to appear in the International Journal of Imaging Systems and Technology, special issue on 3D Imaging

Real-Time Volume Rendering

A. Kaufman, F. Dachille IX, B. Chen, I. Bitter, K. Kreeger, N. Zhang, Q. Tang, and H. Hua

Center for Visual Computing (CVC)*
and Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400

Abstract

With the advent of high-powered, commodity volume visualization hardware comes a new challenge: effectively harnessing the visualization power to enable greater understanding of data through dynamic interaction. We examine Cube-4/VolumePro as the latest advance in real-time volume visualization hardware. We describe tools to utilize this hardware including a software developers' kit, called the Cube Toolkit (CTK). We show how the CTK supports algorithms such as perspective rendering, overlapping volumes, and geometry mixing within volumes. We examine a case study of a virtual colonoscopy application developed using the CTK.

1 Introduction

We are presently in the midst of a paradigm shift in volumetric imaging technology. New commodity hardware systems are rapidly emerging for real-time volume visualization. In the past, inspection of 3D volumetric data was a time consuming offline operation, requiring tedious parameter modifications to generate each still image. For example, a still image of a reconstructed 3D pelvis might take several seconds to several minutes to generate due to the large amount of storage and processing power required for software volume rendering. Although costly texture mapping hardware has been used for interactive, low-quality volume rendering, new hardware volume visualization systems, such as Cube-4/VolumePro [25] by Mitsubishi Electric, enable true real-time 30Hz volume rendering of volumetric datasets. Herein, we describe the methods that we have developed to effectively harness this emerging visualization power for both the developer and end user. We demonstrate our techniques with a virtual colonoscopy application designed to allow mass screening for colon cancer using interactive, guided navigation through a 3D reconstructed model from a computed tomography (CT) scan of a patient's colon.

Much of our data these days comes in volumetric form through either sampling (e.g., CT), simulation (e.g., supercomputer simulation of protein folding), or modeling (e.g., virtual prototyping of replacement bones). Yet computers relegate us to examining 2D projections of our data. To increase our 3D understanding of the objects in 2D images, volume rendering hardware incorporates high-quality shading and interactivity with the kinetic depth effect. The wealth of volumetric data is increasing also in another dimension: time. Dynamic scanning techniques are being developed to acquire such 4D data from patients as a beating heart, brain firing patterns of the thought process, or movement of a fetus during 3D ultrasound. As the hardware is developed to visualize these data in real-time there comes the challenge to make such systems easy to interface and convenient to use. With the onslaught of 3D and 4D information comes the challenge of visualizing and manipulating it to foster greater practical understanding.

Visualization of 3D medical data (e.g., an MRI scan of a patient gastrointestinal tract), is typically accomplished through either slice-by-slice examination or multi-planar reformatting, in which arbitrarily oriented slices are resampled from the data. Direct visualization of 3D medical data is often a non-interactive procedure, despite advances in computer technology. Advanced and expensive graphics hardware for texture mapping allows preview-quality rendering at interactive rates using texture resampling [7]. For increased realism, interactive shading can be incorporated for a slight performance cost [10, 12].

Special purpose hardware for volume rendering is just now appearing in commercial form. Meissner et al. [23] developed hardware based on off-the-shelf digital signal processing (DSP) and field programmable gate array (FPGA) components capable of interactive volume rendering using a single PCI board. Cube-4, developed at the State University of New York (SUNY) at Stony Brook, is an architecture capable of scalable volume rendering suitable for implementations ranging from a single chip up to multiple-VME boards [26]. Cube-4 Light [5] added perspective projection capabilities to the architecture. A close derivative of it was implemented by Japan Radio Co. for inclusion in a 3D ultrasound machine. Mitsubishi Electric implemented a cost effective version of Cube-4 as a single board, called VolumePro, capable of 30Hz rendering of 256^3 datasets with full Phong shading and interactive classification. Additional details can be found in Section 2.

Several application programming interfaces (APIs) have been proposed to standardize volume rendering interfaces [4, 11, 21, 28], although none are designed for dedicated volume rendering hardware. We developed our system around the Volume Rendering Library (VLI) distributed by Mitsubishi to support the VolumePro rendering board. VLI supports basic volume rendering functions and manipulation of rendering parameters, much like OpenGL supports attributed primitive rendering. On top of VLI we designed a software developers' kit for Cube-4, the Cube Toolkit (CTK), in order to ease the burden of designing a working system (see Figure 1). The CTK supports flexible volume rendering with VLI combined with the concept of an OpenInventor-like scenegraph. Although SGI's Volumizer supports volume rendering, it does not support the

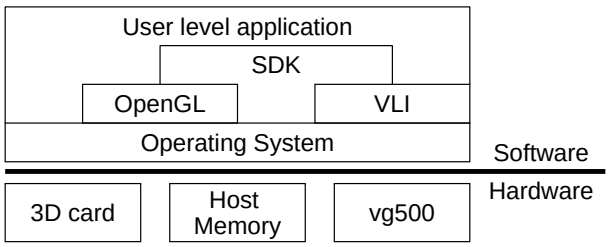

Figure 1: *System-level diagram for an application of the interactive volume visualization system.*

*http://www.cvc.sunysb.edu

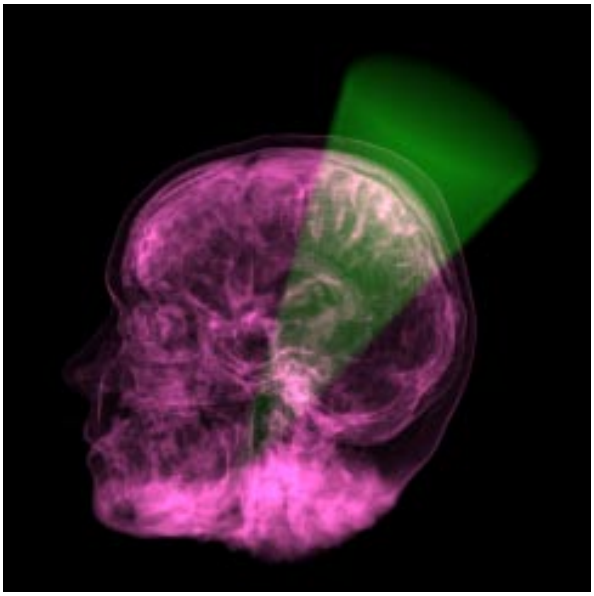

Figure 2: *Intermixed rendering of a radiation treatment beam with a sampled MRI head.*

new breed of high quality volume visualization hardware now coming to market.

We demonstrate in Section 5 the CTK with a virtual colonoscopy application, however, the CTK is flexible enough to support other applications. For instance, a radiation treatment planning system could be designed using the CTK. The CTK allows interactive mixing of geometric radiation beams with the volumetric organ. An engine can compute the effect of the composite beams on the tumor and surrounding tissue and output a dynamic volume of the resulting therapy. The intermixed visualization of all this data is facilitated by the CTK (see Figure 2). Another potential application is a voxel-based flight simulator. Such an application would contain voxelized terrain, volumetric clouds and fog, voxelized *culture* (i.e., man made scenery like houses, buildings, and roads), and pilot specific information such as target highlights, no-fly zones, and dangerous airspace. The CTK handles combined rendering of all these elements as well as coordination for autonomous agents through engines.

2 Real-Time Hardware

2.1 Cube-4

Cube-4, developed at SUNY Stony Brook, is a pipelined, scalable volume rendering architecture which provides hardware support for real-time slice-parallel ray-casting [26]. Others [8, 20, 27] also used slice-parallel ray-casting on massively-parallel machines, but did not achieve real-time frame rates. Additionally, Cube-4 does not require any pre-computation, maintains only one copy of the dataset, computes Phong illumination for every sample, and reads each voxel only once per frame.

Figure 3a shows a top-level diagram of the Cube-4 system with several *Cube-4 units*. By providing direct connections from each unit to its dedicated *Voxel memory* module, Cube-4 solved the memory-processor bandwidth bottleneck. The data is distributed and skewed across the *Voxel memories*. All the Cube architectures [5, 16, 24, 26] utilize a skewed memory organization to enable conflict free parallel access to a row of voxels along any of the major axis. Given M memory modules, a voxel at the 3D position (x, y, z) is stored at memory module $m = (x + y + z) \; mod \; M$.

Voxels of each skewed row are fetched and processed in parallel, distributed over all *Cube-4 units*. Cube-4 processes consecutive data slices parallel to the *baseplane*, the face of the volume buffer which is most perpendicular to the view direction. In each unit pipeline (Figure 3b), after resampling along rays (*TriLin*), the dataflow 4 reaches the (*Shader*) where samples are shaded using central difference gradients and Phong illumination. Next, the shaded samples are composited (*Compos*) and stored in the (*2D memory*). Data for computing a sample along a ray resides in neighboring units, and the nearest-neighbor connections between *Cube-4 units* are used to move the data to the *Cube-4 unit* working on that ray. Using the pixel bus, composited pixels are collected by the *Warp unit* to warp the baseplane image to the framebuffer image.

Along each ray we compute sample values using tri-linear interpolation. Due to the nature of parallel projection, samples in the same slice have the same weights and therefore we can efficiently share intermediate results. Hence, interpolation requires only three multipliers and one row buffer for partially computed samples. After reading, the voxels are buffered on chip in two slice buffers such that they can be used to compute the per sample gradients. Next, we apply a lookup table based transfer function to acquire sample color and opacity. The base sample color is modulated by pre-computed Phong illumination coefficients, indexed by the gradient in a second lookup table. The final shaded $RGB\alpha$ sample is then used in a compositing step which accumulates the contributions along all the ray. Completing computation for all rays yields an image aligned with the *baseplane*. This baseplane image is then 2D-warped onto the user specified framebuffer image.

To handle datasets with more voxels per row than Cube-4 pipelines, we process partial rows generated by breaking a row into equal-sized segments. The size of a row segment is equal to the number of parallel pipelines implemented. Row segments are processed sequentially before starting the next full row.

As neighboring pipelines work on neighboring voxels and rays, only local, fixed communication bandwidth pipeline interconnections are necessary. A simulation of the architecture on the HP Teramac configurable custom hardware machine running at about 1Mhz on a 125^3 dataset with 5 parallel pipelines achieved a frame rate of 1.6 Hz [15]. A VLSI implementation of the architecture scales well even for huge datasets. Running at 100Mhz on 16, 128 or 1024 pipelines, datasets of size 256^3, 512^3 and 1024^3 can be visualized at over 30Hz.

Cube-4 Light [5] improved the algorithm to also allow perspective projections, but with some view dependent filtering artifacts. Enhanced Memory Cube (EM-Cube) [24] augmented the skewed volume storage with a hierarchical blocking scheme which allowed

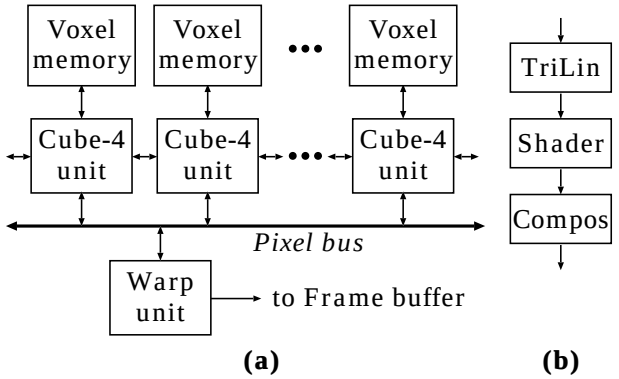

Figure 3: *(a) High level view of the Cube-4 architecture, (b) Cube-4 unit pipeline.*

it to utilize the burst mode of current SDRAM memory chips.

2.2 Cube-4/VolumePro

Mitsubishi Electric has recently completed the manufacturing of the VolumePro/vg500 chip and the corresponding Cube-4/VolumePro board. Being sold under the "RT-Vis" business name, Cube-4/VolumePro is now available on a low cost PCI board delivering the best price/performance ratio of any available volume rendering system. The heart of Cube-4/VolumePro, a single vg500 chip is capable of delivering 500 million shaded volume samples per second giving it the ability to render a 256^3 volume at 30Hz. The first generation of Cube-4/VolumePro supports 8- and 12-bit scalar voxels, allowing the visualization of many of todays common volume dataset formats including 12-bit medical datasets. The design utilizes fixed point internal data formats. To preserve accuracy during rendering, important fields such as accumulated opacity and shaded RGB values utilize 12 or more bits precision. Like Cube-4, Cube-4/VolumePro computes a complete Phong illumination equation at every sample point. It supports weighted summation, minimum intensity, and maximum intensity projection (MIP) as well as front-to-back compositing.

Cube-4/VolumePro computes central difference gradients similar to Cube-4. Additionally, however, Cube-4/VolumePro computes the magnitude of the gradient by first taking the sum of the gradient vector components followed by a single Newton-Rhapson iteration to approximate the magnitude. This can be intuitively considered as a measure of the "strength" of a surface; a high magnitude indicates a high probability of surface crossing. This magnitude value is used in two ways to alter the sample before it contributes to the final image. First, it can be utilized to modulate the opacity of a sample to strengthen (or weaken) its contribution to the image, thus highlighting surfaces in low opacity volumetric data. A gradient magnitude lookup table is utilized to select the amount of modulation from the computed magnitude. Secondly, the diffuse and specular illumination coefficients can be modulated. This allows creation of effects such as a clear, but specular surface in low opacity volumetric data. The gradient magnitude is used to derive shading coefficients, G_d and G_s, for computing the final sample color C_{out} according to:

$$C_{out} = (k_e + (I_d G_d k_d))C_{sample} + (I_s G_s k_s)C_{specular}$$

where k_e, k_d, k_s are the emissive, diffuse, and specular coefficients, I_d, I_s are the diffuse and specular illumination values, C_{sample} is the sample color from the color classification lookup table, and $C_{specular}$ is the specular light color.

Cube-4/VolumePro supports supersampling in x, y, and z in powers of two. This provides higher quality rendering to reduce artifacts by increasing the number of sample points within the volume. Supersampling in x and y are implemented by rendering the volume from the same viewpoint, but shifted by a subvoxel amount in the x and/or y directions, then interlacing the resulting images into a large memory buffer. Supersampling in z is accomplished by repeatedly sampling along the ray in-between two slices.

Although the hardware can only render a single 256^3 volume at a time, there are 128 MB of volume storage on the board (512^3 8-bit voxels) and, beyond this, arbitrarily large volumes can be handled by the driver software automatically swapping volume data to and from host memory. The result is that any sized volume can be rendered at less than 30Hz by cutting it into 256^3 bricks which are rendered separately. The resulting brick images are composited together in image space in the correct order. Also, a subset of a larger volume can be rendered by restricting the volume traversal to a cuboid subvolume.

Cube-4/VolumePro hardware implements a flexible and powerful arbitrary oriented cut plane. This feature allows it to extract an arbitrarily thick slice from the dataset at any orientation, as in multi-planar reformatting of an MRI volume. The opacity of the voxels within the slice are set to fully opaque and the voxels outside are set to fully transparent, while the transition between inside and outside has a variable width, allowing the voxels in the transition to be translucent for antialiasing. This variable falloff region between inside and outside substantially improves the appearance of extracted slices. Cube-4/VolumePro also supports a 3D cursor, as either a crosshair or a set of three axis-aligned planes. This feature makes selection easier and improves spatial relationships when interactively manipulated.

Cube-4/VolumePro supports anisotropic and gantry-tilted datasets by adjusting the rays as they traverse the volume dataset. Since this changes the inter-sample distance along each axis, the opacity values are corrected by a lookup table to represent the proper opacity for the corrected inter-sample distance.

A single vg500 chip is an application specific integrated circuit (ASIC) with approximately 3.2 million logic transistors and 2 Mbits of on-chip SRAM. It is fabricated in 0.35μ technology and runs at 133 MHz clock frequency. The Cube-4/VolumePro board is organized with one vg500 chip, between 4 and 16 volume memory SDRAMs, two section memory SDRAMs, and two pixel memory SDRAMs. The Cube-4/VolumePro algorithm requires an image warp after volume rendering which is achieved by the texture mapping hardware on a standard 3D graphics board. Figure 5 shows a picture of the first generation Cube-4/VolumePro board with the vg500 rendering chip.

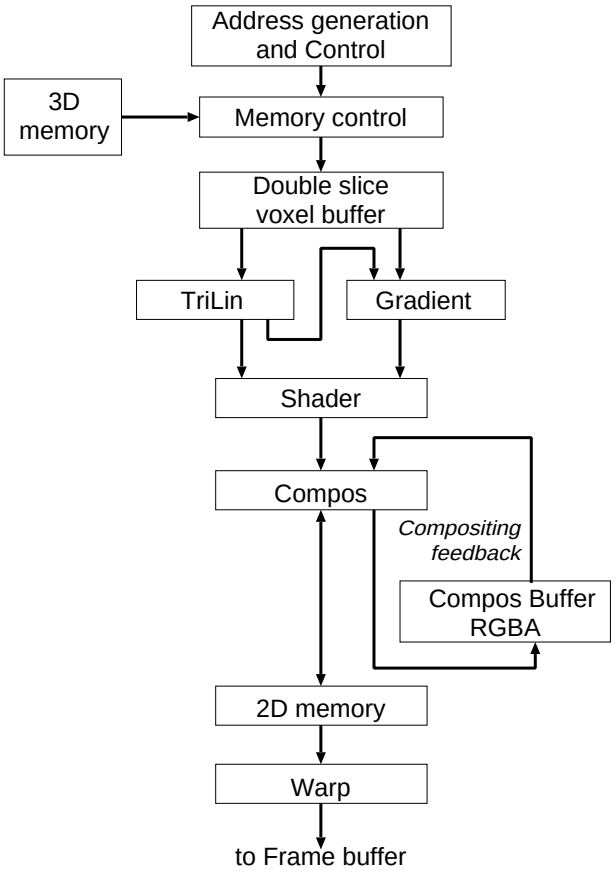

Figure 4: *Cube-4 pipeline dataflow.*

3 Volume Rendering Libraries

3.1 VLI

The Cube-4/VolumePro system provides a Volume Library Interface (VLI) to access all the features of the vg500 chip in a programmer friendly manner. The VLI API consists of a number of C++ classes, such as a volume class used to load and render a volume. VLI, unlike prior volume rendering APIs, provides powerful means to manipulate the advanced shading abilities of the Cube-4/VolumePro board. Several classes represent rendering elements such as lights, cut planes, cropping, color lookup tables, the rendering context, and cameras. A description of some of the classes follows:

VLIVolume Manages volume data handling tasks such as voxel data storage, voxel data format, and transformations to the volume data such as shearing and scaling.

VLIContext Contains and provides manipulation of rendering parameters, and methods for rendering volumes using these parameters. These parameters include one or more of each of the following: lookup table, camera, cropping, cut plane, light, and cursor.

VLILookupTable Defines the lookup tables that contain the color and opacity values to be mapped to the volume. Also used to specify gradient magnitude opacity modulation and gradient magnitude intensity modulation lookup tables.

VLICamera Defines the viewing parameters of one or more cameras, which are stand-ins for the user in viewing the object being rendered.

VLICrop Defines the cropping characteristics, if used, for the rendering process. The VLI supports one crop box.

VLICutPlane Defines the characteristics of a cut plane. The maximum number of cut planes is board-specific.

VLILight Defines a light that illuminates the scene.

VLIGraphicsContext An abstract class specifying the interface between VLI and a native graphics system.

3.2 CTK Overview

Although VLI provides a powerful interface for volume rendering, it still requires tedious custom coding to achieve a working system.

Figure 5: *A picture of the Mitsubishi Electric Cube-4/VolumePro board with the vg500 rendering chip.*

Our software developers' kit for Cube-4 the Cube Toolkit (CTK), provides an easier method to achieve a working system. In addition, as an object oriented C++ library, CTK provides a flexible set of tools which can be combined in interesting ways for high level control and automation. CTK was implemented in object oriented C++ on a standard PC running Windows NT, since personal computers are widely available and affordable compared to workstations running Unix. The basic support for engines was adapted from Apprentice, a free implementation of OpenInventor.

The CTK organizes a virtual scene as a collection of nodes in a directed, acyclic graph. The scenegraph can be specified either programmatically or through a simple scripting language, similar to VRML. The topmost node, a Separator, contains all other nodes in the scene. Like OpenInventor, the graph is traversed depth first and state changes only affect children and right siblings. Nodes represent either volumetric imagery and visible geometry, or attributes that affect the rendering of the scene.

The user of the CTK builds a scenegraph which is automatically rendered into a window. The CTK provides default viewer functionality to allow simple examination of the scenegraph. A scene may be composed of multiple volumes, parallel or perspective cameras, lights, intermixed polygons, and clipping planes, among others. Since VLI currently supports only parallel projection of a single volume, the CTK provides methods to organize all the geometric primitives and render them into a final composite frame. Furthermore, the CTK has engines that allow flexible and creative control over any node in the scenegraph.

3.3 CTK Classes

Most of the VLI classes have been wrapped into CTK nodes. Many other nodes provide additional interactivity and functionality, such as engines. A sample of the complete CTK class tree is given in Figure 6. All classes that may be instantiated into the scenegraph are derived from the Node class. Below is a description of some of the classes:

CtkVolume The CTK volume node wraps the VLI volume class in order to provide additional functionality. The CTK volume node can read and write multiple voxel formats from a file or can read 4D data from a live source. It also provides high level functions such as morphological operators (e.g., erode, dilate, floodfill, binary logical operators) and statistical tools such as a histogram. A histogram is useful for determining pseudocolor mappings and opacity transfer functions.

CtkLookupTable VLI maintains multiple lookup tables to classify and render the volume data. Encapsulating these in CTK nodes allows simple, dynamic modification of lookup tables. For example, an engine can be created that generates an isosurface classification transfer function lookup table based on a scalar value specifying the isovalue. Another possibility is to create random color mappings as a function of the mouse coordinates, to perform a stochastic search for an appropriate transfer function (e.g., [13, 22]). Lookup tables can also be edited manually using a mouse based tool.

CtkEngine Engines are used to implement dynamics in a scenegraph. Engines take some input from external events and scenegraph nodes, compute some function, and output some quantities to other nodes. A simple engine might take as input the current simulation time and outputs a single scalar value as some computed function of time. This can be routed into a transformation node to accomplish simple rotation about some axis. The output of an engine can be a more complex quantity, practically any node or nodal parameter in a scenegraph. Examples of the output of an engine are:

- a 3D vector controlling the cursor,
- a 4D vector defining the cut plane or describing the size and direction of a glyph (e.g., the amount of blood flow in a particular volume region),
- a data array such as a color and opacity transfer function, or
- a more complex data value such as a 3D matrix (e.g., a volume representing fracture probabilities in a bone computed by a volumetric finite element simulation [30]).

Engines can also be written to automatically generate levels of detail of a volume or to perform marching cubes isosurface generation.

CtkCropBox While VLI supports a single crop box per volume, the CTK allows multiple crop box nodes to be active within the scenegraph. The scenegraph automatically combines the multiple crop boxes and renders the correct complex volume region. This is useful for rendering an L-shaped region of a volume, for example. By encapsulating cropping as a node, it allows cropping to be simply specified in a script file and allows dynamic change of properties through engines.

CtkCutPlane The cut plane allows arbitrary planar sectioning of the visible volume. The CTK cut plane node encapsulates the VLI functionality and opens up all the parameters to dynamic change through engines. For instance, the location of a multiplanar reformatted slice can be easily connected to the output of the mouse or other control device.

CtkCursor VLI allows the insertion of a hardware rendered 3D cursor into the volume dataset. The CTK encapsulates the VLICursor class to manage the cursor automatically to allow mapping it to the mouse or other input device.

CtkAppearance The appearance nodes control attributes related to the visual presentation of the scenegraph. For instance, the volume appearance node controls the current style of volume rendering (e.g., maximum intensity projection, minimum intensity projection, weighted summation, or front-to-back compositing)

CtkWindow The Window class is not a node in the scenegraph, but contains a scenegraph root node. It handles user interface functions, like mouse input, and automatic rendering of the scenegraph.

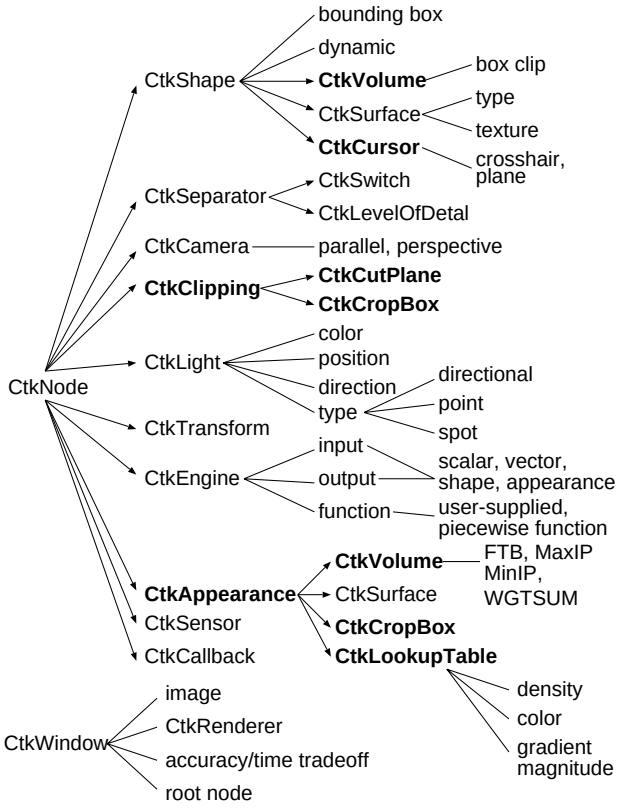

Figure 6: *Sample selection from the complete CTK class tree: CTK classes are prefixed with "Ctk"; those in bold are departures from standard OpenInventor classes. Some sample data members are shown in lowercase. Arrows represent C++ class inheritance, while lines represent class members.*

4 CTK Algorithms

4.1 Perspective Rendering

VLI currently supports only parallel volume projections, yet some applications require perspective projections. For instance, in a colonoscopy application, in which the user is normally viewing along a tubular structure, parallel projection compresses a straight tube into a 2D ring, while perspective creates a view which allows the user to see the interior walls of the tube. To enable perspective rendering, a perspective camera node is added to the scene to replace the default parallel camera. The CTK handles perspective volume rendering using a slabbing technique.

In the slabbing technique, the volume is partitioned into slabs of slices aligned either along a major volume axis or parallel to the image plane. Figure 7 illustrates this process using three axis-aligned slabs. The perspective scaling of each slab is handled by the texture mapping of the graphics card. Since the slab images are drawn from back to front, we blend them into the framebuffer using α channel compositing. Their position in depth is set to the center of the slab which they represent. Obviously, a large number of thin slabs are required when the perspective distortion is great.

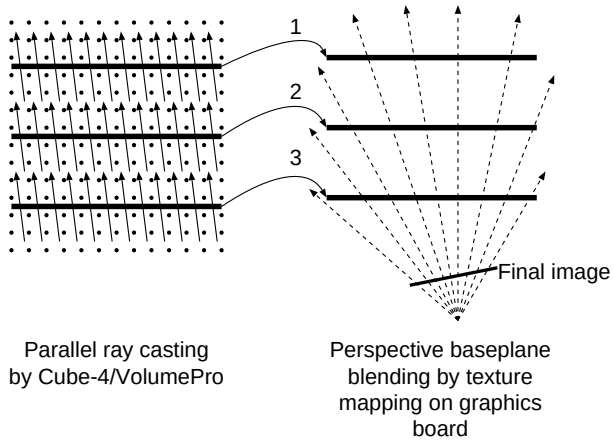

Figure 7: *Approximation of perspective volume projection be rendering slabs using parallel volume rendering hardware and blending the slab images perspectively on the graphics card.*

Slab thicknesses are selected to trade off between quality (with a large number of thin slabs) and speed (with a small number of thick slabs). The Cube-4/VolumePro system is limited to processing cuboid subvolumes (*blocks*) with borders that are multiples of 32 voxels. Thus, once slabs are less than 32 voxels thick, increasing the number of slabs by factor f increases the volume processing time by up to f times. Additionally, there are more slab images to transfer between the Cube-4/VolumePro board and the texture memory.

Drawing fewer slabs introduces artifacts into the image when viewing the volume close to 45 degrees from a baseplane. The projection of the slab images creates artifacts on the edges of the slabs when adjacent slabs are projected more than one pixel apart, creating noticeable steps in sharp edges. These artifacts can be eliminated if the slabs are parallel to the image plane. Unfortunately, rendering non-axis-aligned slabs requires processing the complete axis- and block-aligned bounding volume of the slab. This overhead, shown as the lightly shaded region in Figure 8, is maximized when viewing diagonally from a corner. Depending on the user's preference, the CTK can choose either image- or axis-aligned slabs to provide an accuracy/speed tradeoff.

Since perspective distortion shrinks objects in the distance, it also shrinks the errors in the distance. This allows us to use the principles of adaptive perspective volume rendering [17] to adaptively modify the slab thickness along the viewing direction with constant error bounds. For example, a slab in the back that is twice as far from the viewpoint as a slab in the front projects to half the size; thus, it can be twice as thick and still have the same amount of error. For perspective viewing with a large field of view, this optimization technique can relieve much of the work for the geometry processor, although the volume processor must still process the same amount of data.

4.2 Overlapping Volumes

In a dynamic scene, the moving volumes can easily come across and overlap with each other. It is very important to support the rendering of multiple overlapping volumes. Consider the scenario in which smoke rises up through a cloud, or a radiation therapy beam penetrates a sampled human dataset (Figure 2). Clearly, two overlapping volumetric objects cannot be rendered separately with the images combined. In traditional ray casting algorithms, samples from different volumes interlace with each other and composite in order. For each sample, the color value is obtained by interpolation, classification, and shading. However, Cube-4/VolumePro can only render a single volume at a time.

Our CTK supports rendering of overlapping volumes. When rendering a scenegraph, the CTK traverses the whole scenegraph and computes the intersection of volume bounding boxes. If multiple volumes intersect each other, the CTK enables the overlapped rendering mode and renders the volumes correctly.

Figure 9 illustrates this approach by rendering a sphere overlapping with a cubic volume. Similar to perspective rendering, we cut each volume into slabs, then we interlace and blend them together to form the final image.

For the best quality, all slabs have to be parallel to the image plane. An improved speed algorithm renders the scene aligned with the largest (*anchor*) volume. The anchor volume is rendered using efficient cropping while the others are rendered using cut planes to align them with the anchor volume slices. Figure 9 shows an example of overlapping volumes with the sphere as the anchor volume.

4.3 Mixing Geometry

This method of interlacing slab images of different objects can also be used to properly intermix both opaque and translucent geometry with the volumetric data [18, 19]. For opaque polygons, the geometry is rendered into the framebuffer, then the slab images are drawn with z-depth test to correctly occlude the volume samples behind the geometry. For translucent polygons, the polygons between each slab image are drawn in between texture mapping the individual slab images. This way, all contributing objects are composited in the correct order.

4.4 Rendering a Slab

To generate the slabs, the active subvolume which encloses the current slab is created and a thick cut plane is set up on the Cube-4/VolumePro board. Cube-4/VolumePro requires the active subvolume to be on 32 voxel boundaries. Figure 8 illustrates a top view of a diagonal thick cut plane with its enclosing subvolume. Rendering this subvolume processes many voxels that are very distant from the cut plane. Therefore, the CTK partitions the subvolume into multiple smaller subvolumes which still enclose the slab, but cover less voxels. In our example of Figure 8, these smaller subvolumes appear in a darker grey compared to the single active subvolume. Unfortunately, rendering these multiple subvolumes requires multiple passes (one for each subvolume) and thus, the baseplane image has to be fetched and texture mapped multiple times. The CTK evaluates these tradeoffs and chooses a partitioning scheme depending on the angle such that the overall rendering time is minimized.

Adjusting the thickness of each slab gives another tradeoff between image quality and the rendering speed. Axis-aligned, wide

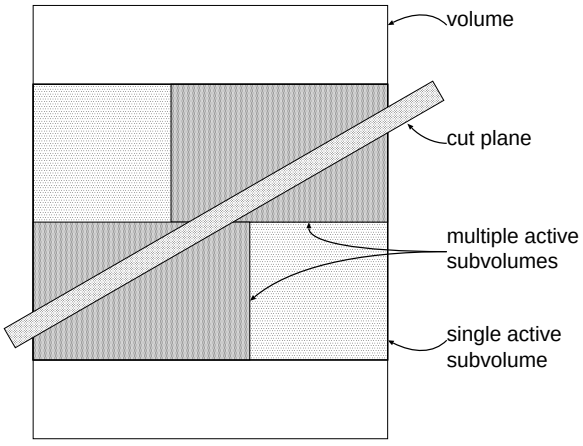

Figure 8: *Using multiple subvolumes to approximate a slab.*

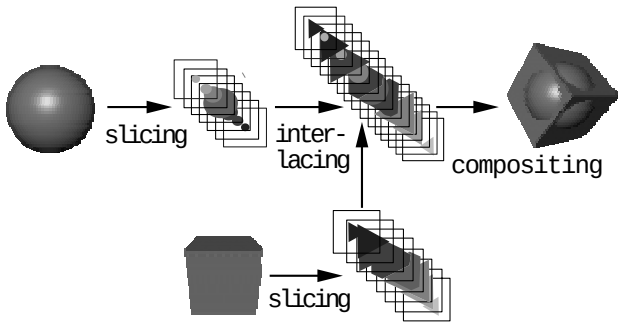

Figure 9: *Rendering overlapping volumes in CTK.*

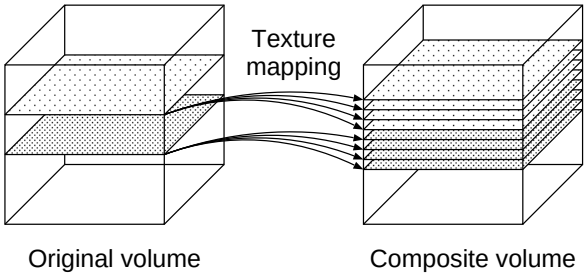

Figure 10: *Texture map each slab image onto multiple geometry planes for compositing.*

slabs achieve the greatest speed, but result in the aforementioned artifacts for diagonal viewing angles. We can alleviate these artifacts without increasing the burden for the Cube-4/VolumePro hardware by texture mapping the slab image onto multiple geometric planes, as depicted in Figure 10. This requires reducing the α values of the computed $RGB\alpha$ slab image. This is done by appropriate modification of the α values of the slab polygon during texture mapping of the polygon. On the geometry processing side, this requires an increased texture mapping fill rate, but not as much as it would seem, because this technique is used only when the slabs are more oblique to the viewer so that the slab images project to smaller areas on the screen. Based on the ultimate frame rate, the CTK evaluates these tradeoffs and determine both the thickness of a slab to render and the number of planes onto which a slab image can be texture mapped.

5 Virtual Colonoscopy Application

Colon cancer is the second leading cause of cancer death in the USA. Previously, optical colonoscopy and barium enema were the only two procedures available for examining the entire colon to detect polyps larger than *5mm* in diameter (i.e., those that are generally malignant). Optical colonoscopy is an invasive procedure which requires intravenous sedation, takes about one hour, and is expensive; barium enema requires a great deal of the patient's physical cooperation, and has a low sensitivity (78%) in detecting polyps in the range of 5*mm* to 20*mm*.

Virtual colonoscopy is a non-invasive computerized medical procedure for examining the entire colon to detect polyps. We have developed an interactive virtual colon navigation system at SUNY Stony Brook [14]. The interactive navigation plots a centerline course from end to end of the colon and defines a potential field. Then, the user's viewpoint is automatically guided along the potential field while avoiding the walls and allowing the user to look and steer toward sites of interest along the way. By using CTK engines, we can automate the guided navigation function into an engine. The engine can take as input a colon volume and the user's steering input and provide output of a camera node. The morphological features of the CtkVolume class are used to help determine the centerline of the colon.

In our original virtual colonoscopy system [14] using conventional graphics hardware, the colon was represented as a polygonal surface. Therefore, physicians could not use it to explore tissues inside a polyp in order to differentiate between benign and malignant structures. A direct volume rendering technique can meet physicians' demand by omitting the intermediate geometric representation and directly mapping certain ranges of sample values of the original volume data to different colors and opacities. Volume rendering provides smoothering rendering and the ability to peer into the colon wall. This "virtual biopsy" technique is superior to surface-based techniques, allowing the physician to confirm possible abnormalities without physical biopsy or surgery.

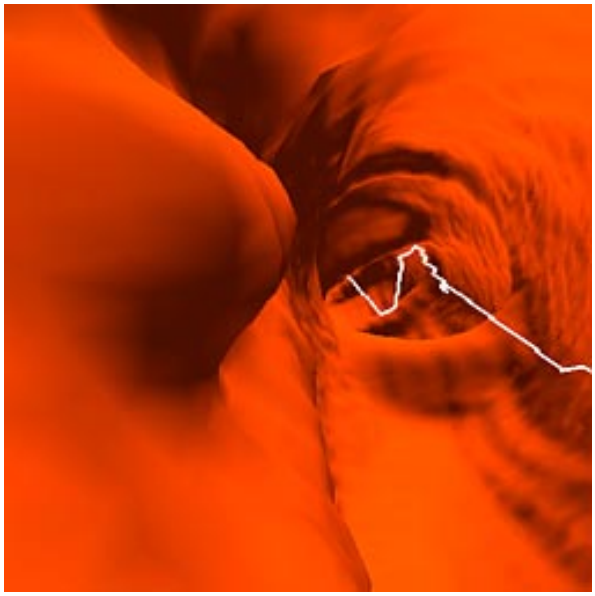

Figure 11: *Polyp (left center) found during navigation in a patient's virtual colon.*

Unfortunately, volume rendering is very expensive, especially for high resolution colon datasets. Further efforts have been dedicated to accelerate software volume rendering using a depth buffer to skip the empty space inside the colon [**?**]. The depth buffer was obtained from reading the z-buffer after the surface rendering. However, the performance achieved still did not meet the interactive speed requirement. Generating a frame of a patient colon takes an average of 1–2 seconds on an SGI Challenge using nine processors. Therefore, volume rendering is used only to supplement the surface rendering as a detailed study and analysis of the sub-surface tissues.

We wish to utilize Cube-4/VolumePro and the CTK to accelerate our virtual colon navigation. However, direct usage of Cube-4/VolumePro will not meet our interactive colon navigation requirement. The patient colon data is usually too large (e.g., 512^3 resolution, to be rendered interactively). In addition, perspective projection is critical for interior colon navigation, but Cube-4/VolumePro currently supports only parallel projection.

Here we present a group of techniques to allow interactive navigation inside a large human colon dataset with perspective projection. We achieve this by taking advantage of the twisted nature of the colon. As a preprocessing step, we first approximate the colon shape with a sequence of piecewise boxes. Then, at a particular view position, we use a scheme called *window closing* to detect the visible boxes within the current viewing frustum so that we only send subvolumes defined by these visible boxes to Cube-4/VolumePro for rendering. To perform perspective rendering, we use the CTK as discussed in Section 4.1.

5.1 Piecewise Boxing of the Colon

We use a sequence of connected boxes to encapsulate the colon, where every box defines a subvolume of the colon volume, as shown in Figure 12a. We generate boxes such that each box intersects the colon on no more than two faces. For each intersecting face, we further approximate the exact intersection with a rectangle and call it a *portal*. Consequently, there are two portals for each

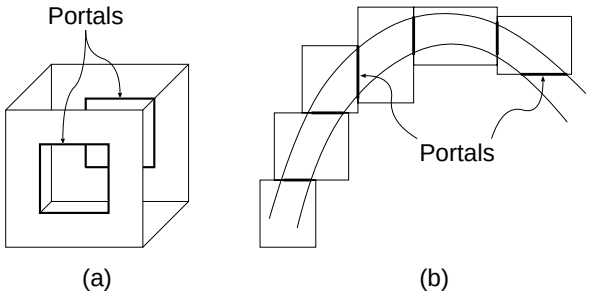

Figure 12: *Approximating a twisted colon structure with a sequence of boxes.*

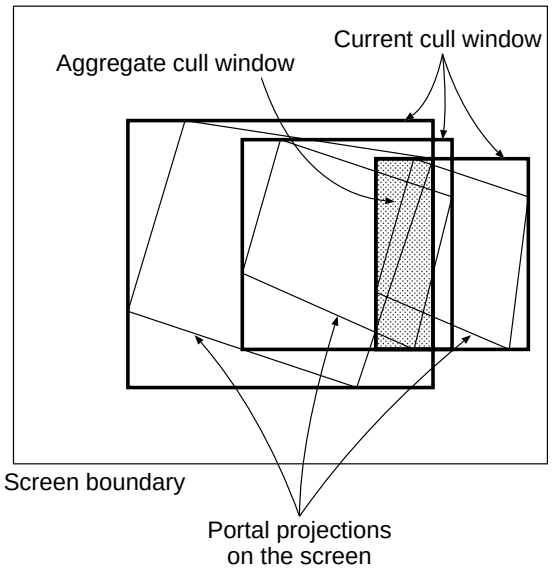

Figure 13: *Visibility detection through window closing algorithm.*

box. Figure 12b shows a top view of the piecewise boxing of a colon. All these boxes are on the 32-voxel boundary of the volume as required by Cube-4/VolumePro subvolume rendering. The darkened segments indicate portals of the boxes.

5.2 Window Closing Visibility Detection

Once we have generated these boxes, we can quickly compute during navigation which boxes are in the current viewing frustum. We propose the Window-Closing algorithm for such visibility detection. The algorithm begins from the closest box to the view point, and computes an axis-aligned rectangular window, called the *aggregate cull window* (ACW), which defines the window through which we can see other boxes behind it. If the ACW has not degenerated yet, we add in the next adjacent box and update the ACW. We continue this procedure until ACW is degenerated. Figure 13 illustrates the procedure.

We project the vertices of the initial portal onto the screen and compute the 2D axis-aligned bounding rectangle of the projected vertices. This bounding rectangle, called the *current cull window* (CCW), is used as the initial ACW. The CCW represents a conservative bound of the portal for efficient computation of the intersection between the ACW and CCW. When a new box is added in, we update the ACW by computing the intersection of the CCW of the added box and the previous ACW. Figure 13 shows the final ACW after projecting three boxes.

There are two ways to decrease the ACW size more quickly so that the rendering terminates earlier. One technique is to generate more portals inside each box to better capture the shape of the colon. However, this can increase the expense of ACW evaluation because we have to project more portals. Another technique is to check the framebuffer image of each subvolume inside the ACW; if it contains rows of opaque pixels, we can shrink the ACW accordingly. The expense of this operation is the access to the framebuffer. If we perform this operation for every subvolume, it can be very expensive. Therefore, we only perform additional pixel checking when the ACW size is smaller than a certain threshold. These two methods are dynamically combined for the most effective ACW reduction.

5.3 Image-Based Rendering

To guarantee interactivity, we provide an image-based rendering scheme to accelerate the rendering. We cache the slab images and reuse them for the generation of novel frames [9], similar to [6]. To decide whether to re-render or reuse the images of the subvolumes, we need to define and compute a priority value of each subvolume. Since our eyes attend to the near object more than the distant object, the priority of each subvolume decreases as it moves further away from the view point. For the near subvolumes, we re-render them in every frame; for the distant subvolumes, we can cache their slab images and reuse them for new frame generation. For the newly visible subvolume, we have to render it because there are no slab images available for it.

In addition, there is another related optimization we can exploit. For each subvolume that we need to re-render, we can specify a different rendering quality so that the CTK chooses the proper thickness of the slab to render, as discussed in Section 4.4. The rendering quality of a subvolume is specified according to its assigned priority value.

6 Conclusions

The forthcoming availability of commodity volume visualization hardware such as Cube-4/VolumePro, is ushering in a new set of visualization applications. Such affordable interactivity enables practical, clinical use of visualization which was previously unavailable. For example, direct visualization of CT data with interactive viewpoint modification and classification are now possible on a standard PC. Our challenge is to create an efficient means to effectively harness the new found power in order to create visualizations that foster greater understanding.

Our Cube Toolkit (CTK) provides a convenient means to specify and interact with dynamic volumetric (and intermixed geometric) scenes. It provides flexible processing engines to allow interesting applications such as radiation treatment planning, virtual colonoscopy, and flight simulation. We described some novel algorithms for approximating perspective volume projection using parallel volume rendering hardware and mixing overlapping volumes or geometry.

We also discussed a new algorithm for hardware volume rendering of tubular colon structures using a series of piecewise boxes with intelligent visibility determination. Finally, we have shown how the CTK can be applied to generate the practical and lifesaving application of virtual colonoscopy.

7 Acknowledgments

This work was supported by NSF grant MIP9527694, ONR grant N000149710402, NIH grant CA79180, and Mitsubishi Electric.

References

[1] *Symposium on Volume Visualization*, Oct. 1996.

[2] ACM. *Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, Aug. 1997.

[3] ACM. *Proceedings of the 1998 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, Aug. 1998.

[4] R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang. Volvis: A diversified volume visualization system. *IEEE Visualization 94*, pages 31–38, Oct. 1994.

[5] I. Bitter and A. Kaufman. A Ray-Slice-Sweep Volume Rendering Engine. In *Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware* [2], pages 121–130.

[6] M. Brady, K. Jung, H. Nguyen, and T. Nguyen. Two-Phase Perspective Ray Casting for Interactive Volume Navigation. In *Proceedings of Visualization '97*, pages 183–189, Oct. 1997.

[7] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Symposium on Volume Visualization*, pages 91–98, Oct. 1994.

[8] G. Cameron and P. Underill. Rendering volumetric medical image data on a simd-architecture computer. *Eurographics Workshop on Graphics Hardware 92*, pages 135–145, Sept. 1992.

[9] B. Chen, Q. Tang, D. Xu, and A. Kaufman. Accelerating volume rendering using image-based rendering. Technical Report TR.99.03.30, State University of New York at Stony Brook, Computer Science Department, Stony Brook, NY 11794-4400, Mar. 1999.

[10] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman. High-Quality Volume Rendering Using Texture Mapping Hardware. In *Proceedings of the 1998 SIGGRAPH/Eurographics Workshop on Graphics Hardware* [3], pages 69–76.

[11] G. Eckel. *OpenGL Volumizer Programmer's Guide*. Silicon Graphics, Inc., 1998.

[12] A. V. Gelder and K. Kim. Direct Volume Rendering with Shading via Three-Dimensional Textures. In *Symposium on Volume Visualization* [1], pages 23–30.

[13] T. He, L. Hong, A. Kaufman, and H. Pfister. Generation of transfer functions with stochastic search techniques. In *IEEE Visualization '96 Conference Proceedings*, Oct. 1996.

[14] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual voyage: Interactive navigation in the human colon. In *Computer Graphics, SIGGRAPH 97*, pages 27–34, Aug. 1997.

[15] U. Kanus, M. Meißner, W. Straßer, H. Pfister, A. Kaufman, R. Amerson, R. J. Carter, B. Culbertson, P. Kuekes, and G. Snider. Implementations of Cube-4 on the teramac custom computing machine. *Computers & Graphics*, 21(2):199–208, 1997.

[16] A. Kaufman and R. Bakalash. Memory and Processing Architecture for 3D Voxel-Based Imagery. *IEEE Computer Graphics and Applications*, 8(6):10–23, Nov. 1988.

[17] K. Kreeger, I. Bitter, F. Dachille, B. Chen, and A. Kaufman. Adaptive Perspective Ray Casting. In *IEEE Symposium on Volume Visualization*, pages 55–62, Oct. 1998.

[18] K. A. Kreeger and A. Kaufman. Hybrid volume and polygon rendering with cube hardware. In *Proceedings of the 1999 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, Aug. 1999. accepted.

[19] K. A. Kreeger and A. Kaufman. Mixing translucent polygons with volumes. In *IEEE Visualization '99 Conference Proceedings*, 1999. accepted.

[20] P. Lacroute. Analysis of a Parallel Volume Rendering System Based on the Shear-Warp Factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):218–231, Sept. 1996.

[21] B. Lichtenbelt, R. Crane, and S. Naqvi. *Introduction to Volume Rendering*. Prentice Hall PTR, 1998.

[22] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Computer Graphics, SIGGRAPH 97*, pages 389–400, Aug. 1997.

[23] M. Meissner, U. Kanus, and W. Strasser. VIZARD II, A PCI-Card for Real-Time Volume Rendering. In *Proceedings of the 1998 SIGGRAPH/Eurographics Workshop on Graphics Hardware* [3], pages 61–68.

[24] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami. EM-Cube: An Architecture for Low-Cost Real-Time Volume Rendering. In *Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware* [2], pages 131–138.

[25] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volumepro real-time ray-casting system. *to appear in Proceedings of SIGGRAPH 1999*, Aug. 1999.

[26] H. Pfister and A. Kaufman. Cube-4 - A Scalable Architecture for Real-Time Volume Visualization. In *Symposium on Volume Visualization* [1], pages 47–54.

[27] P. Schroder and G. Stoll. Data parallel volume rendering as line drawing. *Workshop on Volume Visualization*, pages 25–31, 1992.

[28] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics*. Prentice Hall, 1996.

[29] S. You, L. Hong, M. Wan, K. Junyaprasert, A. Kaufman, S. Muraki, Y. Zhou, M. Wax, and Z. Liang. Interactive volume rendering for virtual colonoscopy. In *Proceedings of Visualization '97*, pages 433–346, Oct. 1997.

[30] Q. Zhu, Y. Chen, and A. Kaufman. Real-time Biomechanically-based Muscle Volume Deformation using FEM. In *Eurographics '98*, pages 275–284, Sept. 1998.