Interactive Volume Rendering of Large Datasets using the Silicon Graphics Onyx4 Visualization System

Christiaan Gribble, Steven Parker, & Charles Hansen

UUCS-04-003

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

January 27, 2004

Abstract

Many recent approaches to interactive volume rendering have focused on leveraging the power of commodity graphics hardware. Though currently limited to relatively small datasets, these approaches have been overwhelmingly successful. As the size of volumetric datasets continues to grow at a rapid pace, the need for scalable systems capable of interactively visualizing large datasets has emerged. In an attempt to address this need, SGI, Inc. has introduced the *Silicon Graphics Onyx4* family of visualization systems. We present the results of our preliminary investigation into the utility of an 8-pipe *Onyx4* system for interactive volume rendering of large datasets. By rendering the image in parallel using an application called Rhesus, we find that the *Onyx4* provides reasonable interactivity for datasets that consume as much as 512 MB of texture memory.

Interactive Volume Rendering of Large Datasets using the Silicon Graphics Onyx4 Visualization System

Christiaan Gribble Steven Parker Charles Hansen

Scientific Computing and Imaging Institute, University of Utah
{cgribble|sparker|hansen}@sci.utah.edu

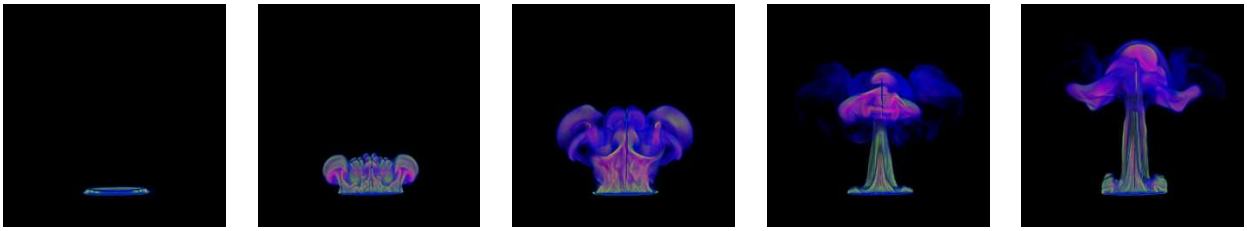


Figure 1: Visualizations of the temperature and velocity magnitude fields from a heptane pool fire simulation as they evolve through time. These images were created using the scalar value, gradient, and Hessian components of the fire-spread datasets. Each time-step consumes 512 MB of texture memory and was rendered in parallel at 0–8 frames per second on an 8-pipe *Onyx4* system.

Abstract

Many recent approaches to interactive volume rendering have focused on leveraging the power of commodity graphics hardware. Though currently limited to relatively small datasets, these approaches have been overwhelmingly successful. As the size of volumetric datasets continues to grow at a rapid pace, the need for scalable systems capable of interactively visualizing large datasets has emerged. In an attempt to address this need, SGI, Inc. has introduced the *Silicon Graphics Onyx4* family of visualization systems. We present the results of our preliminary investigation into the utility of an 8-pipe *Onyx4* system for interactive volume rendering of large datasets. By rendering the image in parallel using an application called Rhesus, we find that the *Onyx4* provides reasonable interactivity for datasets that consume as much as 512 MB of texture memory.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics I.3.7 [Computing Methodologies]: Computer Graphics—3D Graphics

Keywords: Interactive volume rendering, parallel volume rendering, large datasets, visualization systems, Onyx4

1 Introduction

Volume rendering is a visualization technique that maps scalar values of a volumetric dataset to optical properties, typically color and opacity, that can be used to render an image. The mapped quantity can be the original data value contained within the scalar field, or it may be some other value that is derived from the field or even measured using a neighborhood of sampled values within the field. Common examples of derived quantities include the gradient (first-order differential) and the Hessian (second-order differential), though other quantities are certainly possible.

Unlike isosurfacing and other common visualization techniques, volume rendering neither requires nor creates an intermediate geometric representation of the dataset for rendering. The scalar values are resampled, derived, or measured and then mapped to optical properties via a transfer function as the image is being rendered.

In addition, volume rendering enables the simultaneous visualization of multiple scalar fields, that is, multiple sampled, derived, or measured values. Multiple data fields effectively place the ranges of data values representing different features at different locations in a multi-dimensional data space, allowing better classification of the data. These classification abilities lead to better visualizations, which in turn lead to a better understanding of the data.

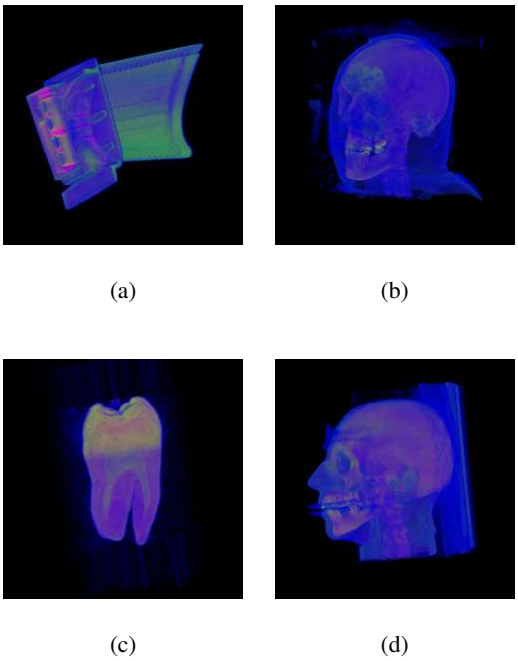
Many recent approaches to interactive volume rendering have focused on leveraging the power of commodity graphics processing units (GPUs). The increasing programmability of these chips means that they are capable of performing more general computations than the basic 3D graphics operations for which they are designed. This programmability, when combined with the fast processor and memory speeds, makes commodity graphics hardware useful for a variety of applications, and GPUs can now be used as general coprocessors.

Implementing volume rendering using GPUs requires extensive use of the texture mapping operations that these chips provide. These operations quickly resample a discrete grid of texels to obtain values at locations that do not lie on the original grid. The hardware alpha blending operations are then used to composite individual samples to determine the values that will be mapped via the transfer function.

More specifically, the volumetric dataset is stored in the graphics hardware's texture memory, either as a single 3D texture or as a stack of 2D textures. Then locations at which the dataset will be resampled are determined by rendering a proxy geometry with interpolated coordinates, typically texture-mapped quadrilaterals. Finally, the resulting texture-mapped geometry is composited from back to front using alpha blending. Typical results of these texture-based techniques are shown in Figure 2.

An obvious constraint on the utility of texture-based volume rendering techniques is the size of the GPU's locally available memory. Commodity graphics hardware currently features memory sizes ranging from 64 MB to 256 MB.

Today's medical, industrial, and scientific datasets already con-



(a) (b)

(c) (d)

Figure 2: *Texture-based volume rendering*—Visualizations of several small datasets: (a) a turbine blade, (b) the Visible Male head, (c) a human tooth, and (d) the Chapel Hill head CT scan. Each of these datasets is easily accommodated by the available texture memory of a single *Onyx4* pipe, and can be rendered at up to 8 frames per second.

sume many hundreds of megabytes of storage. As data acquisition devices are becoming more accurate and as physical simulations are becoming more complex, these already large volumetric datasets are growing larger at a rapid pace. Moreover, multi-field visualizations require even more texture memory than simple single field visualizations. Maintaining interactive frame rates when rendering these large datasets can be quite difficult.

Current commodity graphics hardware cannot store these large datasets in their local memories. While texture-swapping (a technique similar to virtual memory paging in operating systems) can alleviate this problem, it imposes a significant performance penalty when more than a few texture swaps are required per frame. The interactivity provided by the texture-based techniques is therefore greatly reduced. Hence, there is a need for scalable systems that can interactively visualize large datasets.

In an attempt to address this need, SGI, Inc. has introduced the *Silicon Graphics Onyx4* family of visualization systems. The *Onyx4* combines the advantages of the *SGI NUMAflex architecture*, namely high-bandwidth and scalability, with the power of commodity graphics hardware.

In the pages that follow, we present the results of our preliminary investigation into the utility of the *Onyx4* for parallel volume rendering. We begin with a high-level description of Rhesus, the texture-based volume rendering application used in this investigation. Then, we scrutinize Rhesus' performance on the *Onyx4* and clearly identify the factors that limit the attainable frame rate. We conclude by outlining possible optimizations, as well as more general research directions, that could be considered for the *Silicon Graphics Onyx4* platform.

2 Parallel Volume Rendering with Rhesus

One obvious approach to visualizing large datasets interactively involves partitioning the data into several subvolumes and rendering those chunks in parallel. This approach is taken by Rhesus, the texture-based volume rendering application used in this investigation.

Rhesus is derived from a texture-based volume renderer called Simian, which uses multi-dimensional transfer functions to visualize multi-field datasets. In addition, Simian employs direct manipulation widgets for defining the transfer function; for probing, clipping, and classifying the data; and for shading and coloring the resulting visualization.

Originally, Rhesus was targeted for a cluster of commodity computers equipped with nVidia GeForce FX 5900 graphics hardware; this cluster is called *Nebula*. Using the Message Passing Interface (MPI) standard, the parallel processes load and render a specific subvolume of the dataset, and then combine the individual results using the binary-swap compositing algorithm. This process is illustrated for two large datasets in Figure 8.

More specifically, the rendering pipeline employed by Rhesus has five stages: (1) volume rendering, (2) draw order determination, (3) framebuffer readback, (4) pixel swapping and compositing, and (5) image display. (Only the display process executes stage 5.) This pipeline is depicted in Figure 3. In stage 1, each process renders its subvolume using the basic strategy described in Section 1. Rendering is followed by a round of MPI-based communication in stage 2, which determines the proper draw order (that is, the order in which regions of pixels will be swapped between nodes and composited in stage 4). This order is based on the current frame's viewpoint. Stage 3 is simply the framebuffer readback: each process obtains its image from the hardware framebuffer and stores that image in main memory. Then, in stage 4, regions of pixels are swapped and composited according to the order determined earlier. This stage invokes (possibly) several rounds of MPI communication, at the end of which the display process will have the fully composited image stored in its main memory. Finally, that image is drawn to the hardware framebuffer in stage 5, and the pipeline repeats.

Using the extensions to MPI programs (MPE) library, we are able to measure the time required by each stage of the Rhesus pipeline. An example of the results are shown in Figure 4.

For this evaluation, we have ported Rhesus to an 8-pipe *Onyx4* system. In contrast to the *Nebula* cluster, the *Onyx4* is a shared memory multiprocessor. Moreover, the new system employs ATI Fire GL X1 graphics hardware. (Further details of the *Onyx4*'s hardware and software environment are listed in Table 1.) To promote portability between *Nebula* and the *Onyx4*, we chose not to utilize the globally available shared memory; Rhesus is still implemented using MPI (in particular, SGI's native implementation). This choice is justified in Section 3, where it is shown that MPI communication is not a limiting factor in Rhesus' performance on the *Onyx4*. In order to utilize the capabilities of the ATI graphics hardware, however, changes to the volume rendering code were required. Specifically, ATI fragment programs are now used to render the volume, rather than nVidia register combiners.

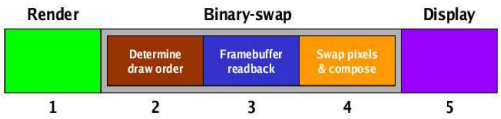


Figure 3: *Rhesus rendering pipeline*—The rendering pipeline has five stages: volume rendering, draw order determination, framebuffer readback, pixel swapping and compositing, and image display.

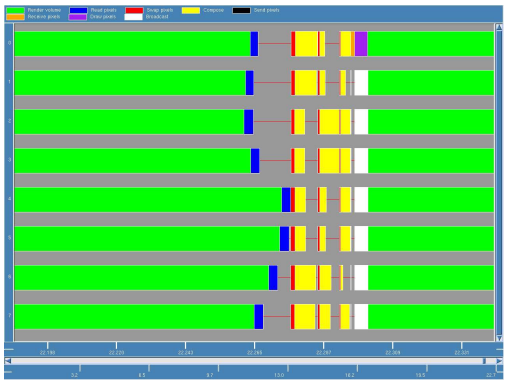


Figure 4: *Pipeline stage timings*—A close-up view of the distribution of time spent rendering a typical frame using 8 Rhesus processes. Here, the green events correspond to stages 1 and 2 of the rendering pipeline; the blue to stage 3; the red, yellow, and orange to stage 4; and, finally, the purple to stage 5.

Component or Subsystem	Characteristics
CPU	16 600 MHz IP35 processors 32 KB L1 I-cache (per processor) 32 KB L1 D-cache (per processor) 4 MB unified L2 cache (per processor) MIPS R16000 processor chip revision 2.1 MIPS R16010 FP chip revision 2.1
Main memory	8192 MB, shared ccNUMA interconnect
3D Graphics	8 ATI Fire GL X1 graphics accelerators 128 MB memory (each pipe) ATI FireGL drivers OpenGL version 1.3 Irix 6.5 GLX version 1.3 Irix 6.5 GLU version 1.3 Irix 6.5

Table 1: *The* Onyx4 *visualization system*—The *Onyx4* combines the *SGI NUMAflex* architecture and commodity graphics hardware. Here, we list the most pertinent hardware and software characteristics of the *Onyx4* system used in this evaluation.

Rhesus is currently a work-in-progress. While it does not yet support all of the features of Simian, Rhesus is nevertheless a valuable research tool for interactively navigating and exploring large datasets. In the remaining sections, we characterize its performance on the 8-pipe *Onyx4* system described above.

3 Results

To understand the performance of the *Onyx4* when rendering large datasets, we began by exploring the scalability characteristics of the system using synthetic datasets. The results for 512x512 images are shown in Figure 5. As the total size of the dataset increases from 64 MB (one pipe) to 512 MB (eight pipes), the volume rendering time consumed by each pipe remains nearly constant. However, the time required to complete the pixel composition stage increases significantly and begins to dominate the other stages of the binary-swap algorithm.

We have also used Rhesus to visualize several large VGH datasets, including those shown in Figures 1, 6, and 8. In particular, we report the performance characteristics observed when visualizing the Visible Woman head dataset depicted in Figure 6. This dataset consumes 512 MB of texture memory and must be partitioned into eight 64 MB chunks. The subvolumes are rendered in parallel using eight Rhesus processes.

Figure 7a depicts the pipeline timings obtained with MPE when rendering 256x256 images of the Visible Woman head dataset. Here, too, the rendering stage consumes the majority of the image generation time and is nearly identical across all 8 pipes.

The timing results depicted in Figure 7a led to a more careful examination of the time consumed by the rendering, readback, and pixel compositing stages. Using a technique we term "peeling the onion", we measured the impact of each stage of the Rhesus pipeline on the overall frame rate achieved for 256x256 and 512x512 images. Limiting the number of pipeline stages that are completed exposes the marginal impact of each stage on the attainable frame rate.

The results of peeling the onion are depicted in Figure 7b. Again, the compositing stage appears to be a good candidate for further optimization.

Finally, we measured the speedup achieved by removing individual calls from the Rhesus pipeline. In particular, we simply removed the volume rendering call first, then the readback call, and finally, the pixel compositing call, leaving the rest of the Rhesus pipeline intact and allowing it to run to completion. The results of this experiment are given in Table 2. Rendering the data clearly has the most significant impact, but these results also suggest that optimizing the compositing stage will result in improved frame rates.

4 Future Work

Overlapping Rhesus' rendering and binary-swap compositing phases using separate processes or threads is the most obvious application-level optimization. The benefits of this optimization include improved performance and better utilization of the *Onyx4*'s 16-way architecture.

Additional performance gains may be possible by taking advantage of the system's global shared memory. However, two factors must be considered before expending the effort to modify Rhesus'

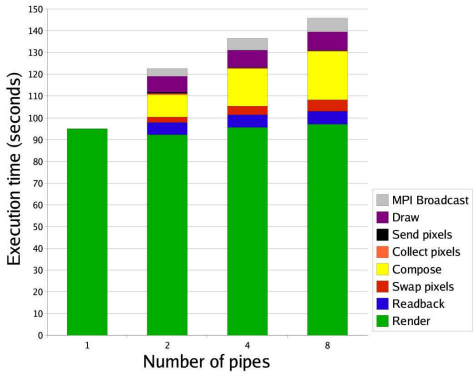


Figure 5: *Rhesus scalability*—Time consumed by the various stages of the rendering pipeline when testing the scalability of the system. As the total size of the dataset increases, the time required to complete the pixel compositing stage (yellow) begins to dominate the other binary-swap stages.

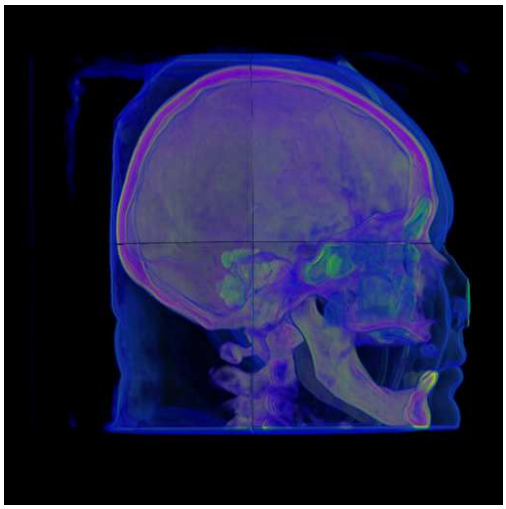


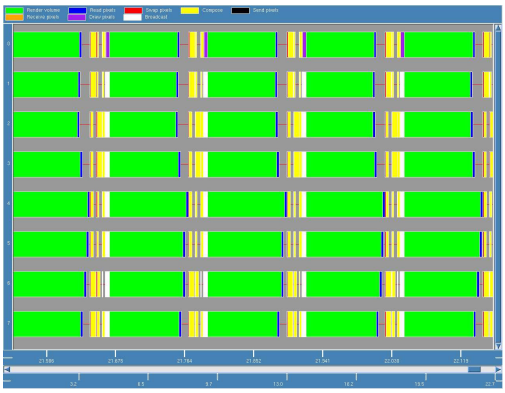
Figure 6: *Rendering large datasets*—A visualization of the Visible Woman head dataset. This VGH dataset consumes 512 MB of texture memory and must be rendered in parallel using all 8 pipes of the *Onyx4*.

design: First, the current performance bottleneck is volume rendering, not MPI communication; and second, the portability of the application between the *Onyx4* and the *Nebula* cluster would be greatly reduced. The current MPI-based implementation offers a highly-portable application with reasonable performance, so any further pursuit of this optimization would first require a shift in the limiting stage of the application from volume rendering to MPI communication.

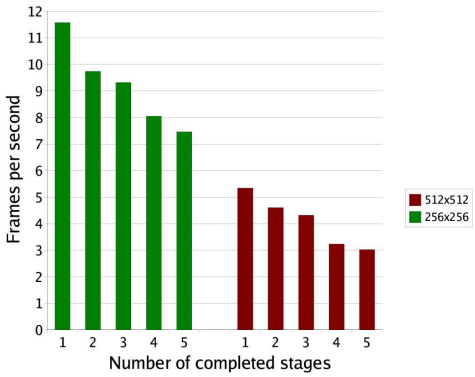
More generally, the *Onyx4* is an excellent platform for examining the the utility of GPUs for general computation. Traditional desktop and workstation platforms are not currently equipped with multiple graphics accelerators, so a parallel application requires moving to a cluster. Unfortunately, clusters can introduce new barriers to creating high-speed GPU-parallel applications, such as the increased latency between communicating threads or processes. The architecture of the *Onyx4* may be able to overcome many of these barriers. In addition, the global shared memory and high-speed interconnect of the *Onyx4* present a unique opportunity for solving large problems in parallel using hybrid CPU/GPU approaches.

Image Size	Removed Call	Frame Rate (fps)	Speedup
256x256	none	7.47	-
	render	50.9	6.82
	composite	9.0	1.21
	readback	8.56	1.15
512x512	none	3.03	-
	render	15.07	4.97
	composite	3.9	1.29
	readback	3.74	1.23

Table 2: *Relative speedup*—Results of removing various function calls from the Rhesus pipeline when visualizing the Visible Woman head dataset. Aside from the noted modifications, the remainder of the pipeline remains intact and runs to completion.



(a)



(b)

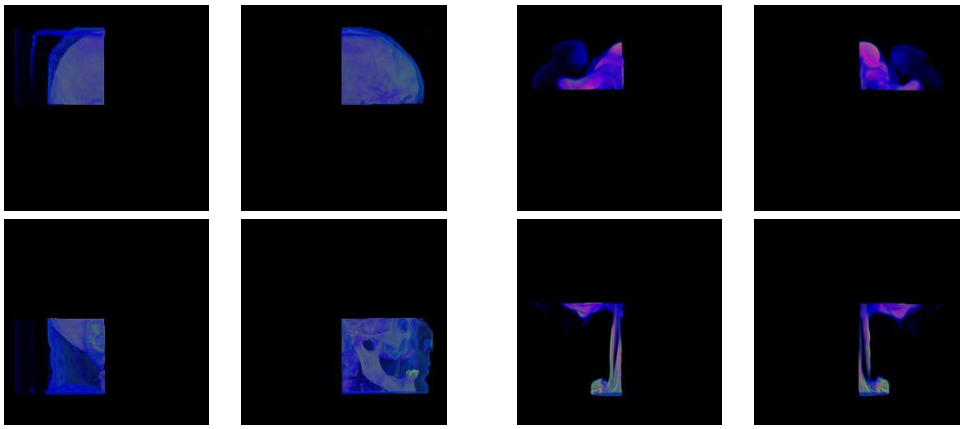
Figure 7: *Parallel pipeline timings*—These images depict (a) the time consumed by various stages of the Rhesus pipeline when rendering a 256x256 image of the Visible Woman head dataset and (b) the results of peeling the onion for 256x256 and 512x512 visualizations of the same dataset.

Acknowledgments

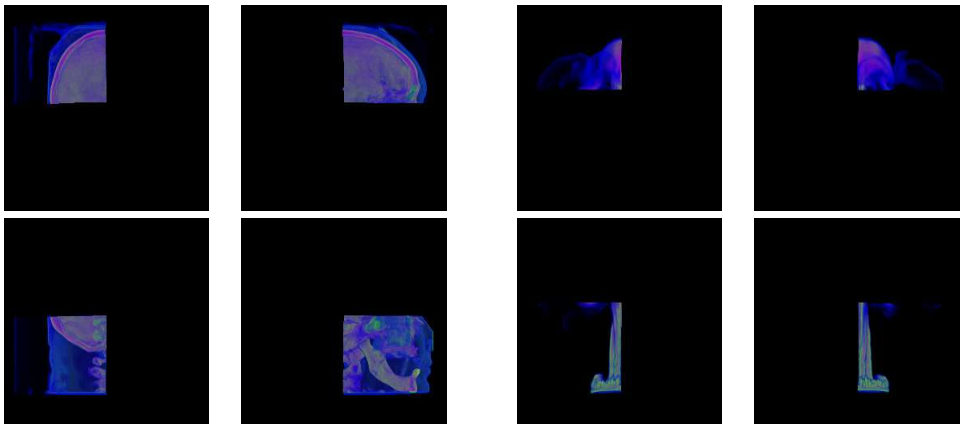
We would like to thank SGI, Inc. for making the *Onyx4* system available to us: Richard Grossen was instrumental in securing the machine for our use, and Mike Matheson provided invaluable technical assistance along the way.

Several members of the SCI Institute also deserve thanks: Greg Jones was helpful as a liaison to SGI, and Joe Kniss and Milan Ikits offered much appreciated guidance during the effort to port Rhesus to the *Onyx4*.

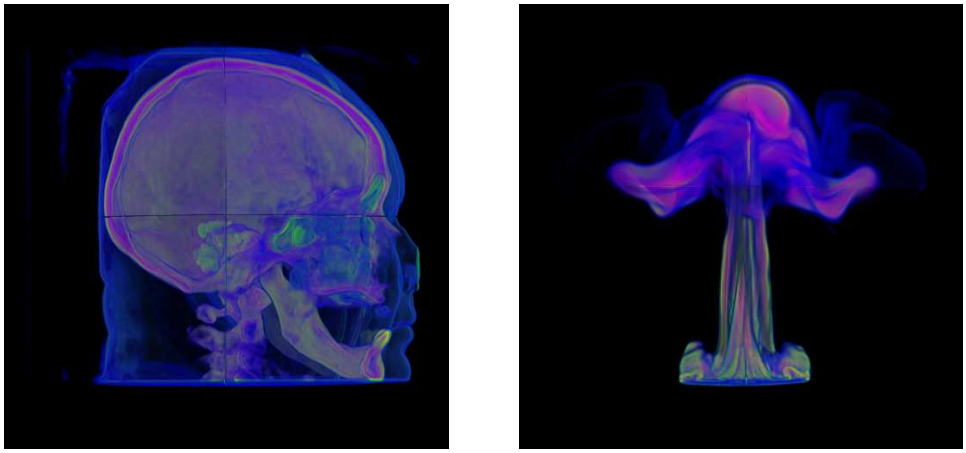
This work was funded by grants from the Department of Energy (VIEWS 0F00584), the National Science Foundation (ASC 8920219, MRI 9977218, ACR 9978099), and the National Institutes of Health National Center for Research Resources (1P41RR12553-2).



(a) Front subvolumes



(b) Back subvolumes



(c) Composited results

Figure 8: *Parallel volume rendering with Rhesus*—These images show the eight subvolumes of both the Visible Woman head (left column) and heptane pool fire (right column) datasets, as well as the resulting visualizations. The subvolumes in (a) are closer to the current viewpoint, while those in (b) are farther away. The final visualizations in (c) were created by applying the binary-swap compositing algorithm to the eight images in (a) and (b).