

Memory Management of Large Data Sets in Volume Visualization Systems

Research Proficiency Exam

September 7, 2001

Susan Frank

Center for Visual Computing and
Department of Computer Science
State University of New York at Stony Brook
Email: sfrank@cs.sunysb.edu
<http://www.cs.sunysb.edu/~sfrank>
Advisor: Arie Kaufman

Abstract

Significant improvements in hardware components result in the potential to process and render large volumes with complex features and interactions. This has challenged system developers to manage several gigabytes of memory at interactive rates. Ray casting a 1024^3 volume with 2 bytes/voxel in 30Hz requires memory bandwidth of 60GB/sec. Some of the volumetric system design issues are: time to market, scalability, component costs and physical layout. Some parallel processing strategies include distributed systems, algorithmic optimizations, leveraging graphics accelerator hardware, and designing volumetric accelerator hardware. Distributed systems use relatively cheap off the shelf computers networked together. With networking hardware speeds exceeding 1Gb/sec it has become feasible to distribute very large data sets among several PCs to manipulate and render in real time. Graphics accelerator texture mapping is used for the compositing phase of many algorithms, but it limits the quality achievable. Some algorithmic improvements can be implemented by the operating system or in hardware. SIMD arrays are appropriate for volume designs because most operations are repeated on vast amounts of data. In any of these solutions, memory management is critical. A memory bandwidth bottleneck is when the processors stall frequently to wait for memory fetches. Studies indicate that dividing the volume into sub-volumes that are distributed among the processing elements reduces view dependence. Inherent with this approach is the need for balanced load distribution. Multi level caching allows implementation of smaller L1 caches. In this paper I describe the memory bandwidth bottleneck problem and some of the decisions that are made when designing a volume rendering architecture. In addition I give an overview of some of the most well-known volume rendering algorithms and well established optimizations, and a description of some previous work on volume architectures, work to date on the Cube-5 architectures and the potential to improve cache scheduling for ray tracing based on the on ray dependency graph.

Topics of Discussion

1 Introduction

- 1.1 Desired Functionality
- 1.2 Sources of Volumetric Data
- 1.3 Rendering Pipeline

2 Optimizations

- 2.1 Volume Reconstruction
- 2.2 Ray Casting
- 2.3 Shear-warp Factorization
- 2.4 Texture Mapping
- 2.5 Parallel Processing
- 2.6 Distributed Visualization
 - 2.6.1 PVR
 - 2.6.2 SAMSON

3 Architecture Design Considerations

- 3.1 SIMD Arrays
 - 3.1.1 PAVLOV
 - 3.1.2 Smart Memories
- 3.2 Memory Bandwidth Bottleneck
 - 3.2.1 Data Distribution
 - 3.2.2 Memory Hierarchy
 - 3.2.3 Commercially Available Memory
- 3.3 RAYA Board

4 Volume Architecture Background

- 4.1 VERVE
- 4.2 VIZARD
- 4.3 Cube Architectures
 - 4.4 Cube-4
 - 4.4.1 Cube Took Kit
 - 4.4.2 EM-Cube
 - 4.4.3 Cube-4L

5 Architectures Under Development

- 5.1 VIZARDII
- 5.2 Algorithmic Optimizations
- 5.3 RACE II

6 Ray Tracing Architectures

- 6.1 Pyramid Clipping
- 6.2 Cube-5 Architecture

- 6.2.1 Global Illumination
- 6.2.2 Perspective Projections
- 6.2.3 Mixing Volumes and Polygons

7 Future Work

8 Conclusion

1 Introduction

Volume visualization has many applications such as medical imaging, gaming, CAD/CAM and many others. Significant improvements in hardware components result in the potential to process and render large volumes with complex features and interactions. The size of a typical volume has grown by several orders of magnitude over the past decade. This has challenged system developers to manage several gigabytes of memory at interactive rates. Ray casting a 1024^3 volume with 2 bytes/voxel in 30Hz requires memory bandwidth of 60GB/sec. Several strategies using parallel processing have been developed to utilize system components efficiently in order to accomplish this task. The VolumePro board is a direct volume-rendering PCI board accelerator now commercially available. The next generation in this family of architectures, Cube-5 is now under development. In this paper I will introduce the fundamental volume rendering algorithms and well established optimizations, describe the memory bandwidth bottleneck problem and some of the decisions that are made when designing a volume rendering architecture. In addition I will give a brief description of some previous work on volume architectures, recent optimizations, the Cube-4 and Cube-5 architectures and work in progress.

1.1 Desired Functionality

Goldwasser et al. [11] describe the ideal physicians workstation. A physician's workstation should be an interactive tool that allows reconstruction, segmentation, isosurface extraction, mixing polygons with volumes, realistic lighting, interactive frame rates and the ability to manipulate data interactively. Other areas of scientific visualization, computer-aided design and gaming require additional features such as global illumination and rendering of translucent and amorphous objects. Research in these areas have resulted in the development of a huge number of algorithms that yield a wide

range of image quality levels with varying computational costs. As volumetric data sources continue to expand the basic expectations remain the same.

It is expected of any volume rendering system that there will be a complete and easy to understand application interface. Lichtenbelt [25] presents a volume visualization API, the Voxelator. The functionality includes visibility testing, gradient computation, classification, lighting, projection, sampling, interpolation, ordering. The resulting fragments are processed by OpenGL fragment operations.

1.2 Sources of Volumetric Data

A scalar field is the collection of all values associated with points in a volume. Volume rendering is the process of displaying scalar fields. Sources of volumetric data include computed tomography (CT Scan), ultrasonic imaging, confocal microscopy, magnetic resonance imaging (MRI), positron emission tomography (PET) ultrasonic imaging, laser scanners, depth images estimated by stereo disparity, supercomputer simulation, and geometric models. A volume rendering system capable of processing and rendering a variety of forms of volume data including voxels, point clouds, implicit surfaces and volumes, and polygonal data would be very useful. A volume specified by points that are not located on a regular grid is called a point cloud. If the points are located on a regular grid they are called voxels. Volumetric modeling simplifies rendering constructive solid geometry (CSG) models. Complex geometric models are typically created using operations on geometric primitives and high-order functions. With volumetric modeling all CSG operations are precompiled, resulting in only one volumetric object instead of a CSG tree of geometric primitives. Only one surface intersection routine is necessary for volumetric objects, while in classical ray tracing a different routine is typically created for each geometric primitive. And a solid texture map can be precomputed for each volumetric object, thereby avoiding expensive texture mapping operations during ray tracing.

1.3 Rendering Pipeline

From acquisition to rendering a volume goes through several processing stages. Pre-processing of view-independent volume characteristics generally increases the run-time frame rate. Pre-processing for non-interactive environments may include volume reconstruction, segmentation, gradient estimation, classification, distance coding, volume/screen space subdivision, and data distribution. Reconstruction is the recreation of the original signal from its samples. Segmentation, or feature extraction, is the process of identifying which pixels or voxels from an image belong to a particular object, such as air, bone, fat or tissue. Classification involves determining value to use from the transfer function, which assigns an $RGB\alpha$ value to voxels as selected by opacity. Pre-shading and distance coding are sometimes used for run time speed up. For each voxel the leaping distance to the next possible non-transparent voxel is determined and stored. Gradient estimation uses the gray-level values (gradient) as a measure of surface inclination. The normal is obtained from the gradient vector by the partial derivatives of the gray-level function (dv/dx , dv/dy , dv/dz). These are approximated by the differences between the gray values of the current voxel and its immediate neighbors. The decision of whether to include an algorithm in the preprocessing stages is a trade-off between additional run time storage and computation. If the volume is being acquired and manipulated interactively, volume preprocessing is not an option. Run time processes include view dependent calculations. The algorithms chosen represent a trade-off between image quality and rendering speed. A typical run time pipeline includes trilinear interpolation for resampling, classification, gradient estimation, shading and compositing.

2 Optimizations

Several optimizations are used in order to improve rendering speed. The most appropriate optimization is depends on the required image quality and the available resources. The two main types of coherence exploited in volume rendering systems are locality of reference or spatial coherence, and frame-to-frame or temporal

coherence. . Some parallel processing strategies include distributed systems, algorithmic optimizations, levying graphics accelerator hardware, and designing volumetric accelerator hardware.

2.1 Volume Reconstruction

Image based rendering is used in the WarpEngine [34] and by Kaufman et al. [17] to improve frame rates. The idea is to reconstruct the volume for an updated viewpoint based on the previous image. Marching cubes is a reconstruction technique where an isosurface is approximated by determining all intersections of the level surface with edges of a lattice. Each cube is classified into one of 15 possible cases (reduced by symmetry from the 28 possible based on each of eight vertices is either above or below the surface) which determines how the surface fills the cube. Volume rendering and reconstruction centers around solving two related integral equations: a volume rendering integral (a generalized Radon transform) and a filtered back projection integral (inverse Radon transform). Cabral et al. [4] implement the *filtered back projection* CT algorithm as a resampling problem. The authors treat the filtered version of the projection map as 2D texture map. They implement reconstruction by 2D backprojecting a set of 1D images (rays) using graphics hardware. These are dimensionally decomposed and approximated using Riemann sums over a series of resampled images. By decoupling the two operations and using texture mapping in combination with an accumulation or summing buffer existing high performance computer graphics and imaging computers can be used to both render and reconstruct volumes at rates of 100 to 1000 times faster than CPU based techniques.

2.2 Ray Casting

Ray casting is an image order algorithm in which rays are cast from the eye point through each pixel into the volume space, and the composite $RGB\alpha$ data accumulated. The simpler of the ray casting technique is parallel projection. In Perspective projection rays converge to a point in the distance rather than being parallel to each other. This adds a great deal of complexity to the computation. Ray casting algorithms have

handled ray divergence of perspective projections by either undersampling the far voxels, resulting in “holes” or oversampling the near voxels.

Space leaping is when the rays skip over areas of insignificant opacity. Early ray termination saves processing by terminating a ray (sampling shading and compositing) when the ray reaches a specified opacity indicating that everything behind the ray is obscured by the voxels along the ray up to that point.

2.3 Shear-warp Factorization

Volume slices are first sheared in Lacroute and Levoy [24] and projected onto a base plane. The sheared object space is an intermediate coordinate system that can be mapped from the object coordinate system using a factorization of the view transformation matrix. This allows an efficient projection. By construction, in sheared object space all viewing rays are parallel to the third coordinate axis. In the final step the base plane image is projected onto the image plane. The view transformation matrix M_{view} can be factored as follows: $M_{\text{view}} = PSM_{\text{warp}}$. P is a permutation matrix which transposes the coordinate system in order to make the z-axis the principal viewing axis, S has the form of a shear perpendicular to the z-axis where each z-slice is scaled uniformly. The algorithm requires only one 3D and one 2D resampling step for an arbitrary perspective viewing transformation. The shear-warp algorithm takes three steps; shear and resample the volume slices, project resampled voxel scanlines onto intermediate image scanlines, and warp the intermediate image into the final image. Scanlines of pixels in the intermediate image are parallel to scanlines of voxels in the volume data. All voxels in a given voxel slice are scaled by the same factor. And for parallel projection every voxel slice has the same scale factor.

2.4 Texture Mapping

Akeley [1] first proposed storing the volume as a solid texture on the graphics hardware, and then to sample the texture using planes parallel to the image plane and composite them into a frame buffer using blending hardware. It quickly produced unshaded images with the disadvantage that the volume must be re-shaded and re-loaded

every time any of the viewing parameters changed. Also since $RGB\alpha$ values are interpolated by the texture hardware, non-linear transfer functions are not interpolated correctly. the Dachille and Kaufman [4] use a shear-warp style method where the texture mapping hardware performs the shearing and perspective scaling. The integrity of the ray-casting algorithm is maintained by using the CPU for shading. Density and gradients are precomputed then loaded into the texture memory once. Ray casting using texture mapping hardware avoids the multiple voxel accesses by using the hardware to perform the resampling. In order to use space-leaping and early ray termination, they send polygons to the graphics pipeline oriented in such a way that they are coplanar with the rays that would end up becoming a row of pixels in the final image plane. The data is loaded back into the main memory, then the raw density value and the three gradient components are extracted and used in a reflectance map computation to generate the Phong shading for each sample. Voltx [40] takes advantage of hardware assisted 3D texture mapping and incorporates a light model. Gradient magnitude is interpreted in the context of the field-data value and the material classification. The texture map is applied to a stack of parallel planes, which effectively cut the texture into many slabs, which are composited to form an image. It is similar to shear-warp method in that it uses planes through the volume. The number of planes can be adjusted at run time in the Voltx system. Increasing the number of planes improves the resolution.

2.5 Parallel Processing

Volume rendering naturally lends itself to parallel processing. Data sets produced by scientific applications on large parallel computers are often too large to fit in memory of a single processor. Processing is divided among several processors to increase speed and/or memory capacity. The data may be replicated or distributed. There are three main approaches to parallelization used in volume rendering: demand-driven, data parallel and hybrid. Demand driven (ray-parallel) techniques divided the screen into a number of regions where each region represents a task, a number of processors execute these tasks and whenever a

task is completed the processor requests a new one from the master. Data parallel approaches partition the object space. Viewpoint and light sources may prove to be hot spots in the scene and lead to load imbalances. With ray-parallel techniques, all voxels along a ray are processed simultaneously. This requires global communication between the volume memory and the processing units. Hybrid approaches combine data parallel and demand driven approaches. The volume can be processed using a beam, ray or slice parallel technique. Beam -parallel techniques operate on samples of several neighboring rays across a scanline of voxels that is parallel to a principal axis of the data set. Stepping along slanted planes of rays requires complicated addressing mechanism and leads to non-uniform processor communication. Slice-parallel techniques process consecutive data slices that are parallel to a face of the volume data set. These have uniform communication patterns and fit nicely into an SIMD architecture design approach. It is important to load balance work by dynamically distributing primitives, by subdividing large primitives or by combining the two techniques. Where very large data sets are involved, data distribution is another fundamental issue.

2.6 Distributed Visualization

Distributed systems use relatively cheap off the shelf computers networked together. With networking hardware speeds exceeding 1Gb/sec it has become feasible to distribute very large data sets among several PCs to manipulate and render in real time.

2.6.1 PVR

PVR [37] uses content-based load balancing. Each subvolume region belongs to a global BSP-tree. It contains a user interface based on a Tcl/Tk shell. Processing is executed SIMD style with one master node and several slave nodes, which are designated as either rendering or compositing nodes. Slave nodes can be clustered to perform large tasks. In addition there is one collector node. Caching clusters are used as smart memory for time-varying data and as buffer nodes to optimize the computation during our content-based load-balancing data distribution. PVR was used to render a 512^2 by 1877 voxel subset of the Visible

Human at Sandia with a remote display in San Diego.

2.6.2 SAMSON

The SAMSON Networked Memory Server [38] is now under development at SUNY at Stony Brook to allow memory-hungry applications to execute nearly as fast using the network memory server as they would if all the memory were local to the client. The ready availability of high-speed (>1Gb/sec), low-latency (<10us) networking hardware now makes it feasible to use commodity components to construct a network memory server that can offer a memory paging service two orders of magnitude faster than paging to and from local disks. A network memory server (NMS) is a device that provides clients with access to a large amount of RAM via a fast network memory paging service.

3 Architecture Design Considerations

Some of the volumetric system design issues are: time to market, scalability, component costs and physical layout.

3.1 SIMD Arrays

SIMD arrays provide the power of an ASIC implementation with the flexibility of a solution on a general-purpose machine. They maximize computational capability per unit hardware (chip, board, etc). They have no synchronization delay and no communication overhead. Most SIMD arrays being fabricated today use a 50 MHz clock frequency. The inherent data parallelism in volume rendering and processing utilizes the advantages of SIMD PE's to such a degree that they overcome the SIMD disadvantage of requiring worst-case processing.

3.1.1 PAVLOV

PAVLOV [23] is a two-dimensional array of SIMD processing elements that is intended to be a volume processing and rendering coprocessor. It uses a RISC based controller processor. They show segmentation as an example of volume processing, but since it is programmable other volume processing rendering algorithms can be implemented. Processing elements communicate

with their left and right neighbor, with wraparound. A portion of the memory is associated with each processing element. The volume is distributed so that a complete beam of voxels along the z-axis is stored on each processing element. This allows conflict-free access to any z- slice so there is only a single clock cycle delay between slices. In order to improve voxel access speed for x and y sweeps, the authors add an array plane of Volume I/O (VOLIO) registers (one per processing element) that effectively use the pipeline time of calculations for one slice for pre-fetching the voxels in the next slice. The SIMD processing elements consist of an 8bit ALU, two input registers, a 256 x 8 working memory, a counter register used to perform conditional loads, a register for communication with the processor elements above and below, and a register for communication with the processor elements to the left and the right, the VOLIO register which provides access to the volume in slice order, and a shading unit.

3.1.2 Smart Memories

In [26] the authors introduce Smart Memories, a partitioned, explicitly parallel, reconfigurable, universal computing element. A Smart Memories tile consists of a reconfigurable memory system: a crossbar interconnection network, a processor core and a quad network interface. A quad is a cluster of four processor tiles with a low-overhead, intra-quad, interconnection network. SRAMs are used for best speed and capacity. There is configurable logic in the address and data paths. The basic memory mat size of 8KB is chosen based on a study of decoder and I/O overheads and an architectural study of the smallest memory granularity needed. There is configurable logic in the address and data paths. VLSI delays are used for reconfigurable logic at already required repeaters. Mats support pipelined writes and conditional write operations. They can be configured as local scratchpad memories, direct mapped, set associative or fully associative, or as vector/stream register files. These simple configurations have higher efficiency and can support higher total memory bandwidth at a lower energy cost per access. Two DMA engines are located in the load-store units of the processor. They generate memory requests to the quad and

global interconnection networks. Tiles contain a dynamically routed crossbar, which supports up to 8 concurrent references. The authors demonstrate how a wide range of programming models and types of parallelism with little performance degradation.

3.2 Memory Bandwidth Bottleneck

The use of higher density memory chips has led to a higher demand for memory intense applications yet the bandwidth per MByte of memory has decreased. Block transfers of cache lines between the cache and memory make it possible to get the most bandwidth out of the memory. Interactive displays, around 30Hz, require low latency and high bandwidth. Volume rendering a 1024^3 volume requires an order of magnitude higher bandwidth than a 256^3 volume. Ray casting 1024^3 volumes with 2 bytes/voxel in 30Hz require 900 billion instructions/sec or memory bandwidth of 60GB/sec. In addition, there is an increasing demand for interactive volumetric reconstruction and high quality images with realistic lighting. Adding global illumination increases the bandwidth required for rendering a 256^3 volume from 2.8GB/s to 4.6 GB/s. Studies on the effects of data distribution and memory hierarchy shows that these decisions are fundamental elements in volume architecture design. For example, the Silicon Graphics power Challenge is a shared-memory multiprocessor containing up to 18 Mips R8000 CPUs clocked at 90 MHz. The processors communicate via a bus with peak bandwidth of 1.28 Gbytes/s.

3.2.1 Data Distribution

System architects choose between a shared or distributed data policy. Large volume data sets cannot fit into on-board eDRAM or local DRAM memory, so replication is not possible when designing a system for these. Tasks must be distributed between several processing elements in order to achieve interactive rates. Processing elements must cache the part of the volume required for the task at hand. Volume data is mostly read-only, so there is no cache coherence needed. Lowest miss rates occur when the block size most closely matches cache line size. Igehy et al [14] evaluate the effects of load imbalance on bandwidth requirements in parallel texture

caching architectures. They found that load imbalance increase with the number of texturing units. They define the working set size as the

amount of memory that is being processed at a particular moment in time. Their results show that as the number of texturing units increases, the working set size for each texturing unit decreases. The point of diminishing returns for cache size is well correlated with working set size.

3.2.2 Memory Hierarchy

L2 caching enables implementation of smaller L1 caches. Cache levels that reflect the texture hierarchy are studied in [5, 7]. Today's PC graphics subsystems have relatively high-bandwidth channels to local memory (SDRAM, DDR SDRAM or RDRAM) for color and z-buffers. By using some of this memory and bandwidth for L2 cache Cox et al. [5] show that 2 to 5 time less local memory than the push architecture is required and even a 2MB L2 cache saved from 18 to 140 times the download bandwidth over the pull architecture.

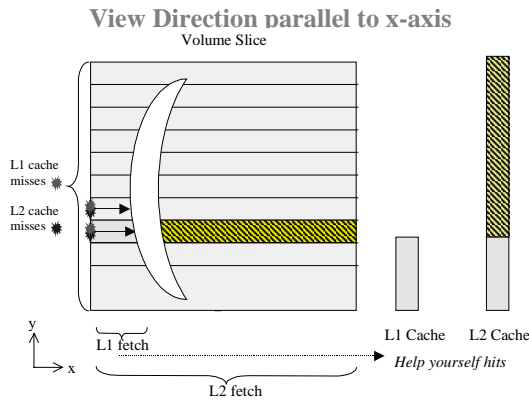


Figure 1. Voxels cached to L2 are not used.

With interactive volume rendering viewpoint independence is important. Palmer et al. [29] show surprising results of viewpoint dependence on the cache hierarchy. If the data is stored linearly by x first, then y, then z, the best cache performance is for a viewing direction along the

y-axis. As shown in figure 1, when a ray travels directly down the x-axis, the first miss fetches into L1 for the next 31 intersections and into L2 the values for the next 127 intersections (*help-yourself hits*). However, the next ray uses none of these cached voxels. However when a ray is traveling down the y-axis the first ray missed in the cache in every ray-voxel intersection. As shown in figure 2, it leaves being in the cache an entire x-y plane of voxel vales in the L2 cache. The dimensions of this plane are the entire y dimension of the data set by the length of a cache line. Because the L1 cache consists of 512 lines of 32 bytes, the data sets used have a y-dimension greater than 512, a single ray down the y axis completely flushes the 1 cache for all three data sets. However such a ray does not flush the L2 cache completely. The next 127 rays, each one pixel to the right of the last, do not miss in the L2 cache at all (*help-your-neighbor hits*). In a similar way, the viewing direction is along the z-axis, and entire x-z plane is left in the L2 cache. These can be fully utilized by scanning across the display in columns instead of by rows. The number of ray-voxel intersections is view dependent because of the data-set extent in each dimension, the effects of diagonal rays in the Bresenham algorithm and the portion of the data set currently visible. The *average time per intersection* normalizes this view dependence for comparison purposes. The view dependence becomes smaller if the data is stored by sub-blocks. The ideal block size found empirically to be approximately half of the L2 cache, thus the ideal number of blocks is data dependent. Their results show that view dependence is more significant than data dependence. The shape of the memory access time graphs are strikingly similar in shape for all three data sets used, both for the optimal number of blocks and for the one block base case. Other results show that more than eight processors making concurrent memory requests as fast as possible saturate the bus.

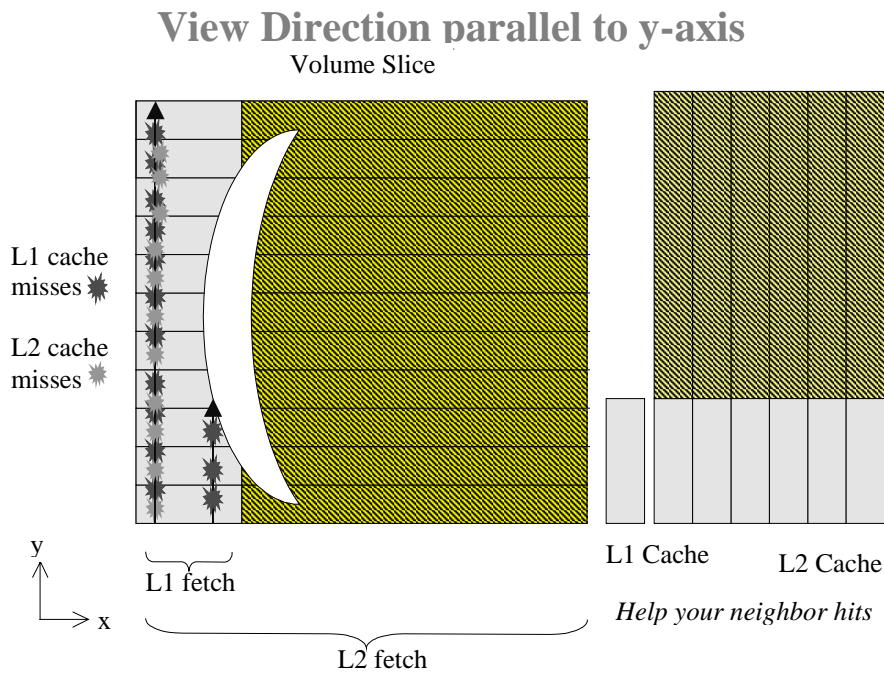


Figure 2. Subsequent rays use L2 caches.

3.2.3 Commercially Available Memory

In addition to decisions about the global structure of the memory hierarchy and data distribution, there are several choices about which memory module to use at any particular level in the memory. These decisions often are driven by factors of cost and available board space. These memory units can be configured as direct mapped or set associative, interleaved, and blocking or non-blocking. With direct mapped memory block n can only go into block $n \bmod \text{block-size}$. With a set-associative design, memory n is placed in block $n \bmod \#\text{sets}$. For example, a 2-way set-associative cache has 2 blocks per set. Cache levels are used to reflect the texture hierarchy by Cox et al. [5] and independently by Dachille et al. [7]. Cox et al. used 2- or 4-way set-associative L1 caches for their texture mapping design because of collisions between MIP levels. They use a fully associative design for the L2 cache because L2 caching must accommodate three additional working sets: intra-object, intra-frame, inter-frame, and multiple texture mapping. A common way of designing a multi-ported cache is to interleave the cache lines across independently

addressed banks. In a texture cache the interleaving across banks should be at the granularity of a texel.

The memory may be pre-configured, and reconfiguration may not be an option. Bender et al. [3] present dynamic search-tree data structures that are memory-efficient at all levels of the memory hierarchy. They do not use any information about memory access times or cache-line or disk-block sizes.

Non-blocking cache allows data cached to supply cache hits during a miss. For example, in GI-Cube [6] the Rambus ASIC cell (RAC) streams voxel read/write operations to and from the RDRAM. Newly retrieved voxels are written to both the cache and the resampling unit.

Here is a brief description of the memory modules currently on the market and their typical usage. Dynamic RAM (DRAM) is generally used for main memory. It allows random access because it has row access scan and column access scan (RAS/CAS). The drawback is that it requires periodic refresh and to write back after read. SRAM, or static RAM is, 8-16 times faster and more expensive than DRAM. It is generally used

for caches. There is no RAS/CAS, no periodic refresh and no write back after read.

Rambus RAM (RDRAM) has a DRAM core without the RAS and CAS. A new interface allows packet switched or split transactions and variable amount of data requests. It has high sustained bandwidth for random accesses, but requires a unique bank address in every four consecutive accesses. Double data rate (DDR) RDRAM allows two accesses per clock cycle effectively doubling the bandwidth. At 800 MHz one RDRAM can supply 1.6GB/s bandwidth. Embedded DRAM (eDRAM) is on-chip DRAM.

3.3 RAYA Board

Figure 3 shows the memory hierarchy for the RAYA architecture [10]. A single RAYA board will connect to the system bus. The RAYA board contains one or more processing ASICs with eDRAM (L1 cache) and some standard memory (SDRAM or Rambus DRAM for L2 cache). The ASICs may contain one or more processing units as well as eDRAM organized into more than one bank. The architecture takes advantage of the high local memory bandwidth to by processing the data currently loaded in the eDRAM device whenever possible. Volume data sets will not fit within the local memory, or even within the L2 cache, so the systems main memory stores the data until the board is ready to work on it. The goal is to avoid fetching the same data more than once.

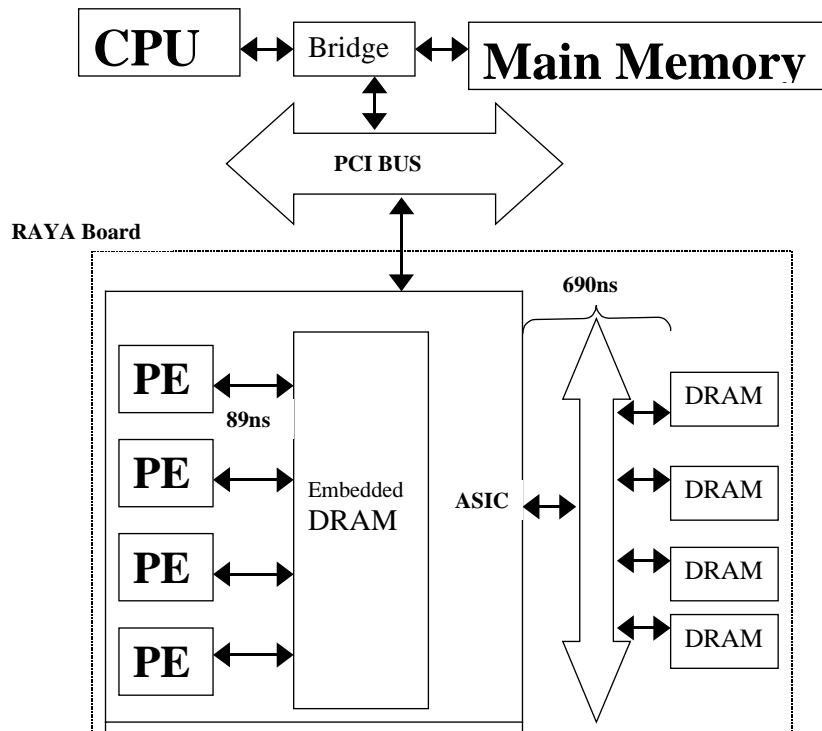


Figure 3. RAYA memory hierarchy

4 Volume Architecture Background

Volume rendering architectures have been developed since the early 1980's. The GODPA/Voxel used as the underlying architecture of a physician's workstation [11] capable of rendering a 64^3 system of 4 bit voxels at 16 frames/second. Voxels were introduced and used to create an interactive workstation prototype that allowed cut planes, depth cues, shading, and a trackball for changing viewpoint. It is a hierarchical pipelined hardware design.

4.1 VERVE

In [17], Knittel introduced VERVE. The memory interface uses eight different memories so all voxels necessary for trilinear interpolation can be fetched in parallel. A 512^3 data set can be processed using 16 SIMMs.

4.2 VIZARD

VIZARD [19] is a PCI-based volume-rendering accelerator that uses DMA to access the volume from main memory. Perspective ray casting is implemented using distance coding. Cache is addressed using the Manhattan Distance of the voxel block relative to the Manhattan Distance Reference Point, which is the closest point of the volume to the observer. A $2 \times 2 \times 2$ voxel-block is used, which means that the maximum Manhattan Distance with a data set of 256^3 is 384. The data is preprocessed for segmentation, shading and data compression. The ray casting stage is accelerated by the hardware. Bandwidth is reduced using a Redundant Block Compression scheme and a SRAM cache exploits ray-to-ray coherence.

4.3 Cube Architectures

Cube-1 [16] performed the first opaque parallel projection of $16 \times 16 \times 16$ data sets using the Cubic Frame Buffer (CFB), 3D skewed volume memory, and a voxel multiple write bus for ray projection. The CFB contains memory modules for distributed skewed storage of voxels as well as an address generator and a control unit. The scene within the CFB could be viewed from one of the six orthographic viewing directions. The voxel multiple-write bus (VMWB) holds a beam of voxels, which compete for the voxel depth bus.

The closest to the observer wins. Each memory module is physically connected to its VMWB processing unit. Cube-2 is a full-scale VLSI-based volume visualization system of Cube-1, capable of interactive rendering rates for 512^3 data sets with arbitrary parallel projection using parallel memory access. Cube-3 [33] performed ray casting using a ray projection cone with various composition algorithms in 3D interpolation and shading units using a modular fast bus; 2D skewed buffers, and pipelined processing of rays in a composition tree. The cubic Frame Buffer CFB is a 3D memory organized in n dual-access memory modules, each storing n^2 voxels. The Projection Ray Plane contains all the rays belonging to the same scan line of a parallel or perspective projection. It can be fetched in n cycles. At the same time it is 2D sheared, or rotated about the viewing axis, by the high bandwidth Fast Bus. Rays are fetched conflict-free and placed into TRILIN units. The tri-linear interpolation uses a look-up table for some combinations of constants and two bi-linear interpolations for efficiency. The resulting continuous projection rays are placed onto ABC Shading Units and converted into intensity and opacity values as determined by lighting and data segmentation parameters. These are hardwired to the Ray Projection Cone (RPC), which can generate one projected pixel value per clock cycle. The RPC is a hierarchical pipeline of $n-1$ primitive computation nodes called Voxel Combination Units (VCU), processing $\log n$ rays simultaneously in a pipelined fashion. A 10-neighborhood gradient estimation and tri-linearly is used to avoid motion aliasing that results when the viewing direction changes from one major axis to another.

4.4 Cube-4

The Cube-4 architecture technology is now commercially available in the form of the Volume-Pro board, produced by Mitsubishi Electric, and the U-Cube ultrasound visualization system produced by Japan Radio Co. Cube-4 uses the skewed memory scheme and local communication between processors to implement the shear warp volume-rendering algorithm. Each voxel of the data set is accessed exactly once per projection. Beams of two adjacent data slices of

voxels are processed simultaneously to compute a new slice for interpolated sample value in between these two slices. The approximate surface normals needed for shading and classification interpolated sample values are used to estimate the gradients on each sample position. The ahead, current behind buffers store the samples one slice ahead and one slice behind in the each sample is shaded and classified by an opacity transfer function. The samples are composited onto the base-plane. This image is transformed onto the viewing plane. The ABC buffer uses shift and delay operations so that spatially adjacent voxels, and those from consecutive slices, can be merged for linear interpolation. Current samples are input without delay as the *ahead* sample. The *ahead* samples are delayed by n cycles and input as the current samples. A delay of the current samples by n cycles produces the behind samples. A reflectance map lookup-table is used for shading. The next sample along the ray must come from a 3×3 neighborhood inside the next slice (due to 26-connected property). Skewing difference is used to predetermine the compositing unit to send the ray to in any particular direction. Beams are further divided into partial beams of p voxels where p is the number of processors. A separate rendering pipeline processes each voxel of a partial beam, which communicate only locally up to three pipelines away.

The Cube-4 was tested on the Teramac custom-computing machine [15]. Teramac is a configurable custom hardware machine developed at Hewlett-Packard Laboratories. It can execute synchronous logic designs of up to one million gates at rates up to 1 kHz. It was built from custom field-programmable logic arrays (FPGAs) packaged in large multi-chip modules. Teramac provides large numbers of programmable gates, wires and memories that can be configured to implement user designs. The Cube-4 was implemented with five rendering pipelines.

4.4.1 Cube Took Kit

The Cube Took Kit (CTK) [17] was implemented in C++ on a standard PC running Windows NT to provide the means to fully utilize the capabilities of the Cube-4/VolumePro system. The VLI API, which ships with the system, provides the interface necessary for advanced shading and

parallel projection of a single volume. CTK allows the user to render a scene composed of multiple volumes, parallel or perspective cameras, lights, intermixed polygons and clipping planes. The scene can be specified by a scripting language or through a program. Perspective volume rendering is handled using a slabbing technique. The perspective scaling of each slab is handled by texture mapping of the graphics card, then α blended into the framebuffer using channel compositing. Bounding boxes are used to determine if volumes overlap. If the bounding boxes intersect, the volume is cut into slabs then interlaced and blended to form the final image. Mixing geometry is achieved with this technique as well. CTK evaluates tradeoffs to determine the slab thickness based on viewing angle, rendering time is minimized.

4.4.2 EM-Cube

EM-Cube [28] is a VLSI architecture for low-cost, high speed, high quality volume rendering proposed by MERL. They use double buffering, exploit the bandwidth inherent in the SDRAMs, keep the pin-count down between adjacent ASICs and reduce the on-chip storage required to hold the intermediate results of rendering. Blocks can be the largest that allows a b^3 block to fit into a single DRAM page, which allows full bandwidth from the SDRAMs using burst transfers, however the max block size is sensitive to voxel size. Sectioning introduced by de Boer et al. [8] is used to reduce the on-chip buffer area. The volume is divided into horizontal sections, which are each processed in turn. This reduces the slice face area and hence the size of the slice buffers. This requires the voxels to be re-read from memory and intermediate results to be moved back and forth from external pixel memory.

4.4.3 Cube-4L

The ray-slice-sweeping algorithm [3] uses distributed memory. Each voxel is used exactly once and no voxel data is communicated between rays. Each volume slice projects all its voxels, only one inter-slice unit in distance. The compositing buffer sweeps through the volume from front-to-back and combines the images of the volume previously swept with the new slice. At the end of the sweep the base plane image is warped onto the image plane. Object order sweep is image-pixel driven, while the warp is

performed in scan line order. If the viewpoint is behind the data set then a back -to- front order is used. Trilinear interpolation is not needed since accumulated values are always aligned with the voxel grid. Processing pipelines communicate with neighboring pipelines and only globally when the image is assembled and warped into the final image plane. The ahead slice comes directly from the CFB, while the behind and current slices are stored in a FIFO buffer. One sweep is required for a complete parallel projection. Multiple sweeps in different processing directions are necessary for perspective projections.

5 Architectures Under Development

5.1 VIZARDII

A second-generation VIZARD system, VIZARDII is presented in [27]. The system platform is a PCI card with programmable devices. It is capable of rendering 256^3 data sets at interactive frame-rates providing high image quality. They use dedicated memory to store the data set on the PCI card. Shading and classification have been moved from pre-processing into the pipeline to enable interactive change of shading and classification parameters. A table of pre-calculated gradients is used so that the gradient at a sample location requires an eight-neighborhood of voxels instead of a 32-neighborhood. Three architectures are proposed by Doggett et al. [9], one with 64 memory modules, one with eight SDRAM memory modules, and one with four DIMMs are discussed. Each assumes both cubic memories accessing similar to the skewed memory of the Cube architectures, and parallel memory access. They propose a buffering scheme that prevents a second memory stall when a ray crosses into a new sub-cube. The voxels are grouped into sub-cube that fit into a row of an SDRAM. The memory is partitioned among eight parallel memory banks. In the second architecture they propose using 8 SDRAM boards because calculation and distribution of 64 independent addresses exceeds the board space on a PCI-Card. In order to use off-the-shelf memory boards they propose using four DIMMs (Dual In-line memory modules) and replicating data. Two consecutive voxels are stored instead of one. A software

simulation compares each of these with/without early ray termination and buffered memory. The results show that the cheaper four DIMMs solution has a slightly slower rendering rate and that the buffered memory scheme improves it somewhat.

5.2 Algorithmic Optimizations

Based on subdividing the volume into sub-cubes and queuing the rays, Vettermann et. al. [40], implement the context switch hardware for multithreading and use distance coding to achieve efficient implementation of space leaping and early ray termination in hardware. Each pipeline contains eight internal SDRAMs since eight neighboring voxels are required for interpolation. The SDRAMs each have four internal DRAM banks, each of which use SRAMs for storing 512Kbyte data of the selected DRAM page. Changes between pages take 90ns. In order take advantage of the short switching times between banks in SDRAMs and to improve load balancing, the row size of one bank is used for a voxel and its neighboring sub-cubes in the x, y, and z direction are placed into the neighboring banks. The remaining voxel neighbors are placed into a second SDRAM.

5.3 RACE II

The RACE II Engine combines algorithmic and hardware acceleration. Voxel-Blocks are used for deterministic access of the data set. The Race II engine [35] implements space-leaping and occlusion on a voxel-block granularity.

6 Ray Tracing Architectures

Perfect scheduling is not possible for ray tracing algorithms, since there is a strict order relationship (timing) in ray trees. As depicted in figure 4, it is not possible to spawn secondary rays until we find the intersection positions of the rays that cause them to be spawned.. Pharr et al. discuss rendering complex scenes with memory-coherent ray tracing [30]. The scene is divided into an acceleration or scheduling grid. Each cell in the scheduling grid has a queue of the rays that are currently inside it and information about

which of the geometry cells overlap its extent. Eye rays are cast through the image plane and partitioned into coherent groups. The scheduler selects the next group to trace based on which part of the scene are already in memory and the degree to which processing the rays will advance the computation. Triangles are added to memory as needed for intersection tests with the geometry (triangles only for optimization), shading calculations are performed and the texture cache manages the texture maps. New rays that are spawned during shading are returned to the scheduler and added to the ray queues. Finally the image samples are filtered and the image is reconstructed. A two level intersection acceleration scheme is used. The geometry cache is organized around a regular geometry grid. Each collection of geometric objects is stored in its own grid. The granularity of the cache is determined by the amount of geometry in the subvolume (cell). Cells with more than a few hundred triangles contain an acceleration grid in order to increase their granularity. Simulations show that with scheduling computation time is significantly better even with a smaller cache size.

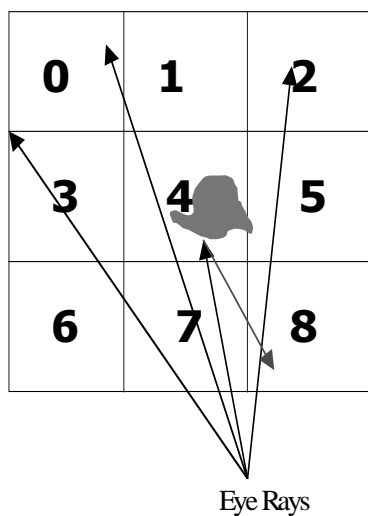


Figure 4. Reflected rays

6.1 Pyramid Clipping

Reinhard et al. [36] extend the hybrid scheduling technique presented in [30], to include data distribution and scheduling using pyramid clipping scheme and the demand-driven

scheduling of shadow ray tracing. Each processor handles both types of tasks, but data parallel tasks are given higher priority. The geometry is distributed and the spatial subdivision structure is replicated. A pyramid is constructed around a bundle of rays and intersected with an octree subdivision of the volume. Pyramid clipping is also used on shadow rays. Load balance is achieved by assigning demand driven tasks to processors that are not busy executing data parallel tasks. Reflected and refracted rays do not exhibit sufficient coherence. These are migrated between processors. This improves the load balance since ray traversal typically takes less than 10% of the total rendering time, so would not be sufficient to balance data-driven tasks. Each processor will store the entire octree, but only the cells assigned according to the data distribution will actually contain geometry data. Despite the large number of inter-reflection rays, the number of shadow rays is still larger. Inter-reflection is costly, so once a hemisphere is sampled its result is stored in an irradiance cache. Area light sources are usually subdivided into patches; a jittered ray is shot towards each of the patches. The shadow rays then all originated from the same intersection point and travel toward adjacent patches.

6.2 Cube-5 Architecture

Cube-5, currently under development at SUNY Stony Brook, is a voxel-based universal rendering and processing engine for volumes, polygons, images and points. Enhanced gradient estimation and super-sampling are used to yield high image quality. It uses programmable processing elements and a dynamic memory hierarchy for flexible implementation and customization. It allows perspective and parallel projections, interactive volume rendering, programmability, ray-directed volume rendering, global illumination, mixing volumes with translucent polygons, overlapping volumes by voxelization, visualization of implicit surfaces, perspective projections, volumetric ray tracing and tomographic reconstruction. It also handles discrete imagery data such as image-based rendering and texture mapping. Volume processing operations allow interactive viewing of the results of segmentation, feature extraction and manipulation. Memory hierarchy and scheduling heuristics allow fast rendering of very

large data sets. Software simulations have been carried out to test much of the functionality of Cube-5.

6.2.1 Global Illumination

GI-Cube is a single PCI board volume ray tracing coprocessor, designed to accelerate volume rendering with Phong shading and local illumination, as well as with global illumination including shadow casting, reflections, glossy scattering and radiosity. In addition it provides volumetric ray tracing acceleration support for various algorithms including hyper-texture, photon maps, polygonal global illumination, tomographic reconstruction, bi-directional path tracing, volumetric textures and BSDF evaluation. Low albedo and high albedo global illumination modes are both provided. Space leaping and early ray termination are incorporated. The digital signal processor (DSP) is directly connected to the framebuffer and has its own SDRAM. It loads the data set, generates lighting and viewing rays, controls processor I/O and sends the result over the PCI interface. The volume is subdivided among the processors. The processors maintain and sort a group of fixed size hardware queues of rays. Each queue is implemented as a pipelined insertion sorter on a separate eDRAM and can hold up to 256 rays. The active queue is processed until it is empty. The queue with the most rays is selected when a processor becomes available. Parallel, distributed memory permits size and bandwidth scalability by the simple addition of identical components. Dynamic load balancing is not implemented. Test results show that 99.6% of computation was dominated by large ray queues and that very short ray queues cause thrashing of the cache. This is because the queue was smaller than the pipeline, so additional rays are not generated before another set of rays from another queue enters the top of the pipeline.

6.2.2 Perspective projections

Kreeger et. al. [20] propose an algorithm that divides the space into slabs called exponential regions base on the view point in which the distance between any two rays doubles at each region boundary. They process the data in slice-order fashion and store the partially computed rays in a compositing buffer, proceeding in either back-to-front or front-to-back order. For back-to-

front processing they use a 2D Bartlett filter with an extent of ± 1 voxel unit in each direction to merge the rays, for front-to-back processing linear interpolation is used. This means that the algorithm can process a row of voxels on a one-dimensional array of processing elements, which only need to communicate with their immediate neighbors.

6.2.3 Mixing Volumes and Polygons

Mixing translucent polygons with volume data has many applications. In pre-operative surgical practice the scalpel can be rendered translucent to give a view of what occurs behind it. Kreeger and Kaufman propose two architectures that each render translucent polygons mixed with volumetric data [22] by dovetailing thin slabs of polygons with volume data. Opaque polygons are pre-warped and aligned along the processing axis so that the volume slice index can be used for the Z-depth check in the final image. Stenciling is used in the final stage of rendering to include these in the image. In [21] they implement this algorithm with 3D texturing in OpenGL.

7 Future Work

Ray-directed volume rendering is utilized in global illumination, mixing volumes with translucent polygons, overlapping volumes, perspective projections, volumetric ray tracing, and tomographic reconstruction. There has been a great deal of research on effective data distribution in parallel volume rendering architectures. Data passing must be minimized and processor workloads must be balanced. Programmable processing elements and a dynamic memory hierarchy are utilized to this end. In order to achieve interactive rates on very large data sets, the volume is organized into cubic blocks, which are distributed among multiple processors. Since cache misses are so costly, each volume block should be processed to the fullest extent possible once it is cached. A cache scheduler chooses the block that contains the most rays in order to reduce cache misses. There is no guarantee that the "most-work" policy is optimal, but this algorithm has been successful. However by determining the optimal schedule, we could take advantage of frame-to-frame

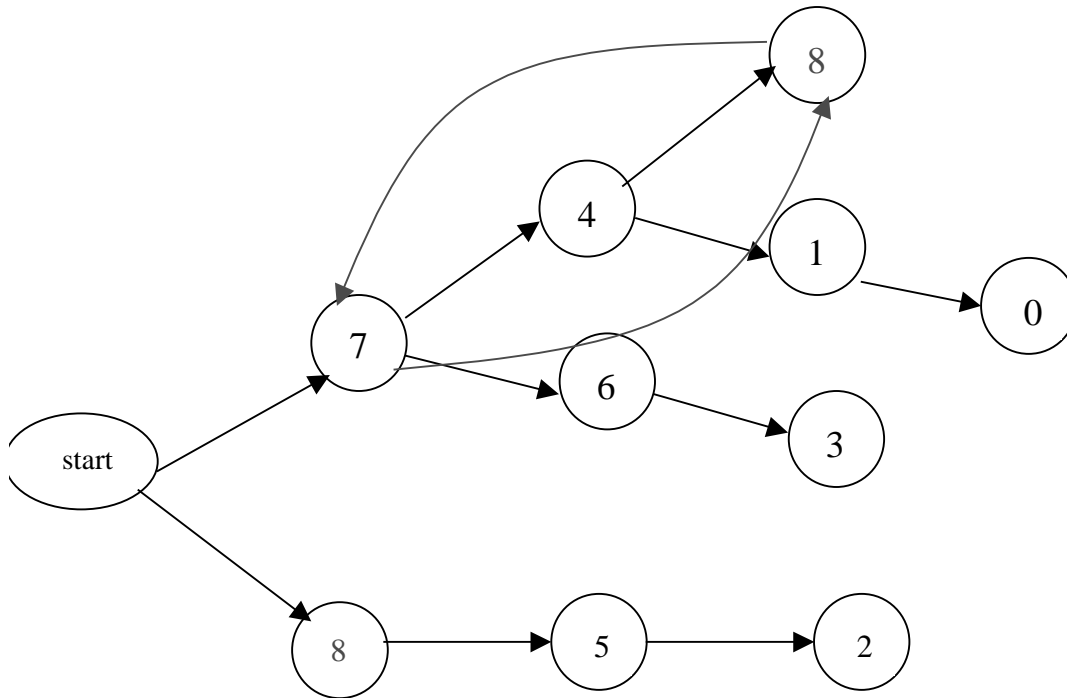


Figure 5. Ray dependency tree.

coherency. In ray tracing, the rays may revisit the same block, thus the ray dependency graph is cyclic (figure 5). This has discouraged researchers from investigating ray dependency graph scheduling algorithms. We are developing a cache-scheduling algorithm for the ray-tracing paradigm. Simulations have shown that the ray dependency graph can be used to improve cache coherency. We are investigating scheduling heuristics to determine the most effective and efficient algorithm, and the feasibility of a hardware implementation. We are also working on using the SAMSON networked memory server to render the Visible Human data set.

8 Conclusion

Memory management is a challenging task with many dimensions. Simply increasing memory capacity does not solve it. Typical scientific and medical imaging data sets do not fit on even a high-end processor with several gigabytes of RAM. As memory gets cheaper and smaller, data sets become increasingly complex and large. With improved networking connections comes the potential to manage larger amounts of data. In

addition, all application domains do not have equivalent resources. Hence there will always be the need to get the most processing done for each dollar spent on processing elements and memory.

Acknowledgments

This work has been supported by NSF and by ONR.

References

1. K. Akeley. RealityEngine Graphics. *SIGGRAPH '93*. August 1993. 27:109-116.
2. M. Bender, E. Demaine and M. Farach-Colton. Cache-Oblivious B-Trees. *FOCS*. 2000. 399-409.
3. I. Bitter and A. Kaufman. A Ray-Slice-Sweep Volume Rendering Engine. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*. 1997. 121-130
4. B. Cabral, N. Cam, and J. Foran, Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. *Symposium on Volume Visualization*. 1994. 91-98.

5. M. Cox, N. Bhandari, and M. Shantz. Multi-Level Texture Caching for 3D Graphics Hardware. *IEEE*. 1998. 86-96.
6. F. Dachille and A. Kaufman. GI-Cube: An Architecture for Volumetric Global Illumination and Rendering. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*. p. 119-129 Interlaken, Switzerland, 2000
7. F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman. High-Quality Volume Rendering Using Texture Mapping Hardware. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*. 1998. 69-76
8. de Boer, A. Gropl, J. Hesser, and R. Manner. Latency and Hazard-Free Volume Memory Architecture for Direct Volume Rendering. *Hardware Workshop of the ACM SIGGRAPH/Eurographics* 1996. 109-118.
9. M. Doggett, M. Meissner, and U. Kanus. A Low-Cost Memory Architecture for PCI-Based Interactive Ray Casting. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. August 1999. 7-14.
10. S. Frank and K. Kreeger. Cache Scheduling for a Ray Architecture. *SUNY Stony Brook Graduate Research Conference*, March 2001.
11. S. Goldwasser, R. Reynolds, T. Bapty, D. Baraff, J. Summers, D. Talton, and E. Walsh. Physician's Workstation with Real-Time Performance. *IEEE Computer Graphics and Applications*. 1985. 44-56.
12. T. Günther, C. Poliwoda, C. Reinhard, J.Hesser, R. Manner, H. -P. Meinzer and H. - J. Baur. VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. 1994. 103-108.
13. Z. Hakura and A. Gupta. The Design and Analysis of Cache Architecture for Texture Mapping. *International Symposium on Computer Architecture*. 1997. 108-120.
14. H. Igehy, M. Eldridge, and P. Hanrahan. Parallel Texture Caching. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. August 1999. 95-106.
15. U. Kanus, M. Meissner, W. Strasser, H. Pfister, A. Kaufman, R. Amerson, R. Carter, B. Culbertson, P. Kuekes, and G. Snider. Implementations of Cube-4 on the Teramac Custom Computing Machine. *Computers and Graphics*. 1997. 21(2):199-208.
16. A. Kaufman and R. Bakalash. Memory and Processing Architecture for 3D Voxel-Based Imagery. *Computer Graphics & Applications*. November 1988. 8(6):10-23
17. A. Kaufman, F. Dachille, B. Chen, I. Bitter, K. Kreeger, N. Zhang, Q. Tang, and H. Hua "Real Time Volume Rendering," *Special Issue on 3D Imaging of the International Journal of Imaging Systems and Technology*, vol. 11. 2000. 44-52.
18. G. Knittel. VERVE – Voxel Engine for Real-time Visualization and Examination. *Computer Graphics Forum*. 1993. 12(3):37-48.
19. G. Knittel and W. Strasser, VIZARD – Visualization Accelerator for Realtime Display. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. August 1997. 139-146.
20. K. Kreeger, I. Bitter, F. Dachille, and A. Kaufman, Adaptive Perspective Ray Casting. *Symposium on Volume Visualization*. October 1998. 55-62.
21. K. Kreeger and A. Kaufman. Mixing Translucent Polygons with Volumes. *Visualization '99*. October 1999. 191-198.
22. K. Kreeger and A. Kaufman. Hybrid Volume and Polygon Rendering with Cube Hardware. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. 1999. 15-138.
23. K. Kreeger and A. Kaufman. PAVLOV: A Programmable Architecture for Volume Processing. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, August 1998. 77-86.
24. P.Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. *SIGGRAPH*. 1994. 451-457.
25. B. Lichtenbelt. Design of a High Performance Volume Visualization System. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. August 1997. 111-119.
26. K. Mai, T. Passke, N. Jayasena, R. Ho, W. Dally, and M. Horowitz. Smart Memories: A Modular Reconfigurable Architecture. *Proceedings of the 27th International Symposium on Computer Architecture*, 2000. 161-71.

27. M. Meissner, U. Kanus, and W. Strasser. VIZARD II, A PCI-Card for Real-Time Volume Rendering. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. 1998. 61-68.
28. 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. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. 1997. 131-138
29. M. Palmer, B. Trotty, and S. Taylor. Ray Casting on Shared Memory Architectures. *IEEE Concurrency*. 1998. 20-35.
30. M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering Complex Scenes with Memory-coherent Ray tracing. *SIGGRAPH*. August 1997. 101-108.
31. H. Pfister, J. Hardenbergh, J. Knittel, and L. Seiler. The VolumePro Real-Time Ray-Casting System. *SIGGRAPH*. 1999. 251-259.
32. H. Pfister and A. Kaufman. Cube-4: A Scalable Architecture for Real-Time Volume Rendering. *Volume Visualization*. October 1996. 47-54.
33. H. Pfister, A. Kaufman, and T. Chiueh. Cube-3: A Real-Time Architecture for Volume Visualization. *Volume Visualization*. October 1994. 73-83.
34. V. Popescu, J. Eyles, A. Lastra, J. Steinhurst, N. England, L. Nyland. The WarpEngine: An Architecture for the Post-Polygonal Age. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. 2000. 433-442.
35. H. Ray and D. Silver. The Race II Engine for Real-Time Volume Rendering. *SIGGRAPH / Eurographics Workshop on Graphics Hardware*. 2000. 129-137.
36. E. Reinhard, A. Chalmers, F. Jansen. Hybrid Scheduling for Parallel Rendering using Coherent Ray Tasks. *Parallel Visualization and Graphics Symposium*. 1999. 21-28.
37. C. Silva and A. Kaufman. PVR: High Performance Volume Rendering. *IEEE Computational science & Engineering*, December 1996. 3(4): 16-28.
38. G. Stark. SAMSON Networked Memory Server Project. <http://www.bsd7.starkhome.cs.sunysb.edu/~samson>
39. Van Gelder and K. Kim. Direct Volume Rendering with Shading via Three-Dimensional Textures. *Volume Visualization*. October 1996. 23-30.
40. B. Vettermann, J. Hesser, and R. Manner. Solving the Hazard Problem for Algorithmically Optimized Real-Time Volume Rendering. *Volume Graphics*. 1999.