Cluster-Based Interactive Volume Rendering with Simian

Christiaan Gribble, Xavier Cavin, Mark Hartner, & Charles Hansen

UUCS-03-017

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

September 3, 2003

Abstract

Commodity-based computer clusters offer a cost-effective alternative to traditional large-scale, tightly coupled computers as a means to provide high-performance computational and visualization services. The Center for the Simulation of Accidental Fires and Explosions (C-SAFE) at the University of Utah employs such a cluster, and we have begun to experiment with cluster-based visualization services. In particular, we seek to develop an interactive volume rendering tool for navigating and visualizing large-scale scientific datasets. Using Simian, an OpenGL volume renderer, we examine two approaches to cluster-based interactive volume rendering: (1) a "cluster-aware" version of the application that makes explicit use of remote nodes through a message-passing interface, and (2) the unmodified application running atop the Chromium clustered rendering framework. This paper provides a detailed comparison of the two approaches by carefully considering the key issues that arise when parallelizing Simian. These issues include the richness of user interaction; the distribution of volumetric datasets and proxy geometry; and the degree of interactivity provided by the image rendering and compositing schemes. The results of each approach when visualizing two large-scale C-SAFE datasets are given, and we discuss the relative advantages and disadvantages that were considered when developing our cluster-based interactive volume rendering application.

Cluster-Based Interactive Volume Rendering with Simian

Christiaan Gribble
SCI Institute
University of Utah

Xavier Cavin
Project Isa
Inria Lorraine

Mark Hartner
SCI Institute
University of Utah

Charles Hansen
SCI Institute
University of Utah

ABSTRACT

Commodity-based computer clusters offer a cost-effective alternative to traditional large-scale, tightly coupled computers as a means to provide high-performance computational and visualization services. The Center for the Simulation of Accidental Fires and Explosions (C-SAFE) at the University of Utah employs such a cluster, and we have begun to experiment with cluster-based visualization services. In particular, we seek to develop an interactive volume rendering tool for navigating and visualizing large-scale scientific datasets. Using Simian, an OpenGL volume renderer, we examine two approaches to cluster-based interactive volume rendering: (1) a "cluster-aware" version of the application that makes explicit use of remote nodes through a message-passing interface, and (2) the unmodified application running atop the Chromium clustered rendering framework. This paper provides a detailed comparison of the two approaches by carefully considering the key issues that arise when parallelizing Simian. These issues include the richness of user interaction; the distribution of volumetric datasets and proxy geometry; and the degree of interactivity provided by the image rendering and compositing schemes. The results of each approach when visualizing two large-scale C-SAFE datasets are given, and we discuss the relative advantages and disadvantages that were considered when developing our cluster-based interactive volume rendering application.

CR Categories: I.3.7 [Computing Methodologies]: Computer Graphics—3D Graphics;

Keywords: Cluster-based visualization, interactive volume rendering

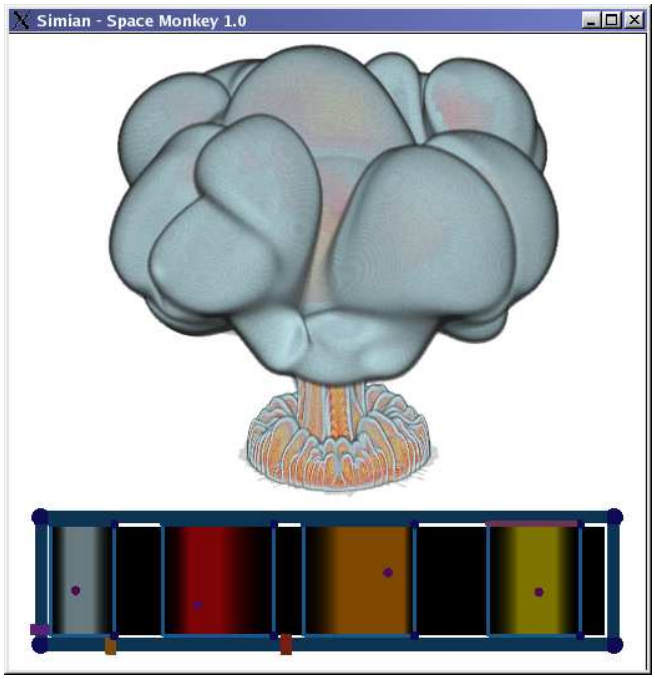

Figure 1: *Interactive volume rendering with Simian*—Simian is a scientific visualization tool that utilizes the texture processing capabilities of consumer graphics accelerators to produce direct volume rendered images of scientific datasets. It provides a rich interface that allows users to interact with volumetric datasets and explore interesting features in real-time.

1 INTRODUCTION

Over the past several years, explosive growth in the performance of consumer computing hardware has lead to relatively inexpensive central processing units boasting clock speeds in the multi-gigahertz range. Similarly staggering growth has been demonstrated by consumer graphics hardware, which now offer programmable graphics processing units capable of rasterizing millions of triangles per second. These advancements have resulted in affordable low-end desktop computers that rival traditional workstation-class systems. Moreover, interconnection hardware operating in the gigabit-per-second range continues to become more affordable and more widely available. As a result, commodity-based computer clusters now offer a cost-effective alternative to traditional large-scale, tightly coupled computers as a means to provide high-performance computational and visualization services.

The Center for the Simulation of Accidental Fires and Explosions (C-SAFE) at the University of Utah employs a commodity-based cluster, and we have begun to experiment with cluster-based visualization services. In particular, we seek to develop an interactive volume rendering tool for navigating and visualizing large-scale scientific datasets. Using Simian, the OpenGL volume renderer shown in Figure 1, we examine two approaches to cluster-based interactive volume rendering: (1) a "cluster-aware" version of the application that makes explicit use of remote nodes through a message-passing interface (MPI), and (2) the unmodified application running atop the Chromium clustered rendering framework.

Section 2 provides background, including a brief introduction to the C-SAFE fire-spread simulations in Section 2.1, the clustered computing environment of C-SAFE in Section 2.2, direct volume rendering with Simian in Section 2.3, and the Chromium clustered rendering framework in Section 2.4. We then provide a detailed comparison of the two approaches. In this comparison, which is found in Section 3, we carefully consider the key features and open issues of parallelizing Simian with each approach. This comparison includes an examination of the richness of user interaction; the distribution of volumetric datasets and proxy geometry; and the degree of interactivity provided by the image rendering and compositing schemes. Section 4 presents the results of each approach when visualizing two large-scale C-SAFE fire-spread datasets. In Section 5, we discuss the relative advantages and disadvantages that were considered when developing our cluster-based interactive volume rendering application. Finally, we outline possible future work in Section 6.

Component	Type
Motherboard	Supermicro P4DC6+ (Intel 860 chipset)
CPU	Dual Intel Xeon 1.70GHz (256KB L2 cache)
Memory	2x512MB Corsair ECC RDRAM
Network card	Intel Pro1000/XT
Graphics board	NVIDIA
GPU	NVIDIA GeForce3 NV20 (64MB RAM)
Hard drive	18GB Seagate Cheetah U160 (15000 RPM)

Table 1: Hardware components of a single C-SAFE cluster node

2 BACKGROUND

C-SAFE is an alliance between the University of Utah and the Department of Energy Accelerated Strategic Computing Initiative. The objective of C-SAFE is to develop a system comprised of a problem-solving environment in which fundamental chemistry and engineering physics are combined with non-linear solvers, optimization, computational steering, scientific visualization, and experimental data verification. It focuses specifically on providing state-of-the-art, science-based tools for the numerical simulation of accidental fires and explosions, especially within the context of handling and storage of highly flammable materials.

Specifically, the project focuses on three distinct, sequential steps that parallel the events in our physical problem: Ignition and Fire Spread, Container Dynamics, and High Energy (HE) Transformations. A fire or explosion is initiated by an ignition and, depending upon the magnitude of heat generation and dissipative terms, a perturbation by the ignition source either decays or grows into a flame. This ignition may be followed by a spreading fire and possibly an explosion. The fire or explosion can cause the container of HE material to undergo changes, perhaps rupture, and, simultaneously or sequentially, the HE material itself can undergo transformations that lead to an explosion. These computational steps are integrated into a coupled fire and explosion system.

2.1 C-SAFE Fire-Spread Simulation

The focus of the C-SAFE Fire-Spread team is to develop computational modules that realistically simulate the scenario in which a missile, located within a pool of jet-fuel that catches fire, may explode. The team studies large-scale fires using Large Eddies Simulations (LES) that "evolve" the fire over time. These simulations will help researchers answer important questions like: How quickly do these fires spread? What temperatures do they reach? What kinds of byproducts are produced?

The current LES codes include mixing, reaction, and radiation models. These computational models produce very large, very complex time-dependent datasets that are often difficult to analyze. To aid in the understanding of the fire-spread simulation results, the domain scientists use volume rendering to track the buoyant flow activity from pool fires.

2.2 C-SAFE Clustered Computing Environment

C-SAFE employs a 32-node visualization cluster composed of commodity hardware components that are interconnected with a high-speed network. Using the C-SAFE cluster, we have begun to experiment with interactively volume rendering the C-SAFE fire-spread datasets produced by the LES codes described above.

Component	Device Driver
Network	Intel e1000 4.3.2 (June 2002)
Graphics	GLX and kernel 1.0-2960 (May 2002)
AGP	AGP4x with kernel AGPGART

Table 2: Device drivers used on a single C-SAFE cluster node

The individual nodes of the cluster are comprised of dual Intel Xeon 1.70GHz processors, 1GB of physical memory, an NVIDIA GeForce3 NV20 graphics accelerator with 64MB of texture memory, and an Intel Pro 1000/XT network interface card. The detailed hardware configuration of a single cluster node can be seen in Table 1. The nodes are connected via an Extreme Networks 6816 Black Diamond switch and communicate using Gigabit Ethernet over copper.

Each node runs the Linux kernel (version 2.4.20, November 2002) and the XFree86 X Window System (version 4.2.0, January 2002). Table 2 lists the specific device drivers used for the different hardware components.

2.3 Interactive Volume Rendering with Simian

Simian is a scientific visualization tool that utilizes the texture processing capabilities of consumer graphics accelerators to produce direct volume rendered images of scientific datasets [4]. The true power of Simian is its rich user interface. Simian employs direct manipulation widgets for defining multi-dimensional transfer functions; for probing, clipping, and classifying the data; and for shading and coloring the resulting visualization [5]. As an example, the clipping plane widget is shown in Figure 2.

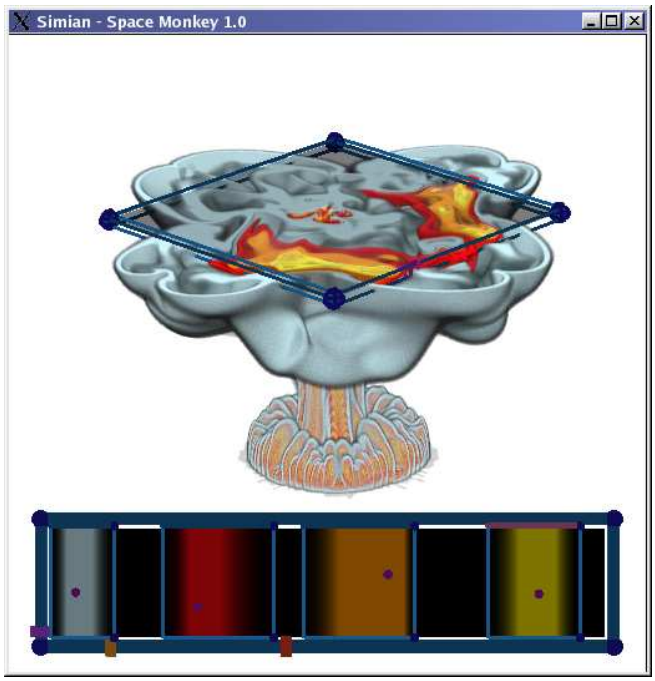

Figure 2: *Simian's rich user interface*—Direct manipulation widgets enable users to identify and explore the interesting features of a dataset interactively. Here, the clipping plane widget reveals the internal structure of a plume from a heptane fire dataset generated by the C-SAFE LES codes.

Direct manipulation widgets are typically constructed from simple geometric objects (such as spheres, cylinders, or cones), and they are rendered as part of the visualization. Each part of a widget controls some parameter of the visualization, and so they offer the user a three-dimensional (3D) interface for setting and controlling these parameters with relative ease [9]. Users are able to quickly identify and explore the most relevant features of their data. The result of this interface is a very powerful interactive visualization tool that assists scientists and engineers in interpreting and analyzing their results.

A complete discussion of direct volume rendering using commodity graphics hardware is given in [1]. For more on using multi-dimensional transfer functions in interactive volume visualization, see [6]. All of the direct manipulation widgets provided by Simian are described thoroughly in [5].

The size of a volumetric dataset that Simian can visualize interactively is largely dependent on the size of the texture memory provided by the local graphics hardware. For typical commodity graphics accelerators, the size of this memory ranges from 32MB to 128MB. However, even small scientific datasets can consume hundreds of megabytes, and these datasets are rapidly growing larger as time progresses. Although Simian provides mechanisms for swapping smaller portions of a large dataset between the available texture memory and the system's main memory (a process that is similar to virtual memory paging), performance drops significantly and interactivity disappears. Moreover, because the size of the texture memory on commodity graphics hardware is not growing as quickly as the size of scientific datasets, using the graphics accelerators of many nodes in a cluster-based system is necessary to interactively visualize large-scale datasets.

Naturally, cluster-based visualization introduces many challenges that are of little or no concern when rendering a dataset on a single node, and there are many techniques for dealing with the problems that arise. Our goal is to create an interactive volume rendering tool that provides a full-featured interface for navigating and visualizing large-scale scientific datasets.

2.4 Chromium Clustered Rendering Framework

Chromium is a system for manipulating streams of OpenGL graphics commands on commodity-based visualization clusters [2]. For Linux-based systems like the C-SAFE cluster, Chromium is implemented as a set of shared-object libraries that export a large subset of the OpenGL application programming interface (API). Extensions for parallel synchronization are also included [3]. Chromium's *crappfaker* libraries operate as the client-side stub in a simple client-server model. The stub intercepts graphics calls made by an OpenGL application, filters them through a user-defined chain of stream processing units (SPUs) that may modify the command stream, and finally redirects the stream to designated rendering servers. The Chromium rendering servers, or *crserver*s, process the OpenGL commands using the locally available graphics hardware, the results of which may be delivered to a tiled display wall, returned to the client for further processing, or composited and displayed using specialized image composition hardware such as Lightning-2 [10]. Depending on the particular system configuration, it is possible to implement a wide variety of parallel rendering systems, including the common sort-first and sort-last architectures [8]. For the details of the Chromium clustered rendering framework, see [2].

In principle, Chromium provides a very simple mechanism for hiding the distributed nature of clustered rendering from OpenGL applications. By simply loading the Chromium libraries rather than the system's native OpenGL implementation, graphics commands can be processed by remote hardware without modifying the calling application. However, for the Simian volume rendering application, Chromium does not currently provide features that sufficiently mask the underlying distributed operation and still enable the application to realize the extended functionality that we seek. OpenGL applications may still require significant modifications to effectively utilize a cluster's resources, even when employing the Chromium framework.

It was necessary to implement some OpenGL functionality that was not supported by Chromium (beta release, April 2002). First, we implemented the subset of OpenGL calls related to 3D textures, including `glTexImage3D`, which is the workhorse of the Simian volume rendering application. In addition, we also added limited support for the NVIDIA `GL_NV_texture_shader` and `GL_NV_texture_shader2` extensions, implementing only those particular features of the extensions that are explicitly used by Simian.

3 Comparison of the Clustered Rendering Approaches

As noted in Section 2, there are a number of techniques for implementing parallel rendering algorithms on a cluster-based visualization system. One obvious solution is to simply modify the application to make explicit use of remote nodes through MPI. Another is to employ a software package such as Chromium to hide the distributed nature of the rendering from the application, potentially allowing it to be used with minimal modification.[1]

In the sections that follow, we compare the relative strengths and weaknesses of these two approaches for our application. We use a cluster-aware version of Simian that is based the beta distribution of Simian (version 1.0) and on the LAM/MPI implementation of MPI. The unmodified application that runs atop Chromium is based on the same distribution of Simian. Both of these applications employ the hardware of the C-SAFE cluster, which was described in Section 2.2.

3.1 Richness of User Interaction

Generating a volume rendered visualization that displays the features of interest is only possible with a good transfer function, so the specification of transfer functions is therefore a very important task. Unfortunately, specifying good transfer functions is also a very difficult task. Kniss, *et al.*, [5], have identified three reasons why this task is so problematic: (1) the transfer function has a large number of degrees of freedom in which users can get lost; (2) typical interfaces for specifying the transfer function are neither constrained nor guided by the datasets being visualized; and (3) transfer functions do not include spatial position as a variable for defining the assignment of color and opacity of dataset values. While multi-dimensional transfer functions, which are the type that Simian uses, alleviate the third issue, they exacerbate the first and second problems.

To address the first two issues, the direct manipulation widgets introduced in [5] link user interaction in one domain with feedback from another, offering insight into the way the domains are connected. This "dual-domain" interaction is the mechanism that is employed by Simian for identifying and exploring the features of interest.

[1] Note that using Chromium to implement the communications layer of a cluster-aware application is also possible; however, it has has not been considered here. For the description and evaluation of a cluster-aware parallel volume rendering system that utilizes Chromium's connection-based networking abstractions, see [2].

Simian's interface is therefore a key feature. The array of tools that Simian provides and ease with which users can employ them to interact with a dataset leads directly to better, more useful visualizations, which in turn leads to better, more useful results. In order for users to navigate and visualize large-scale scientific datasets effectively, a parallel version of the tool must provide the rich and easy-to-use interface of the current version. Simply stated, any parallel implementation of Simian must support direct manipulation widgets.

The success of Simian's dual-domain interaction relies heavily on inter-widget communication. In a clustered environment, this communication can be problematic because different nodes may render the geometric primitives composing a single widget. Furthermore, responding to changes and updates to a given widget's state is more difficult in a clustered environment. Handling the distribution of widget geometry and providing a means to propagate changes in widget state introduce new issues that must be considered carefully in either approach.

Currently, the cluster-aware version of Simian does not attempt to incorporate distributed direct manipulation widgets. While specifying the node or nodes responsible for actually rendering the widgets would not be difficult, the application lacks the appropriate infrastructure for notifying remote widgets of changes or updates to any particular widget's state.

The unmodified Simian running atop Chromium also does not provide for distributed widgets. Although Chromium is capable of automatically distributing the OpenGL commands that render the widgets, the mechanism it provides is not aware that a particular widget's geometric primitives form a cohesive whole. Like the cluster-aware Simian, Chromium is not able to correctly handle changes or updates to a widget's state, and it currently lacks the infrastructure to notify other widgets of such changes.

By making Simian's display node responsible for rendering the widgets and for responding to changes in their state, the interaction of Simian could be restored with either approach. Creating and integrating a distributed events mechanism that can handle inter-widget communication is one option for the cluster-aware version. Similarly, the Chromium Rendering Utility Toolkit, which is currently under development, may support the functionality needed to implement inter-widget communication in the second approach.

3.2 Distribution of Volumetric Data and Proxy Geometry

Currently, Simian handles datasets that are too large to load into the available texture memory using the swapping facility as previously described. Unfortunately, this mechanism for visualizing large-scale datasets drastically reduces performance and noticeably degrades the level of interactivity.

However, by dividing a large dataset into subvolumes that can be distributed across multiple nodes, interactive performance can be restored. This "divide-and-conquer" technique is used by our cluster-aware version of Simian. A given dataset is first decomposed into 2^n subvolumes, and then, because the application is explicitly aware of the remote resources made available in the clustered environment, it distributes the subvolumes among the remote nodes. Each node is responsible for rendering its subvolume(s) using the locally available graphics hardware, and the individual results are combined using a binary-swap compositing algorithm to form the final image. If, by distributing these small subvolumes, texture swapping can be reduced or eliminated, then the cluster-aware version of Simian is able to restore interactive rates when visualizing large-scale datasets.

In contrast, the unmodified version, which remains unaware of remote nodes and the resources they offer, must rely on Chromium

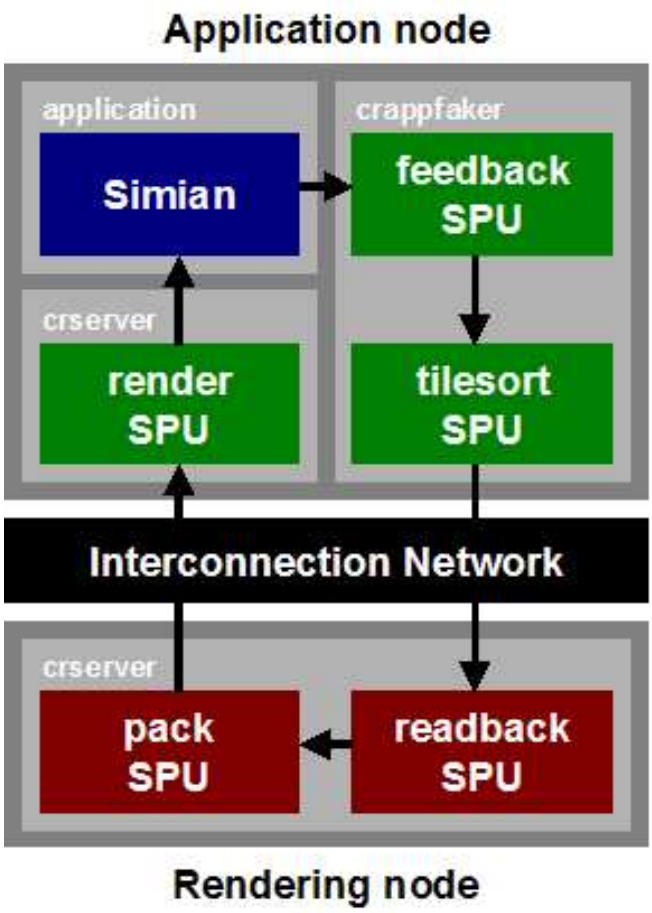

Figure 3: *Chromium SPU configuration*—On the application node, the *crappfaker* library loads the `feedback` SPU, which provides a limited degree of interaction, and the `tilesort` SPU, which distributes OpenGL commands. Downstream, *crserver*s use the `readback` and `pack` SPUs to render and return partial results to the client. Note that, while only one rendering node is shown, all rendering nodes load these SPUs. Finally, the application node's *crserver* uses the `render` SPU to combine the partial results and form the final image.

to handle large datasets. Unfortunately, no SPUs that automatically handle the distribution of subvolumes are currently available.

Chromium is capable of distributing the proxy geometry that Simian uses to render a dataset, however. Using the SPU configuration shown in Figure 3, the rendering nodes are assigned disjoint extents, or tiles, of the total image space. As Chromium intercepts and buffers OpenGL commands, it updates an image space bounding box that surrounds the geometry created by the current set of outstanding commands. Then, when the stream is flushed, this bounding box is tested against each server's tile. If they intersect, the entire command buffer is delivered to that server.

Using Chromium may reduce the amount of proxy geometry that any given server is responsible for rendering, but the tile-sort mechanism does not impact the unmodified application's ability to visualize large-scale datasets interactively. There is no clear gain, either in terms of the maximum dataset size or interactive performance, over using the resources of a single node to produce the visualization. Lacking the ability to decompose a large-scale dataset into

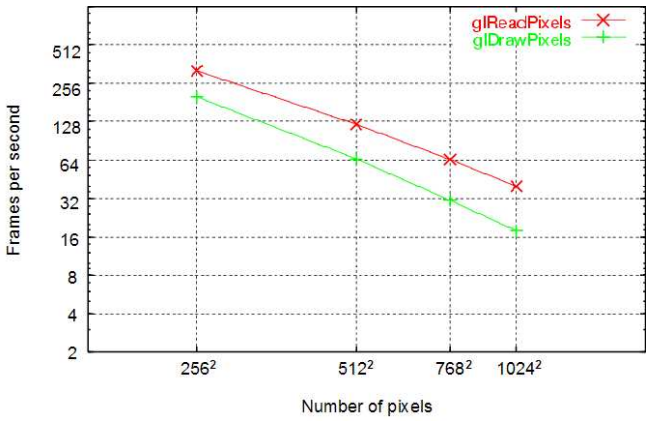

Figure 4: *Framebuffer performance*—The OpenGL API provides the `glReadPixels` and `glDrawPixels` functions to manipulate rectangular areas of the framebuffer. This graph shows the performance of reading and writing various sizes of 32-bit RGBA framebuffers on a single cluster node.

smaller pieces and distribute them to remote nodes is a major drawback for the Simian/Chromium combination.

An obvious way to overcome this limitation is to simply incorporating the divide-and-conquer technique used by the cluster-aware version of Simian into a Chromium SPU. While this solution would overcome the immediate issue of subvolume creation and distribution, other factors, such as the time and effort required to implement the functionality or the flexibility of the resulting code, must be considered carefully.

3.3 Degree of Interactivity Provided by the Image Rendering and Compositing Schemes

The performance degradation introduced by Simian's current swapping mechanism for handling large-scale datasets and the loss of interactivity that results provide the impetus for moving to a cluster-based approach. The goal, then, is to restore the interactive performance for large-scale datasets.

As noted in the previous section, the cluster-aware Simian creates and distributes subvolumes to reduce or eliminate the need for texture swapping. It stands to reason that interactive performance is a natural consequence of distributing the data across many nodes. However, new subtleties arise that are of no concern when running the application on a single node.

Because the dataset is distributed across the cluster, no one node has rendered the entire volume. The result of any individual node is simply an image of its portion of the dataset. These "subimages" must be recombined before being displayed, and this new composition phase introduces latencies that can impact the application's performance.

First, the images must be read from each node's framebuffer. This readback operation adds latency. The images must then be transmitted to the compositing node or nodes, introducing communications latencies. Next, the partial images are composited by alpha-blending the images on a pixel-by-pixel basis. The transmission and composition steps may be repeated a number of times depending on the specific compositing algorithm that is employed. Finally, the final image must be written to the display node's framebuffer, which, like the readback operation, adds latency.

A number of image compositing algorithms have been proposed, and we have chosen to implement the binary-swap scheme introduced in [7]. Both of the framebuffer operations discussed above negatively impact the overall performance of this scheme. In fact, their necessity will plague any parallel version of Simian that must output the final image to a particular node's display. Figure 4 shows the performance of reading and writing regions of the framebuffer for the cluster's graphics hardware. However, the latencies resulting from the transmission and combination of the partial images currently bound the maximum attainable frame rate of the cluster-aware application. Despite these latencies, the cluster-aware version of Simian is able to restore interactive performance (multiple frames per second) for large-scale datasets.

There are a few points worth noting about the Simian/Chromium combination despite the limitations described above. First, using the `tilesort` SPU results in a much simpler compositing algorithm: at the end of a frame, the rendering nodes simply read the contents of their framebuffers and transmit the partial images to the display node. Because each server renders some disjoint subset of the total image space, each subimage can simply be written to the appropriate portion of the display node's framebuffer. No pixel-by-pixel operations are required, and the partial images are only transmitted once per frame. A second, though obvious, point is that this version of the parallel application also suffers from the framebuffer readback and write operations. However, there may be a significantly smaller number of these operations depending on the particular cluster configuration. Finally, as we discuss in Section 4, using the unmodified application with Chromium does not result in interactive performance for the large-scale datasets that we seek to visualize.

Given the straightforward compositing step that arises from the `tilesort` SPU's operation, handling large-scale dataset distribution so that it permits a simple image space composition scheme may be promising. As with implementing a "divide-and-conquer" SPU, careful attention must be given to factors like the time and effort required to implement new functionality or the resulting code's flexibility.

4 Results

To examine the potential performance benefits of each approach, we experimented with two C-SAFE fire-spread datasets simulating a heptane pool fire, h300_0075 and h300_0130, that were produced by the LES codes described in Section 2. Both of the original datasets were 302x302x302 byte volumes storing scalar values as `unsigned chars`. Simian is capable of rendering volumes using gradient and Hessian components in addition to scalar values. Therefore, each of the fire-spread datasets was pre-processed to included these additional components and then padded to a power of 2, resulting in two 512x512x512x3 byte volumes that store the value, gradient, and Hessian components ("*vgh*") as `unsigned chars`. [2]

We first attempted to visualize both the original and *vgh* versions of each fire-spread dataset using the stand-alone version of Simian and a fairly powerful workstation. This workstation was equipped with an Intel Xeon 2.66GHz processor, 1GB of physical memory, and an NVIDIA GeForce4 Quadro4 900 XGL graphics accelerator with 128MB of texture memory. On this machine, Simian was able to load and render the original datasets using the swapping mechanisms described in Section 2. As expected, the need to swap severely penalized the application's performance, resulting in rates of 0.55 frames per second. However, the OpenGL drivers could not

[2]For the remainder of this discussion, it is assumed that the number of bytes consumed by a raw volumetric dataset is the given size multiplied by three, accounting for each of the three, 1-byte components in a *vgh* dataset.

Rendering Approach	Number of Nodes	h300_0075 *vgh* dataset	h300_0130 *vgh* dataset
Cluster-aware Simian	8	0.52	0.58
	16	1.47	2.15
	32	2.87	3.44
Simian/Chromium combination	8	0.07	0.05
	16	0.05	0.04
	32	0.03	0.02

Table 3: Average frame rates (in frames per second) using various cluster configurations

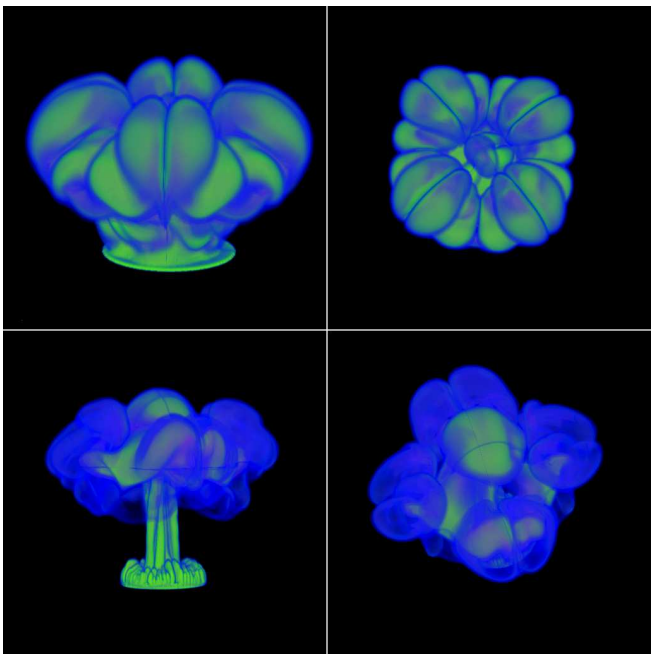

Figure 5: *Visualizations of the C-SAFE* vgh *fire-spread datasets*—Here, two views of the h300_0075 dataset (top) and of the h300_0130 dataset (bottom) that were generated by the cluster-aware version of Simian using 8 cluster nodes are shown.

properly handle the large-scale *vgh* datasets, crashing the application as result.

Having firmly established the need for a cluster-based approach, we tested both the cluster-aware version and the Simian/Chromium combination with the *vgh* datasets using 8, 16, and 32 of the C-SAFE cluster nodes. The results are summarized in Table 3, and Figure 5 shows two views of each *vgh* dataset rendered by the cluster-aware version of Simian.

Note that, with 8 and 16 rendering nodes, the *vgh* subvolumes distributed by the cluster-aware version are 256x256x256 bytes and 256x256x128 bytes, respectively. These subvolumes exceed the maximum "no-swap" volume size permitted by the cluster's graphics hardware (128x128x256 bytes), so in either of these configurations, even the cluster-aware Simian must invoke its texture swapping facilities. However, with 16 rendering nodes, the swapping is much less frequent, resulting in reasonably interactive frame rates. With 32 rendering nodes, the subvolume size is reduced to 128x128x256 bytes, so no texture swapping occurs and interactive frame rates are restored.

As previously described, Chromium is not able to distribute subvolumes among the rendering nodes. As a result, Simian must rely on its texture swapping mechanism. When the application calls for a new block to be swapped into texture memory, Chromium must transmit the block to the appropriate rendering nodes. The resulting delays impose severe performance penalties that grow with the number of rendering nodes. This behavior is reflected in the low frame rates given in Table 3.

5 Discussion

In the preceding sections, we have discussed the problems and open issues that arise when considering two approaches to cluster-based interactive volume rendering with Simian. We have also examined the performance of each approach when visualizing large-scale C-SAFE datasets. It is clear that each approach supports some features that will contribute to Simian's usefulness as a tool for navigating and visualizing large-scale datasets in a clustered environment.

As a result of our initial experience with the C-SAFE cluster and Simian, we have identified several advantages of developing cluster-aware visualization tools. First, these tools can exploit application-specific knowledge to reduce overhead and enhance performance and efficiency when running in a clustered environment. It is often very difficult or extremely impractical to account for application-specific attributes when developing lower-level clustered rendering frameworks, and it may not be possible to realize optimal performance with such packages. Second, cluster-aware applications that are built upon open standards such as MPI are readily ported to a wide variety of hardware and software platforms. While frameworks that mask the clustered environment may provide certain advantages (noted below), using standard interfaces allows an application's components to be reused or combined in new, more flexible ways. Third, cluster-aware applications are not dependent upon the functionality provided by a clustered rendering package. By their very nature, this fact is not true of applications that utilize or incorporate such packages. Finally, because they do not access remote resources via a lower-level framework, cluster-aware applications can exploit the capabilities of the underlying system directly. Adding new functionality to cluster-aware applications may be easier for this reason as well.

Of course, building cluster-aware visualization tools is inherently more difficult than simply running an unmodified application atop a clustered rendering framework. The programmer must consider and account for the clustered environment in which the application is running. The application logic may become more complex, possibly requiring greater programming effort or longer development times. It is important to note, however, that the functionality of an application that relies on a clustered rendering framework may be severely limited by the functionality of the framework itself.

Nevertheless, using clustered rendering frameworks such as Chromium may offer certain advantages. The application can potentially utilize a cluster's resources without being explicitly aware of those resources. While such was not found to be the case with Simian, many applications do fall into this category. Given such an application, programmers may be able to focus their efforts on developing new or more efficient rendering techniques without having to consider the complexities introduced by the clustered environment.

Despite the progression toward the full-featured interactive application that we seek, much more work is necessary. Moving to a cluster-based version of Simian has introduced several new issues, many of which have clear solutions using either approach. What is much less clear, however, is: (1) the flexibility, reusability, and portability of the resulting tools, and (2) the extent of the time and effort required to implement the possible solutions. Although we believe that building cluster-aware visualization tools will satisfy the demands of (1) without exacerbating (2), a thorough discussion of these topics is beyond the scope of this work. We note, how-

ever, that sound software engineering principles demand flexible, reusable, and portable software, and producing tools that satisfy these demands should be the goal of development efforts under either approach. It is also clear that significant time and effort are necessary to develop cluster-based visualization services, regardless whether that effort is expended in developing a cluster-aware application or a lower-level framework.

After carefully weighing these considerations, we have determined that relying on a clustered rendering framework is currently not sufficient for our purposes. Therefore, C-SAFE has chosen to pursue further development of a cluster-aware version of the Simian interactive volume rendering tool.

6 Future Work

Supporting user interaction via direct manipulation widgets in a clustered environment will be the next step toward the application that we seek. As noted, supporting this functionality will require appropriate methods for controlling how and where the widgets are rendered, as well as efficient mechanisms for distributing changes in widget state among the participating nodes.

Enhancing the performance of our software-based image composition scheme is also of interest. By exploiting the instruction-level parallelism provided by Intel Xeon processors, we hope to reduce the negative impact of the pixel-by-pixel alpha blending stage. Further increasing the size of the datasets that can be handled interactively, for example, by using multi-resolution volume rendering techniques, is a topic of interest as well.

Acknowledgments

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).

The authors would like to thank Joe Kniss, Gordon Kindlmann, and James Bigler, all of the Scientific Computing and Imaging Institute, for their contributions to this project. In addition, Wing Yee, of the C-SAFE Fire-Spread team, deserves much thanks for his valuable input.

References

[1] M. Hadwiger et al. High-quality volume graphics on consumer pc hardware. IEEE Visualization 2002 Course Notes.

[2] G. Humphreys, Mike Houston, Yi-Ren Ng, Randall Frank, Sean Ahern Peter Kirchner, and James T. Klosowski. Chromium: A stream-processing framework for interactive graphics on clusters. In *Proceedings of SIGGRAPH*, 2002.

[3] H. Igehy, G. Stoll, and P. Hanrahan. The design of a parallel graphics interface. In *SIGGRAPH Computer Graphics*, 1998.

[4] J. Kniss. Simian–volume rendering. `http://www.cs.utah.edu/~jmk/simian`, Accessed March 23, 2003.

[5] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization*, 2001.

[6] J. Kniss, G. Kindlmann, and C. Hansen. Multi-dimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, pages 270–285, July 2002.

[7] K. Ma, J. Painter, C. Hansen, and M. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14(4):59–68, 1994.

[8] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, July 1994.

[9] James T. Purciful. Three-dimensional widgets for scientific visualization and animation. Master's thesis, Univeristy of Utah, 1997.

[10] G. Stoll, M. Eldridge, D. Patterson, A. Webb, S. Berman, R. Levy, C. Caywood, M. Taveira, S. Hunt, and P. Hanrahan. Lightning-2: A high-performance display subsystem for PC clusters. In *SIGGRAPH Computer Graphics*, 2001.