amount of data to be processed resulting in high storage, memory bandwidth, and arithmetic performance requirements. For example, projection of a 256^3 16-bit per voxel dataset at 30 frames per second requires 32MBytes of storage, a memory transfer rate of 1GBytes per second, and roughly 50 billion instructions per second (assuming 100 instructions per voxel per projection).

Two strategies have been developed to address this challenge. The first makes use of massively parallel multiprocessor architectures to achieve rapid image production rates [20][25][26][28]. The second strategy aims at the development of co-processors or special-purpose rendering engines that separate real-time image generation from general-purpose processing (see [7] Chapter 6). This paper presents and compares three special-purpose architectures that were recently developed to meet the requirements of real-time direct volume rendering.

All three architectures implement *ray-casting*, a volume rendering technique that offers high image quality and maximum flexibility in the choice of viewing parameters [15]. However, the approaches differ greatly in overall machine architecture, processing strategy, performance, and supported dataset resolutions. It is our goal in this paper to reveal the similarities and fundamental differences of these special-purpose ray-casting architectures and to offer an outlook on future trends in special-purpose hardware for volume rendering. The next section describes ray-casting and classifies it into different algorithmic variants, namely object order ray-cast-

* Lehrstuhl für Informatik V, D-68131 Mannheim, email: [hesser,maenner]@mp-sun1.informatik.uni-mannheim.de
† WSI/GRIS, D-72076 Tübingen, Auf der Morgenstelle 10-C9, email: [knittel,strasser]@gris.informatik.uni-tuebingen.de
‡ Department of Computer Science, SUNY Stony Brook, NY 11794-4400, U.S.A.; email: [pfister,ari]@cs.sunysb.edu

ing, image order ray-casting, and a mixture of the two approaches that we call hybrid ray-casting. The object order approach is implemented by VIRIM, developed at the University of Mannheim, Germany, and presented in Section 3. VOGUE, developed at the University of Tübingen, Germany, implements image order ray-casting and is discussed in Section 4. The third architecture, Cube-3, developed at the State University of New York at Stony Brook, U.S.A., implements a hybrid between these two methods and is presented in Section 5. Section 6 summarizes and compares the main features of these approaches, and in Section 7 we take a look at the future of special-purpose volume rendering hardware.

2 Ray-Casting Algorithms

Ray-casting is a powerful volume rendering technique that offers high image quality while allowing for algorithmic optimizations which significantly reduce image generation times [16][17]. Rays are cast from the viewing position into the volume data. At evenly spaced locations along each ray, the data is interpolated using values of surrounding data point voxels. Central differences of voxels around the sample point yield a gradient as surface normal approximation. Using the gradient and interpolated sample value, a local shading model is applied and a sample opacity is assigned. Finally, all samples along the ray are composited into pixel values to produce an image [15]. Figure 1 shows the classification of ray-casting algorithms into three categories:

- **Object order ray-casting:** The volume is transformed to be aligned with the view direction prior to ray-casting, such that the resample locations coincide with integer grid points [3].
- **Image order ray-casting:** A ray is sent from each pixel and the volume is resampled at sample points along the ray [15].
- **Hybrid ray-casting:** An intermediate image aligned with one of the volume faces is produced and then transformed to the view direction [14][25][27].

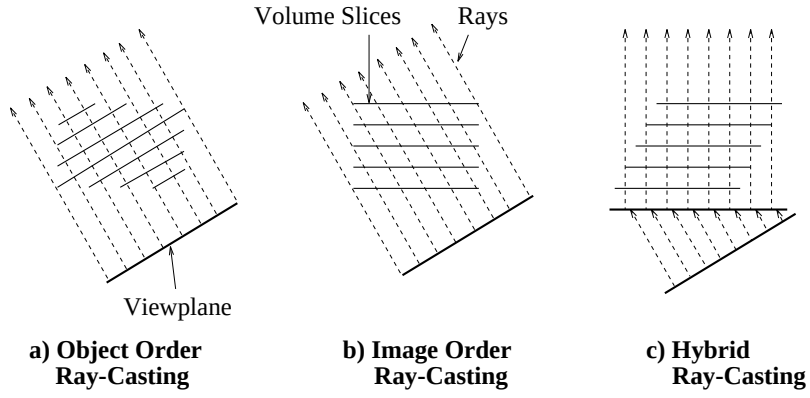

Figure 1. *Ray-Casting Taxonomy*

The object order approach is also called data-parallel volume rendering since operations on the volume semantically involve all voxels at once. For efficiency, the transformation of the volume is typically decomposed into a set of affine transforms requiring shear/scale operations along orthogonal axes [6]. The resulting regular data access to voxels leads to high performance implementations on massively parallel architectures [24][26].

VIRIM [5] implements object order ray-casting in a flexible and programmable fashion. The hardware consists of two separate hardware units, the first being responsible for 3D resampling of the volume using lookup tables to implement different interpolation schemes. The second unit performs the ray-casting according to user programmable lighting and viewing parameters. The underlying Heidelberg Raytracing model [19] allows for arbitrary parallel and perspective projections. VIRIM is discussed in Section 3.

In image order algorithms, the volume is left in its original coordinate system and rays are cast from the image

plane. Traditional implementations pre-compute a color and opacity volume from the original dataset [3]. Colors are assigned by using the local gradient as surface normal approximation and by performing a local shading calculation. A user definable transfer function assigns opacity values based on the data and gradient values [15]. This results in two new datasets, one that holds the color of the shaded samples, and a classified volume that holds sample opacities. For image generation, rays are then cast into these two datasets.

However, any change in classification or shading parameters requires the opacity and color arrays to be recomputed. Storing the gradients and gradient magnitudes of all voxels for fast classification and shading becomes prohibitive for higher resolution datasets. Thus, the algorithm does not allow for interactive exploration and visualization of dynamically changing datasets.

A simple modification to the algorithm allows to operate directly on the original dataset and to perform shading and classification during ray traversal. The original data is resampled along the ray using tri-linear interpolation. The sample gradient is computed using central differences of neighboring voxels. Shading and classification are performed based on the reconstructed sample value and the local gradient.

This approach is taken by VOGUE, a compact, modular, and extensible hardware implementation of image order ray-casting [11][12][13]. For each pixel a ray is defined by the host computer and sent to the accelerator. The VOGUE module autonomously processes the complete ray, consisting of evenly spaced resample locations, and returns the final (sub-)pixel color of that ray to the host. Several such modules can be combined to yield higher performance implementations. VOGUE is discussed in Section 4.

Many methods have been developed to avoid computations in transparent regions of the volume by encoding areas with high-opacity voxels into hierarchical data structures [2][16][18]. These data structures must be accessed once for every ray, leading to multiple traversals and to redundant computation. The achievable data reduction is dataset dependent, and if a wide range of samples must be examined (e.g., temperature distributions) almost no reduction is possible. Image order methods generally lead to redundant data accesses due to the non-uniform mapping of samples onto voxels, since voxels may contribute to more than one ray sample or may be involved in multiple gradient calculations.

To get a mapping of ray-samples onto the volume which is one-to-one, hybrid algorithms transform the volume into an intermediate coordinate system which allows efficient projections onto a face of the volume. This distorted image is then warped (2D transformed) onto the view plane. The intermediate volume transformation typically involves a shear and, for perspective projections, a scale of the original slices of the dataset [14].

Using this factorization, Yagel and Kaufman [27] describe a template based ray-casting scheme to simplify path generation for rays through the volume, and Cameron and Underill [1] efficiently reduce data communication in a SIMD parallel processor. Schröder and Stoll [25] use the idiom of line drawing and achieve sub-second rendering times for a 128^3 dataset on a Princeton Engine of 1024 processors. Lacroute and Levoy [14] recently reported on a fast implementation using a shear-warp transformation, and were able to achieve interactive rendering times for 256^3 datasets on a graphics workstation.

Most of these implementations require a pre-processing step to calculate the gradient field or to generate color and opacity volumes, thereby inheriting the disadvantages discussed above. To avoid any pre-computations, a modified hybrid ray-casting algorithm has been developed for Cube-3 [21][22]. Cube-3 is a high-performance architecture that uses a special memory organization allowing simultaneous access to n voxels parallel to a main axis of the n^3 volume dataset. This memory system allows for the storage of high-resolution datasets without any duplication of the original data, and enables the real-time visualization of dynamically changing volume data, called 4D (spatial-temporal) visualization. Cube-3 is discussed in Section 5.

3 Object Order Ray-Casting: VIRIM

VIRIM is an object order ray-casting engine currently being assembled and tested at the University of Mannheim [5]. It provides a flexible resampling method and freely programmable shading. The hardware of VIRIM consists of several modules, each composed of a geometry unit for volume rotation, resampling, and gradient computation and a ray-casting unit for the final image generation.

The VIRIM system has been designed to achieve 10Hz frame rates for 8 million voxels using the Heidelberg Raytracing algorithm [19]. In contrast to many other direct volume rendering algorithms, the Heidelberg Raytracing model allows shadowing (see Figure 2b). Shadows are generated by considering that incident light is absorbed when cast into the volume. Two light sources are placed at 0° and 45° with respect to the viewing direction. At each sample point along these paths, light is partially absorbed and reflected towards the viewer using the Phong shading model. X- and Y-gradients are estimated with a 2D Sobel operator [4]. The final image is composited in front-to-back order, whereby the light is attenuated a second time on its way to the viewer.

3. 1 General Architecture

The basic approach taken in VIRIM is the division between volume resampling, performed by the geometry units, and the subsequent image generation, performed by the ray-casting units (see Figure 2a). This division is fostered by the different data access patterns of these units. Due to the object order approach, the geometry unit requires access to the whole original dataset for the resampling of a possibly rotated, translated, and zoomed data volume. The ray-casting unit, on the other hand, requires access to the resampled data only along the major viewing directions.

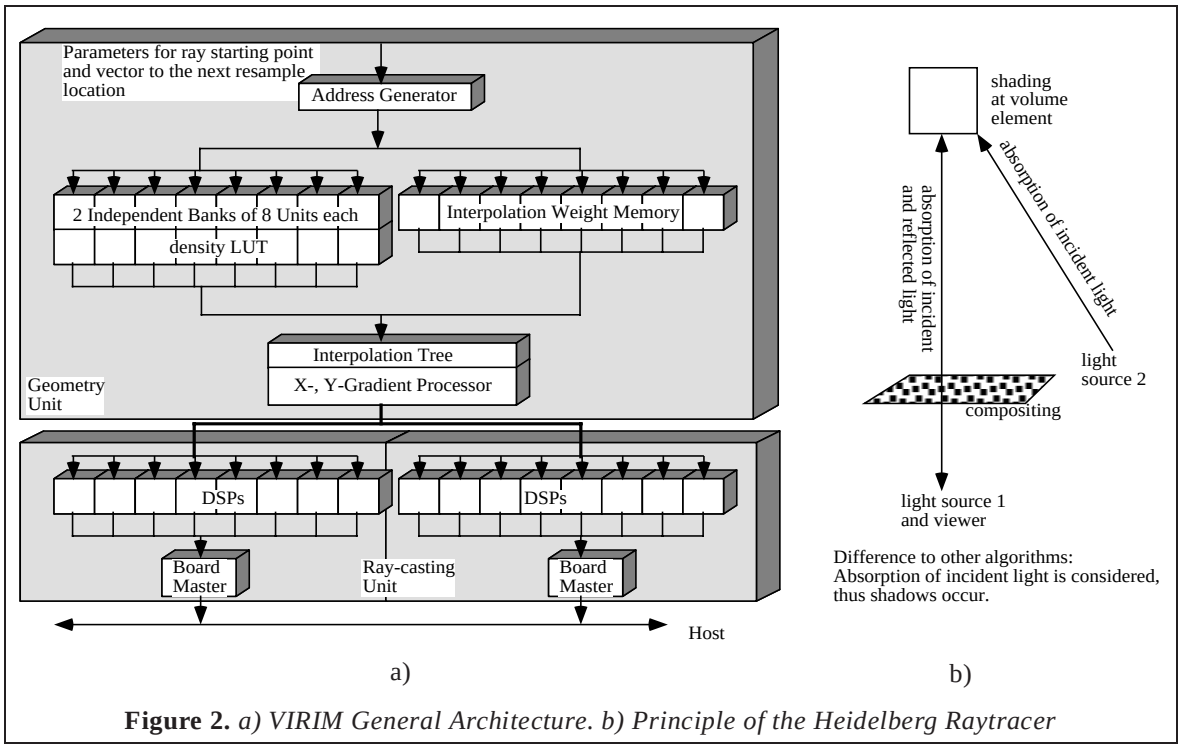

Figure 2. *a) VIRIM General Architecture. b) Principle of the Heidelberg Raytracer*

3. 2 Geometry Unit

In a software implementation, 80 percent of the computation time per projection are required for geometry operations, perspective and gradient calculations, and resampling using VIRIM's basic visualization algorithm. The geometry unit contains special-purpose hardware for these operations and one unit can generate 26 to 36 million transformed locations per second using true 3D resampling. The maximal original dataset size for 16-bit voxels is 256^3 for the currently used 4Mbit DRAMs and 512×512×256 for 16Mbit DRAMs. The maximal size of the resampled dataset is limited to 512^3 by the gradient buffer size.

For each scanline of light rays, the starting point of the ray and the vector to the next resample location are stored. Address generation hardware generates the positions of the sampling points, whereby the rounded X-, Y-, Z-components denote the addresses of the neighbors in the original dataset.

A dedicated memory system inside each geometry unit allows data access rates of up to 640MBytes per second (40MHz×8 neighbors×2Bytes/voxel) using commercial DRAMs. This high data rate is achieved by accessing the 8 voxel neighbors conflict free and by using the Fast Page Mode, alternate memory bank read-out, a 1-entry cache for storing previously used data voxels, and a controller that returns 0 when the address lies outside of the data cube.

Rotation using backward mapping is performed by weighted interpolation among the 8 voxel neighbors of the resampling location. Before interpolation, the voxel values are mapped onto density values using a density lookup table (LUT). This feature allows simple grey value segmentation that is a valuable tool for the visualization of biomedical data without prior segmentation.

A special feature of VIRIM is that the interpolation weights are precalculated and stored in weight memories of size 256K×16. Each weight is addressed by three 6-bit fractional parts, one per coordinate, of the resampling location. 6 bits resolution turned out to be sufficient to make artifacts, originating from the discreteness of the weights, invisible to the viewer. Different interpolation filters like tri-linear interpolation or a local approximation of a *sinc(x)* can be used in order to improve the resampling quality.

After interpolation a gradient hardware estimates the X- and Y-component of the gradient using 2D Sobel filters.

$$W_{i,j} = \frac{1}{8} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

Data resolution and accuracy of all calculations are 16 bits for density and gradient values. All memories and LUTs can be accessed by the host system so that they can be reloaded in real-time.

3. 3 Ray-Casting Unit

Using the sample and gradient values of the rotated dataset, the ray-casting unit generates the final image. In order to allow maximum flexibility, VIRIM uses programmable digital signal processors (DSPs) for this task. Any shading model can be implemented provided that all data for the calculations remain in one sample plane.

The sample density and gradient values are transmitted from the geometry to the ray-casting units over a bus with a peak transfer rate of 240MBytes per second at 40MHz. Each DSP receives its values from a first-in-first-out (FIFO) buffer that can be accessed asynchronously. This mechanism decouples fast geometry units from slower DSPs. For the Heidelberg Raytracing algorithm, a geometry unit is about a factor of 16 faster than a DSP.

Within 15 clock cycles the DSPs compute the interaction of light and material for each volume element. One or more scanlines of the resulting projection are calculated per DSP and stored in local memory.

A local master CPU on the board collects all scan lines of the final image from the DSP memories and transfers the results to the host system. If required, all intra-DSP communication is carried out via this local master CPU. However, for our current visualization model no communication between DSPs is necessary.

3. 4 Performance Estimation

The performance of VIRIM is estimated under the condition that the Heidelberg Raytracing algorithm is used. One VIRIM module consists of four boards of size 36×39cm^2, and four modules fit into one crate.

# of Modules	Dataset Size	Frame Rate
1	256×256×128	2.5Hz
4	256×256×128	10Hz
8	256×256×256	10Hz

Table 1: *VIRIM Performance Estimation*

4 Image Order Ray-Casting: VOGUE

VOGUE is a compact and scalable image order ray-casting unit [11][12][13] which provides interactive rendering speed at moderate hardware costs. Real-time performance can be achieved by operating multiple units in parallel. The basic unit consists of the volume memory and just four VLSI chips. Nevertheless it provides arbitrary perspective projections (e.g., for walk-throughs), Phong shading, a freely movable point light source, depth-cueing, and interactive, non-binary classification using opacity and color transfer functions.

Rather than using pre-shaded and pre-segmented datasets, all processing is performed on the fly on the original voxels. For each resampling location a specific set of neighboring voxels is read out from which the function value, the local gradient and the gradient magnitude are computed. Function value and gradient magnitude are then used as pointers into several lookup tables, which hold the classification transfer function (opacity α), the color assignment (RGB), and material properties (such as the specular reflection coefficient k_s) for shading. Each sample is Phong shaded and the intensities of all points along a ray are composited in front-to-back order according to their opacity.

Image quality and rendering speed can be balanced according to the users requirements. In the fastest operation mode the sample value S is tri-linearly interpolated from $V_0 \ldots V_7$ (see Figure 3a), whereas the gradient components are approximated by $G_x = P_1 - P_0$, $G_y = P_3 - P_2$ and $G_z = P_5 - P_4$. Due to the eight-port memory

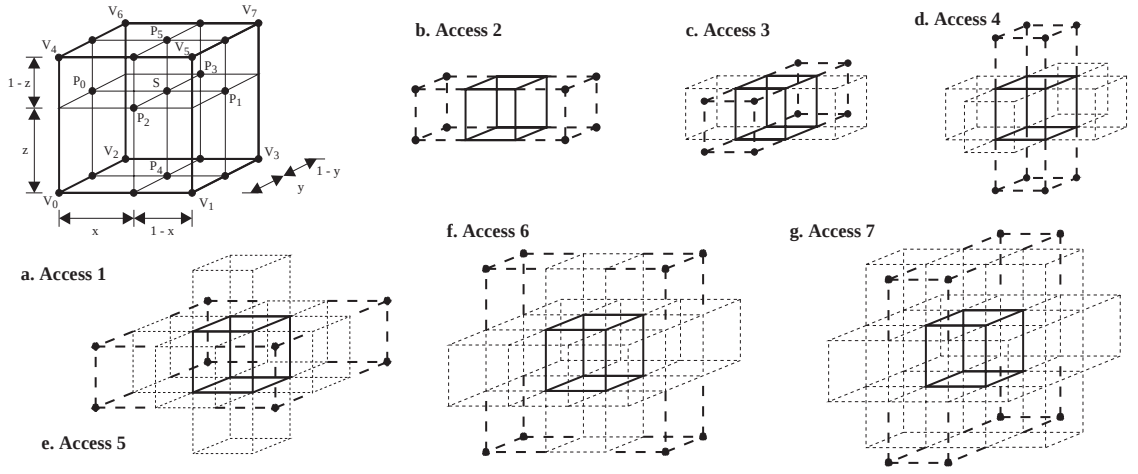

Figure 3. *VOGUE Access Types and Gradient Approximation*

system, the processing of a sample point requires only one memory access in this mode.

Three additional accesses, as shown in Figures 3b - d, enable a more accurate gradient approximation. Access 2 yields all values needed to compute the x-components of the gradients at the corner positions of the cell in question. These components are then fed into the tri-linear interpolator to give G_x at the resample location. The same is done for the y- and z-direction. In this mode, the processing of one raypoint takes place in four steps.

The accuracy of the gradient can further be enhanced by spending three more accesses (see Figures 3e through g). The x-components of the gradient at the cell corners are then computed from weighted surrounding voxels $V_{a,b,c}$ according to

$$G_x(x,y,z) = \sum_{a=x\pm1} \sum_{b=y-1}^{y+1} \sum_{c=z-1}^{z+1} (W_{a-x,\,b-y,\,c-z} \cdot V_{a,\,b,\,c}) \qquad \text{where} \qquad W_{\pm1,\,1\ldots-1,1\ldots-1} = \begin{bmatrix} 0 & \pm0.5 & 0 \\ \pm0.5 & \pm1 & \pm0.5 \\ 0 & \pm0.5 & 0 \end{bmatrix} \quad (2)$$

and again tri-linearly interpolated at the resample location. The same is done for the y- and z-components. In this mode, seven memory accesses are required for the processing of one resample location.

4. 1 General Architecture

The algorithmic steps are mapped in a straight-forward way onto a pipelined architecture as shown in Figure 4a.

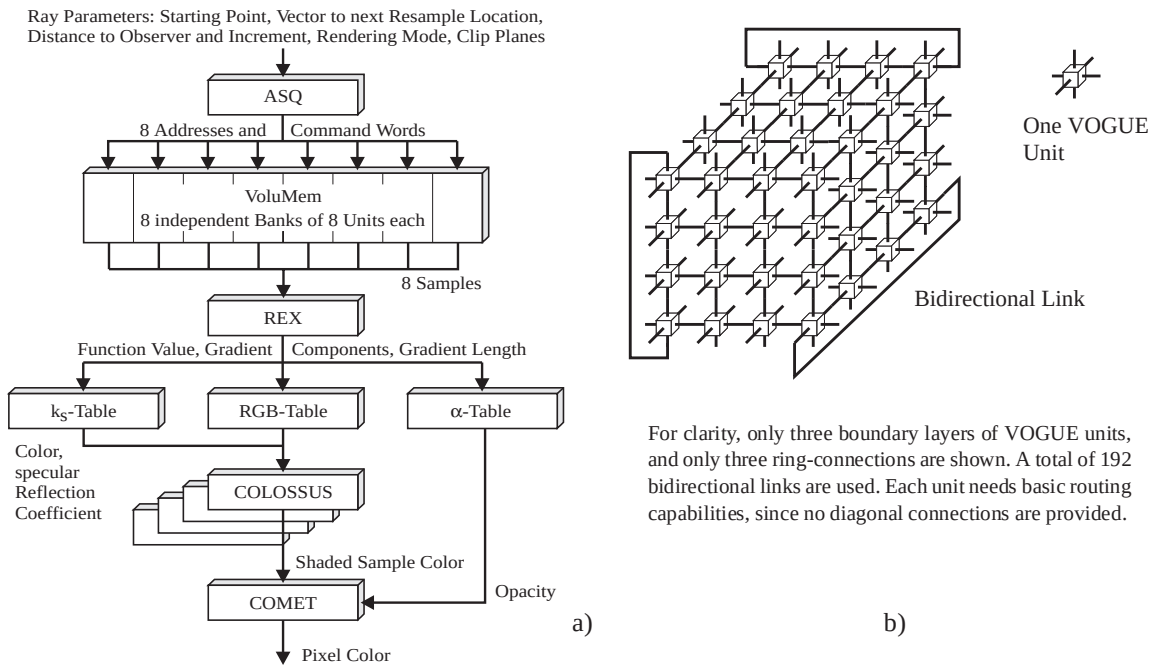

Figure 4: *a) VOGUE Hardware Architecture. b) Parallelization Structure*

After having obtained all ray parameters from the host, the address sequencer (ASQ), a VLSI unit, sequentially generates all raypoints. For each of them, up to seven sets of eight addresses are passed to the memory system. The volume memory (VoluMem), which has a capacity of 256MBytes for 512^3 16-bit voxels, consists of eight independent memory banks and delivers eight voxels per access. The reconstructor and extractor (REX), a VLSI chip, performs the tri-linear reconstruction and computes the gradient and gradient magnitude in all rendering modes. Sample value and gradient magnitude then address several lookup tables to yield the sample color (RGB) and its material properties (opacity α, k_s). The cascadable shading unit COLOSSUS implements the unrestricted Phong illumination model (non-parallel light, perspective projection) for a single point light source and performs depth-cueing. The compositing unit COMET finally sums up the intensities of all points on a ray and passes the pixel color to the frame-buffer.

4. 2 Parallel Operation of Multiple Units

The dataset is divided into subvolumes, which are distributed across different units. Each unit processes a given ray as long as it traverses through its own subvolume. On exit, the properties defining the ray at this point are sent to the neighboring unit. Ideally the number of rays which can be processed simultaneously equals the number of units. A special data distribution scheme has been developed which removes the gaps between the subvolumes by replicating a certain set of boundary voxels [12]. A multi-master bus can be used for up to eight engines, requiring a bandwidth of 640MB/s for an eight-fold speedup. For a larger number of units we propose a ring-connected cubic network as shown in Figure 4b for 4×4×4 units. Simulations show that each link needs a transfer bandwidth of about 27MByte/s, giving a total transfer rate of about 5.2GBytes/s.

4. 3 Performance Estimation

A performance estimation depending on rendering mode and dataset size is given in Table 2.

# of Units	Size	Frame Rate / Fastest Mode	Frame Rate / 4-Access Mode	Frame Rate / 7-Access Mode
1	256^3	2.5Hz	0.6Hz	0.3Hz
8	256^3	20Hz	4Hz	2Hz
64	512^3	20Hz	4Hz	2Hz

Table 2: *VOGUE Performance Estimation*

5 Hybrid Ray-Casting: Cube-3

Cube-3 was developed at the State University of New York at Stony Brook for arbitrary parallel and perspective projections of high-resolution volumetric datasets [21][22]. The Cube-3 hybrid ray-casting algorithm requires at most one memory access to each voxel per projection without any pre-processing or data dependent optimizations. Consequently, it allows for the real-time 4D visualization of dynamically changing datasets, for example, of the in-situ fluid flow in rocks. The hardware contains a specially organized and fully distributed memory system that provides enough throughput for the visualization of 512^3 16-bit per voxel datasets at 30 frames per second.

5. 1 Cube-3 Algorithm

Figure 5a shows an outline of the Cube-3 algorithm. In order to fetch every voxel only once per projection from the cubic frame buffer (CFB), discrete voxel-rays are generated from the continuous rays using a 3D variation of Bresenham's line drawing algorithm [10]. This algorithm guarantees constant stepping by a distance of one along the major axis (e.g., Z) of the viewing direction. The steps in the two non-major directions (e.g., X and Y) are stored in lookup tables, so called templates [27]. Each new discrete ray is generated using these templates, thereby avoiding any overlap of voxels between neighboring discrete rays [25].

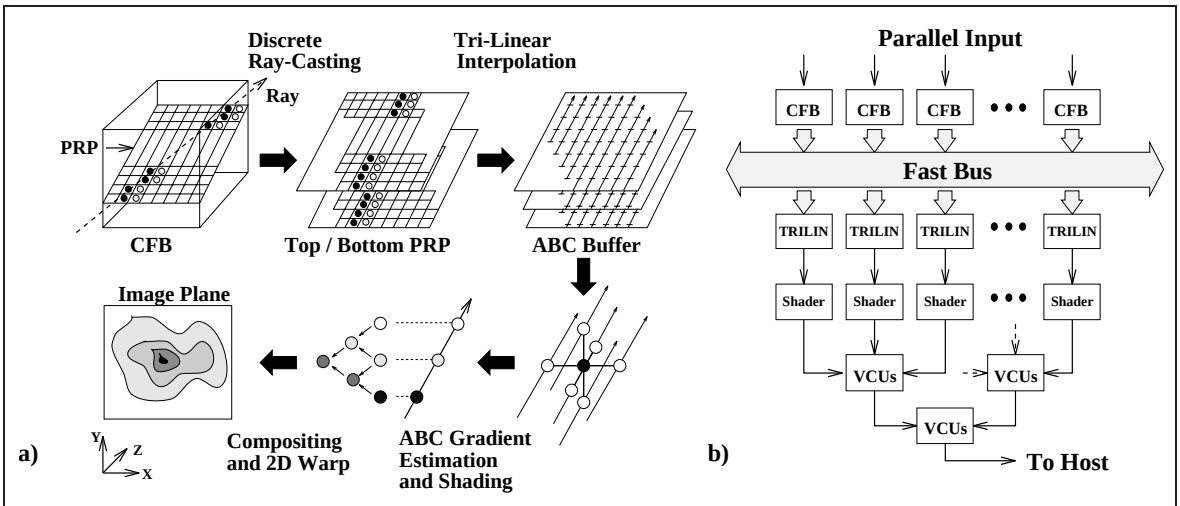

Figure 5. *a) Cube-3 Algorithm Overview. b) Cube-3 General Architecture. (CFB: Cubic Frame Buffer; PRP: Projection Ray Plane; ABC Buffer: Above, Below and Current Buffer; TRILIN: Tri-linear Interpolation Unit; VCU: Voxel Combination Unit)*

The discrete rays are cast from each pixel on the base-plane, which is the volume face that is most perpendicular to the viewing direction. All discrete rays belonging to a scanline of the base-plane form a plane in the dataset. We call this (possibly slanted) plane of discrete ray samples the Projection Ray Plane (PRP). The algorithm projects a distorted intermediate image onto the base-plane. A warping of the base-plane projection onto the viewing plane produces the final image.

Two discrete PRPs (top and bottom PRP in Figure 5a) are used to generate the sample values of the original continuous rays. A sheared tri-linear interpolation is performed using the voxels of neighboring discrete rays from the top and from the bottom PRP [22]. The resulting values yield a continuous plane of ray samples. Three such continuous planes are stored in the above, below, and current (ABC) sample buffers for gradient estimation and shading. In order to evaluate the gradient at a certain resample location, we use central differences between the samples of rays on the immediate left, right, above and below, and along the current ray [22]. This allows to share already computed ray samples between neighboring rays and avoids any additional access to voxels in the CFB.

The samples of the rays are shaded and opacities are assigned using a user controllable transfer function. The shaded rays are composited into a final pixel color using one of several projection schemes, such as first or last opaque projection, maximum voxel value, weighted summation, or compositing. The final base-plane pixel value is transmitted to the host where it is 2D transformed and interpolated into the final view plane image.

5. 2 General Architecture

The Cube-3 architecture is highly parallel and pipelined and allows for the visualization of 512^3 16-bit per voxel datasets at 30 frames per second. Figure 5b shows a diagram of the overall design.

The CFB of an n^3 dataset is organized in n dual-access memory modules, each storing n^2 voxels. A special 3D skewed organization enables the conflict-free access to any beam of n voxels [8]. A beam is a discrete ray of voxels parallel to a primary axis of the CFB. A voxel with space coordinates (x,y,z) is being mapped onto the k-th module by:

$$k = (x + y + z) \bmod n \qquad 0 \le k, x, y, z \le n - 1 \tag{3}$$

The internal mapping (i,j) within the module is given by: $i=x, j=y$. Since two coordinates are always constant along any beam, the third coordinate guarantees that voxels from any beam reside in different memory modules.

PRPs are fetched according to the discrete ray-templates as a sequence of voxel beams. The beams are stored in discrete ray buffers that are part of the TRILIN units shown in Figure 5b. A high-bandwidth global bus (Fast Bus) aligns all discrete rays in each PRP parallel to a main axis of the 2D buffers (see Figure 5a). Using a 2D skewing similar to that of the CFB memory, the 2D buffers support conflict-free storage of beams coming from the CFB and conflict-free retrieval of axis-aligned discrete rays.

Consecutive discrete rays are fetched each clock cycle and placed into n tri-linear interpolation units (TRILINs). Overlapping voxels of neighboring discrete rays can be shared between neighboring TRILIN units. The interpolated continuous ray samples are stored in ABC buffers and the gradients at each sample location are estimated. The interpolated sample and gradient values are forwarded to shading units (Shaders), where a user controlled look-up table of transfer function values assigns each sample an associated opacity value. The color value at each sample location is calculated according to a local illumination model.

The n shading units are the leaves of a folded and circular binary tree that contains a hierarchical pipeline of $n-1$ primitive computation nodes called voxel combination units (VCU). This tree, called the ray projection cone (RPC), takes one ray of opacity and color samples and generates one projected pixel value per clock cycle. The RPC implements all of the projection schemes mentioned above. The resulting pixel of the base-plane is transmitted to the host where it is 2D warped and stored in the frame-buffer.

5. 3 Performance Estimation

Achievable frame rates of Cube-3 are limited only by the data-transfer rate on the Fast Bus due to the fully pipelined implementation of all units. Table 3 gives some examples of performance depending on bus clock frequencies. A Cube-3 implementation for 30 projections per second of a 512^3 16-bit dataset requires 8 custom boards and a specially fabricated bus backplane.

We are currently developing the Cube-4 architecture which overcomes the global voxel communication bottleneck of the Fast Bus. Cube-4 has local and fixed data and control connections between processing elements while still preserving the algorithmic features of Cube-3. Preliminary information is available in [23].

Bus Frequency	Dataset Size	Frame Rate
8MHz	128^3	30Hz
33MHz	256^3	30Hz
66MHz	512^3	15Hz
125MHz	512^3	30Hz

Table 3: *Cube-3 Performance Estimation*

6 Comparison

VIRIM, VOGUE and Cube-3 represent different solutions for volume ray-casting accelerators which are strongly characterized by their main target specifications, i.e. flexibility for VIRIM, compactness for VOGUE and high rendering speed for Cube-3. A comparative summary is given in Table 4.

VIRIM offers flexibility and versatility for a large variety of application areas. Using object order ray-casting, the architecture provides a farm of DSP processors for the ray-casting portion of the algorithm. Fixed functions, such as the reconstruction of arbitrarily oriented slices through the volume, are assigned to dedicated hardware units for maximum speed. The programmable ray-casting processors allow the user to implement different visualization models and to balance image quality versus rendering speed.

The VOGUE project aims at making interactive volume visualization available in single-user workstations. An efficient image order ray-casting algorithm is directly mapped onto a pipelined architecture, yielding a compact and modular design. Higher rendering performance can be achieved by distributing subvolumes of the dataset among several basic units interconnected by a high-speed network. This modularity allows the user to trade machine size for performance.

Rendering high-resolution datasets at high projection rates is the primary goal of Cube-3. A special memory organization and a highly parallel and pipelined machine architecture yield the required high performance. The use of a hybrid ray-casting algorithm allows for exploitation of coherency and successfully solves the traditional memory access bottleneck. The resulting real-time projection rates make new applications such as 4D volume visualization possible.

To sustain the high-memory bandwidth requirements of volume rendering, all three architectures employ memory interleaving. VIRIM and VOGUE use a similar eight-fold interleaving to simultaneously access eight neighboring voxels out of the dataset. A specially skewed and 512-fold interleaved memory in Cube-3 allows for the conflict free retrieval of 512 voxels parallel to a main axis of the volume dataset.

Another characteristic of all three approaches is a pipelined architecture for high sustained rendering performance. The pipeline stages of VIRIM and VOGUE are processing single ray samples. VIRIM employs a DSP farm in the shading and classification stage, whereas VOGUE maintains a fully pipelined architecture for these operations. The architecture of Cube-3 is ray-oriented, and each pipeline stage processes all sample values belonging to a ray simultaneously.

In contrast to Cube-3, which already exhibits the largest possible degree of parallelism, VIRIM and VOGUE offer different approaches for the parallel operation of multiple units. To circumvent data dependency problems, VIRIM replicates the entire volume memory in each parallel geometry unit. VOGUE partitions the dataset and distributes the subvolumes over multiple units, thereby allowing for a high degree of modularity.

7 Outlook

Although great efforts are currently undertaken to reduce the algorithmic complexity of volume visualization towards high-speed software implementations, the evolution of surface-oriented graphics shows that in the long run hardware accelerators are the ultimate solution. In this paper we discussed three different architectures, which have the limited memory bandwidth as their central design aspect in common. A huge amount of data must be read out of the memory and transferred to computational units before it is finally reduced to a single

	VIRIM	VOGUE	Cube-3
Algorithm	Object order ray-casting	Image order ray-casting	Hybrid ray-casting
Overall architecture	Separate geometry and ray-casting (image generation) units	Single, compact module containing volume memory and four VLSI chips	Highly parallel architecture, specially skewed volume memory, ray-projection cone
Interpolation	Programmable interpolation using a LUT	Tri-Linear Interpolation	Tri-Linear Interpolation
Shading	Two fixed light sources, 2D Sobel X- and Y-gradient estimation, shading models (inc. Phong) programmable	Up to four movable point light sources, Phong shading, fast 8-voxel or high-quality 56-voxel gradient estimation	Single, movable parallel light source, Gouraud or Phong shading, ABC gradient estimation using 10 or 12 samples from neighboring rays
Parallelism	16 programmable DSPs per ray-casting unit. 1 geometry and 1 ray-casting unit per module, 4 modules per crate, high-bandwidth interconnection bus, replicated dataset in each geometry unit	Up to 64 basic units interconnected by 5.2GB/sec ring-connected cubic network, each unit fully pipelined, dataset partitioned in subcubes with replicated boundary voxels	Fully distributed and skewed cubic frame buffer, high-speed 8GB/sec bus interconnection, ring-connected ray-projection cone, fully pipelined architecture
Dataset size	256^3, 16 bits per voxel	512^3, 16 bits per voxel	512^3, 16 bits per voxel
Performance	10Hz for 256×256×128, using 4 modules	2.5Hz for 256^3 using 1 unit, 20Hz for 512^3 using 64 units	30Hz for 512^3, at 125MHz bus clock
Operational Status	Prototype assembled, undergoing testing	Proposed architecture, software simulation	Proposed architecture, simulation in software and Verilog

Table 4: *Architectural Features of VIRIM, VOGUE and Cube-3*

pixel. Therefore, the future trends in hardware-supported volume visualization might be strongly influenced by the just emerging logic-embedded memory technology. The 256Mbit-DRAM will appear well within this decade, and then a 256^3 dataset of 16-bit voxels will fit on a single chip. The most obvious thing to do then is to place the computational units needed for the visualization on the memory chip as well, and to exploit the enormous internal bandwidth while drastically reducing the required external bandwidth. For example, integrating eight parallel memory subsystems and a tri-linear interpolator on the same chip could reduce the required external bandwidth by a factor of eight. If we succeed in placing the entire visualization pipeline on a single chip (e.g. by using different algorithms, or by placing only a subvolume on a device), communication bandwidth requirements as well as the size and costs of a voxel graphics system could be reduced by orders of magnitude.

Moreover, with the advent of even higher integrated memory devices (the 1Gbit chip has already been presented), true cubic frame buffers holding volume datasets as well as voxelized surface-defined objects will be feasible. Since ray-tracing volumetric datasets is not affected by the complexity of the scene [9], this could represent a great step towards real-time rendering with global illumination models.

8 Acknowledgments

VIRIM is supported by the German Ministry of Research and Technology (BMFT) under grant 01 IR 406 A 8. Special thanks to Christof Reinhart, Christoph Poliwoda and Thomas Günther who mainly designed hardware and software of the VIRIM system. The work in the VOGUE project was done for the research project SFB 328 funded by the Deutsche Forschungsgemeinschaft DFG. The Cube-3 project has been supported by the National Science Foundation under grants MIP 88-05130 and CCR-9205047. Special thanks to Lisa Sobierajski and Rick Avila for helpful discussions and to Frank Wessels for his MS work during his stay at Stony Brook.

9 References

[1] **G. Cameron and P. E. Underill**, "*Rendering Volumetric Medical Image Data on a SIMD Architecture Computer*", Proceedings of the Third Eurographics Workshop on Rendering, 1992

[2] **J. Danskin and P. Hanrahan**, "*Fast Algorithms for Volume Ray Tracing*", Proceedings of the '92 ACM Workshop on Volume Visualization, 1992, pages 91-98

[3] **R. A. Drebin, L. Carpenter and P. Hanrahan,** "*Volume Rendering*", Computer Graphics, Vol. 22, No. 4, 1988, pages 65-74

[4] **R. C. Gonzalez and P. Wintz**, "*Digital Image Processing*", Addison-Wesley, Reading, MA, 1987

[5] **T. Günther, C. Poliwoda, C. Reinhard, J. Hesser, R. Männer, H.-P. Meinzer and H.-J. Baur**, "*VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine*", Proceedings of the 9th Eurographics Hardware Workshop, 1994, pages 103-108

[6] **P. Hanrahan**, "*Three-Pass Affine Transforms for Volume Rendering*", Computer Graphics, Vol. 24, No. 5, 1990, pages 71-78

[7] **A. Kaufman**, "*Volume Visualization*", IEEE CS Press Tutorial, Los Alamitos, CA, 1991

[8] **A. Kaufman and R. Bakalash**, "*Memory and Processing Architecture for 3D Voxel-Based Imagery*", IEEE CG&A, Vol. 8, No. 6, 1988, pages 10-23

[9] **A. Kaufman, D. Cohen and R. Yagel**, "*Volume Graphics*", Computer, Vol. 26, No. 7, 1993, pages 51-64

[10] **A. Kaufman and E. Shimony**, "*3D Scan-Conversion Algorithms for Voxel-Based Graphics*", Proceedings of the ACM Workshop on Interactive 3D Graphics, 1986, pages 45-76

[11] **G. Knittel**, "*VERVE: Voxel Engine for Real-Time Visualization and Examination*", Computer Graphics Forum, Vol. 12, No. 3, 1993, pages 37-48

[12] **G. Knittel**, "*A Scalable Architecture for Volume Rendering*", Proceedings of the 9th Eurographics Hardware Workshop, 1994, pages 58-69

[13] **G. Knittel and W. Straßer**, "*A Compact Volume Rendering Accelerator*", Proceedings of the ACM/IEEE '94 Symposium on Volume Visualization, 1994, pages 67-74

[14] **P. Lacroute and M. Levoy**, "*Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform*", Computer Graphics, Proceedings of SIGGRAPH '94, 1994, pages 451-457

[15] **M. Levoy**, "*Display of Surfaces from Volume Data*", IEEE CG&A, Vol. 8, No. 5, 1988, pages 29-37

[16] **M. Levoy**, "*Efficient Ray Tracing of Volume Data*", ACM Trans. on Graphics, Vol. 9, No. 3, 1990, pages 245-261

[17] **M. Levoy**, "*Volume Rendering by Adaptive Refinement*", The Visual Computer, Vol. 6, No. 1, 1990, pages 2-7

[18] **D. Meagher**, "*Efficient Synthetic Image Generation of Arbitrary 3D Objects*", Proceedings of IEEE CS Conference on Pattern Recognition and Image Processing, 1982

[19] **H.-P. Meinzer, K. Meetz, D. Scheppelmann and U. Engelmann**, "*The Heidelberg Ray Tracing Model*", IEEE CG&A, Nov. 1991

[20] **S. Molnar, J. Eyles and J. Poulton**, "*Pixelflow: High-Speed Rendering Using Image Composition*", Computer Graphics, Vol. 26, No. 2, 1992, pages 231-240

[21] **H. Pfister, A. Kaufman and T. Chiueh**, "*Cube-3: A Real-Time Architecture for High-Resolution Volume Visualization*", Proceedings of the ACM/IEEE '94 Symposium on Volume Visualization, 1994, pages 75-83

[22] **H. Pfister, F. Wessels and A. Kaufman**, "*Sheared Interpolation and Gradient Estimation for Real-Time Volume Rendering*", Proceedings of the 9th Eurographics Hardware Workshop, 1994, pages 70-79

[23] **H. Pfister, F. Wessels and A. Kaufman**, "*Cube-4: A Scalable Architecture for Real-Time Volume Rendering*", Technical Report TR-950115, Computer Science Department, SUNY at Stony Brook, NY 11794-4400, 1995

[24] **P. Schröder and J. B. Salem**, "*Fast Rotation of Volume Data on Data Parallel Architectures*", Proceedings of Visualization '91, IEEE CS Press, 1991, pages 50-57

[25] **P. Schröder and G. Stoll**, "*Data Parallel Volume Rendering as Line Drawing*", Proceedings of the '92 ACM Workshop on Volume Visualization, 1992, pages 25-31

[26] **G. Vézina, P. Fletcher and P. Robertson**, "*Volume Rendering on the MasPar MP-1*", Proceedings of the '92 ACM Workshop on Volume Visualization, 1992, pages 3-8

[27] **R. Yagel and A. Kaufman**, "*Template-Based Volume Viewing*", Computer Graphics Forum, Vol. 11, No. 3, 1992, pages 153-167

[28] **T. S. Yoo, U. Neumann, H. Fuchs, S. M. Pizer, T. Cullip, J. Rhoades and R. Whitaker**, "*Direct Visualization of Volume Data*", IEEE CG&A, Vol. 12, No. 4, 1992, pages 63-71