

# Scalable Interactive Volume Rendering Using Off-the-Shelf Components

Santiago Lombeyda<sup>1</sup>

Laurent Moll<sup>2</sup>

Mark Shand<sup>2</sup>

David Breen<sup>1</sup>

Alan Heirich<sup>2</sup>

<sup>1</sup>California Institute of Technology

<sup>2</sup>Compaq Computer Corporation

## Abstract

This paper describes an application of a second generation implementation of the Sepia architecture (Sepia-2) to interactive volumetric visualization of large rectilinear scalar fields. By employing pipelined associative blending operators in a sort-last configuration a demonstration system with 8 rendering computers sustains 24 to 28 frames per second while interactively rendering large data volumes (1024x256x256 voxels, and 512x512x512 voxels). We believe interactive performance at these frame rates and data sizes is unprecedented. We also believe these results can be extended to other types of structured and unstructured grids and a variety of GL rendering techniques including surface rendering and shadow mapping. We show how to extend our single-stage crossbar demonstration system to multi-stage networks in order to support much larger data sizes and higher image resolutions. This requires solving a dynamic mapping problem for a class of blending operators that includes Porter-Duff compositing operators.

**CR Categories:** C.2.4 [Computer Systems Organization]: Computer-Communication Networks—Distributed Systems; C.2.5 [Computer Systems Organization]: Computer-Communication Networks—Local and Wide Area Networks; C.5.1 [Computer System Implementation]: Large and Medium (“Mainframe”) Computers—Super Computers; D.1.3 [Software]: Programming Techniques—Concurrent Programming; I.3.1 [Computing Methodologies]: Computer Graphics—Hardware Architecture; I.3.2 [Computing Methodologies]: Computer Graphics—Graphics Systems; I.3.3 [Computing Methodologies]: Computer Graphics—Picture/Image Generation; I.3.7 [Computing Methodologies]: Computer Graphics—Three-Dimensional Graphics and Realism

**Keywords:** sort-last, parallel, cluster, shear-warp, volume rendering, ray-casting, shadow mapping, CFD, VolumePro, OpenGL, VIA, Clos.

## 1 Introduction

In previous work [13, 14] a commodity-based architecture (Sepia) was presented for constructing scalable display subsystems for distributed clusters. The architecture allows large numbers of computers to post images onto rectilinear tiles of a large display, and to apply compositing, blending, and other per-pixel operators to those images. This paper reports on the application of a second generation implementation (Sepia-2) of the architecture to the problem of

structured volume visualization using hardware acceleration. While the small (8 rendering computers plus 1 display server) demonstration system sustains performance levels that are unprecedented, we attach more significance to the fundamental principles the demonstration elucidates. By careful recourse to established architectural and rendering theory we argue that the demonstration validates the architecture at larger scales, for a broad range of problems, using a variety of image generating devices.

In this paper we apply the Sepia architecture to the problem of interactive rendering for scalable volumetric data. We use the VolumePro 500 ray casting engine [20] and partition the volumetric data into subvolumes that can be interactively rendered by this engine. The images of these subvolumes are blended concurrently by the Sepia architecture, prior to being warped for final display.

The next section briefly reviews the Sepia architecture in relation to two similar projects called *Lightning-2* and *MetaBuffer* [2, 27]. We identify two fundamental defects in these projects: they cannot support blending, or any non-commutative image compositing operators; and their costs scale explosively, making them impractical in large configurations. The origin of these defects lies in their common use of a mesh topology which is statically ordered, and which scales explosively in complexity as  $(X * Y)$  where  $X$  is the number of inputs and  $Y$  the number of outputs. The Sepia architecture is both more efficient and more powerful because it uses a hierarchical switched topology rather than a statically ordered mesh. This topology is more efficient because it scales in complexity as  $(X + Y)$ , and is more powerful because it supports non-commutative image compositing operators including blending and Porter-Duff operators [5, 19, 21, 25]. We also identify two advantages of these other projects: the use of efficient DVI image acquisition in *Lightning-2*; and a useful class of viewport mapping operators defined for the *MetaBuffer*. We explore the potential of both of these advantages in the discussion section.

The rest of this paper is organized as follows. We first describe the Sepia architecture in comparison to prior work. We introduce the scientific problem addressed in this application, visualization of teravoxel data sets collected from physical fluid mechanical experiments and from simulations. We describe the hardware and application software and firmware, including a derivation of firmware blending arithmetic, and an analysis of system throughput and latency. We conclude our demonstration by describing the data sets used to produce the final images and the performance we observed with those data sets. We then move to a discussion of scalability in which we rely on architectural theory to argue that this demonstration system validates scalability in object space. We discuss technical issues involved in scaling the current application in image space and what this implies for image scaling in general. We conclude with a general discussion.

## 2 Sepia architecture

The Sepia architecture is an FPGA and network-based subsystem for interactive parallel visualization using remote displays and commodity graphics accelerators [13, 14]. The architecture allows general “sort-last” style parallel programming [16] and supports other programming models by providing concurrent high speed com-

positing of large numbers of image streams. The architecture is physically implemented by PCI boards that perform image acquisition, compositing, and redisplay when connected to ports on a high speed network. Figures 1 and 3 show the layout and manufactured second generation prototype PCI board and the cluster used in this demonstration.

The PCI board (Sepia-2) incorporates a VIA-based high speed network interface (ServerNet-2) with attachments for high speed digital image acquisition and display. In addition to the network interface each board contains 3 Xilinx Field Programmable Gate Array devices (FPGA-1, -2, and -3), RAM buffers, and digital expansion interfaces. Configurations at any scale require one board for each graphics accelerator and one for each display device in addition to network switching. The board acquires frame buffer content from the graphics accelerator and merges it with pipelined content arriving through the network. Each network path sustains a 180 MB/s image stream in operation out of a theoretical peak of 220 MB/s. This sustained traffic is equivalent to 80 frames per second of 1024x768 RGB pixels, 48 fps of 1280x1024 RGB, or 32 fps of 1600x1200 RGB. Individual image streams may be tiled together to make larger displays.

The network ASIC drives two network ports with a peak wire speed of 1.25 gigabits per second in each direction on each port. The ASIC sends the inbound data through a 66 MHz 64 bit PCI interface to FPGA-3. In operation an image stream arrives through the network multiplexed across the two ports and is sent to FPGA-3 where it is de-multiplexed and passed to FPGA-2 for processing. In operation the board sustains 180 MB/s of inbound image traffic with an equivalent amount of outbound traffic in the opposite direction, for a sustained PCI throughput of 360 MB/s (inbound plus outbound). The outbound image traffic carries the results of a compositing operation computed by FPGA-2. This traffic is multiplexed by FPGA-3 for transmission by the ServerNet-2 ASIC.

The compositing operation in FPGA-2 combines pixels from the inbound image stream with pixels from a local image source. At small data rates the local image source can be obtained through the host PCI interface controlled by FPGA-1. In our demonstration system we have measured single read performance of 120 to 150 MB/s from host memory through the host PCI, and write performance of 300 MB/s. For higher data rates the digital I/O expansion connectors can support image streams of 500 MB/s in or out, equivalent to over 90 frames per second of 1600x1200 RGB images.

The architecture supports efficient large scale per-pixel image compositing with sub-microsecond switching to dynamically sequence compositing operations. Chains of compositing operators are mapped through network switches onto logical pipelines that correspond to individual tiles of large displays, with arbitrary many-to-1 mappings between graphics accelerators and display tiles. The use of FPGA technology allows firmware reprogrammable compositing operators that can accommodate various pixel formats and application requirements. Our initial focus is to support standard OpenGL depth and blending operators for traditional *sort-last* rendering [7, 9, 13, 15, 16, 17, 18, 24, 28] and to implement operators for photo-realistic shadow mapping [1, 8, 12]. A primary motivation for the architecture has been to visualize extremely large scale problems in science and engineering [26].

## 2.1 Prior art

Sepia addresses the same problem as two similar recent projects, Lightning-2 and the MetaBuffer design. In addition it builds on a body of previous research and commercial efforts including the PixelFlow (by Hewlett-Packard) and the Reality Monster (by SGI) [2, 7, 13, 14, 15, 27].

Like Sepia, Lightning-2 is a hardware system that delivers pixel data from graphics accelerators to remote tiled displays. The

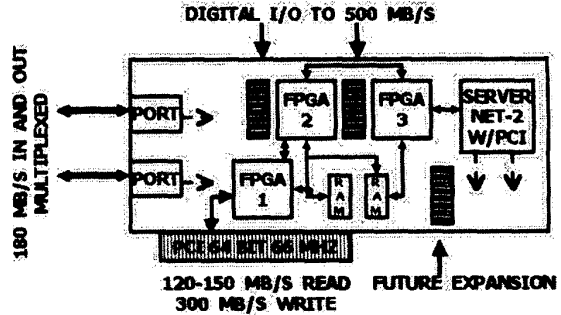


Figure 1: Component layout of the manufactured Sepia-2 board in figure 3. This PCI board contains three Xilinx XC4044XLA and XC4085XLA FPGA devices running at 66 MHz, a 66 MHz ServerNet-2 "Colorado-2" network interface ASIC, buffering, and three expansion connectors.

Lightning-2 hardware is contained in a standalone chassis that connects to computers and displays through DVI cables of approximately three meters in length. A chassis contains some number of boards with each board supporting four inputs from graphics accelerators and driving up to eight outputs to DVI display devices. To support larger configurations the boards may be tiled together into a rectangular mesh and connected with high-speed (unswitched Myrinet) network links. Lightning-2 supports a flexible scheme for mapping viewport pixels onto displays at the level of individual scanline fragments with support for optional per-pixel depth compositing and chroma-keying. The flexibility of this mapping scheme comes at the price of corrupting the source color buffer, a fact that may pose a problem for some applications. For example, it is impossible to distribute a full scanline from a viewport to a set of disjoint tiles without losing some of the pixel content. This makes it impossible to support the usage model illustrated in figure 2. Lightning-2 has pioneered the use of DVI acquisition of RGB content. DVI acquisition is necessary for meeting the latency, throughput and performance requirements of a production quality system.

The MetaBuffer design specifies a mesh-based topology with DVI image acquisition similar to Lightning-2. In place of scanline fragment mapping MetaBuffer describes a rich set of viewport mappings that include multi-resolution support, anti-aliasing, and translations in addition to optional per-pixel depth compositing. Some of the architectural principles of the MetaBuffer design have been demonstrated in an application to large scale iso-surface visualization [29].

At large scales both Lightning-2 and MetaBuffer suffer from mesh scaling. A mesh configuration with 1000 computers and 100 displays requires roughly 100 times as many components as a corresponding Sepia configuration. In configurations where the number of displays is proportional to the number of computers the mesh complexity scales quadratically. In addition to being more scalable the Sepia configuration is more versatile with support for blending as well as depth compositing, shadow mapping and potentially other novel capabilities.

## 2.2 A dynamic mapping problem

The mapping problem is intrinsic to every distributed computation that involves scheduling interdependent operations on multiple functional units. The general problem is formulated in terms of *graph embedding* with a goal to embed an interdependency graph

into a fixed host graph that represents the functional units and their connections. Specific examples include compiling arithmetic instruction sequences for shared memory multiprocessors, scheduling concurrent processes for execution on a cluster, and scheduling scientific calculations on a mesh-based systolic array architecture [4, 10, 23].

In the architectural problem studied here the goal is to map image sources to display tiles through an ordered sequence of image compositing operations. In graph-theoretic terms the problem is to embed any valid set of logical pipelines into a single fixed network topology without exhausting the available network bandwidth at any point along any path. This general problem is NP-complete, an obstacle that may be overcome by judicious assumptions about the host graph. If we want to solve this mapping problem dynamically, potentially on every new frame, it will be necessary to make assumptions that allow efficient polynomial time solution.

Call  $S = \{A, B, \dots, Z\}$  a set of images, each image an  $n$ -tuple of pixels  $A = (a_1, a_2, \dots, a_n)$ , and each pixel a tuple of components  $a_i = (r_{a_i}, g_{a_i}, b_{a_i}, z_{a_i}, \alpha_{a_i}, stencil_{a_i})$ . An image stream computes the result  $A \oplus_1 B \oplus_2 C \oplus_3 \dots$  of a specified sequence of per-pixel imaging compositing operators  $\oplus_i$  that have commutative, associative and/or distributive arithmetic properties.

These arithmetic properties determine constraints for routing data among functional units, as follows. Associative operators may be reparenthesized to express potential concurrency, for example transforming  $((A * B) * (C * D))$  into  $(A * (B * (C * D)))$ . Changing concurrency does not affect requirements for sequencing the operations and as a result these operators impose the most severe scheduling constraints. Examples include the set of associative blending operators which include Porter-Duff compositing operators.

Commutative operators may be scheduled for execution in any sequence and therefore impose easier routing requirements. Depth compositing is an example of a commutative operation. These operators are order-independent. Another class of operators are distributive operators. Supporting this class of operators requires a broadcast capability.

The Sepia dynamic mapping problem is to embed a set of logical pipelines of compositing operators into a physical network. Each pipeline stage corresponds to a Sepia-2 board that implements a compositing operator in firmware. Each stage must be joined to its successor by a network path that guarantees adequate bandwidth along every network link.

The small demonstration system connects the Sepia-2 boards using a full-duplex single stage crossbar. The necessary properties are satisfied trivially in this crossbar which allows any pattern of disjoint pipelines to be embedded with no contention for bandwidth. We provide a general large-scale solution in section 4.1.

### 3 TeraVoxel visualization

The TeraVoxel project<sup>1</sup> couples fluid mechanical experiments with computer simulations with a goal to visualize the resulting data. Physical experiments are performed by observing physical fluid quantities on a regular 2D grid of points within the flow. The data set is extended into three dimensions by sampling repeatedly over time. When the ultimate intended physical resolution of  $1024^2$  points is achieved with sampling at 1024 frames per second the project will generate a teravoxel of data every 17 minutes. Our current system visualizes one-eighth gigavoxel interactively and this exceeds the requirements that the project has generated to date. Our ultimate goal will be to visualize a full gigavoxel ( $1024^3$ ) volume. We expect to achieve this using the same configuration simply by incorporating newer and more powerful graphics accelerators.

<sup>1</sup>National Science Foundation grant EIA-0079871

In order to visualize this data we partition a large data set into smaller subvolumes and visualize each subvolume concurrently. This is followed by concurrently blending the subvolume images, with the blended result delivered to a computer than supports a graphical user interface. This is explained in detail in the rest of this section.

### 3.1 Hardware configuration

We equipped a cluster of eight graphics workstations (Compaq SP750 running Windows2000) with Sepia-2 boards and VolumePro 500 ray casting accelerators [20]. The VolumePro is a black box image generator that loads a data volume and produces a series of viewpoint-dependent images according to a pre-determined hardware algorithm (object order "shear-warp" ray casting [11]). It can support high quality programmable lighting effects and deliver a continuous stream of images to host memory without interrupting the rendering operation. This makes it convenient for image acquisition into the Sepia-2 board through host memory and the host PCI interface. The relatively small image size of  $512 \times 512$  RGBA pixels (1 MB per frame) keeps the resulting transfer latency tolerable. We have observed that the card sustains 24.8 to 27.5 frames per second operating in this mode with volumes of  $256^3$  voxels. We locate the VolumePro and Sepia-2 boards in different peer PCI busses in order to eliminate potential resource conflicts that might result from bus sharing.

The Sepia-2 boards are connected symmetrically to a pair of ServerNet-2 switches. A ninth workstation containing a Sepia-2 board is also connected to these switches. This ninth workstation functions as a display for the eight node cluster. It contains a standard OpenGL accelerator in an AGP-4x interface. This display workstation loads the blended images into texture memory on the accelerator and then displays the texture on the surface of a polygon. This is the standard method used to display images from the VolumePro when it is used in a single workstation.

### 3.2 Application software

The application is a display process that communicates with a set of rendering processes running on the cluster. The display process supports a user interface and allows interactive viewpoint control through mouse movements. It distributes viewpoint information to the renderers over a shared Ethernet. The large data volume is broken into subvolumes with each subvolume assigned to a rendering process. The renderers activate the VolumePro to generate images into host memory, pad these images to a larger size, and then activate the Sepia-2 blending pipeline. At every frame the pipeline blending order is recomputed based on a continuously changing viewpoint. The display process waits for the blended result and displays it on the screen. The stages of rendering, blending and display are fully overlapped and as a result the interactive latency remains between two and three frames. This is only about one frame more than is incurred with the VolumePro in normal usage.

In the standard shear-warp algorithm *base plane* (BP) images are generated in object order and then warped by a 2D linear transformation into the viewing plane [11]. The motivation for this approach is to preserve the efficiencies of object-order ray casting. VolumePro generates a  $512 \times 512$  pixel BP that contains within it the viewpoint-dependent image of a  $256^3$  data volume. In conventional usage this BP is loaded into the texture memory of an OpenGL graphics accelerator. The texture is then applied to a distorted polygon in order to correct the viewpoint-dependent error in the BP image. The resulting viewpoint-corrected image is usually displayed at a higher pixel resolution than the original BP.

In order to avoid artifacts that would result from repeated re-sampling we apply blending to the raw BP image data before it is

texture mapped. Each rendering client generates a 512x512 pixel BP image containing a subimage of the data volume. A client copies this subimage into a larger 1024x1024 pixel “superimage base plane” (SBP). It is these larger SBPs that are blended in real time by the Sepia-2 hardware. The display process receives a blended 1024x1024 pixel result that is ready to load into GL texture memory for the final warp and display. Parameters for this warping are generated by the VolumePro with each image. These parameters are passed to the display process by one of the rendering processes in the form of extra data in an unused part of the SBP image.

### 3.2.1 Firmware arithmetic

The VolumePro generates pixels by applying compositing operations to a series of voxels along a line of sight. This is the operation of *ray casting*. We encounter two arithmetic issues in parallelizing this operation. The first issue is arithmetic equivalence between a single VolumePro computation and the combined result of a set of (smaller) VolumePro computations. The second issue is operator associativity, without which concurrent evaluation is arithmetically undefined. These issues require that we define an associative compositing operator that yields a concurrent result arithmetically equivalent to the original serial result. This operator is not the same as the VolumePro blending operator. In this section we define such an operator and prove its correctness.

The ray casting computation blends the successive contributions of an ordered set of voxels that intersect a line of sight. If  $V(a, s)$  corresponds to a blending operator that combines a new voxel sample  $s$  with an accumulated result  $a$  then the total accumulated result  $a_T$  of an ordered front to back ray casting computation is

$$a_T = V(V(\dots V(V(0, s_1), s_2), \dots), s_n) \quad (1)$$

where  $s_1 \dots s_n$  are successive voxel samples and 0 represents the contribution from a transparent voxel. The ray casting engine implements the following front-to-back blending operator  $V$  for all color channels  $C$  (where  $\alpha$  is the opacity channel,  $C_a$  the accumulated color,  $\alpha_s$  the opacity of sample  $s$ , etc.)

$$\begin{aligned} C_a &\leftarrow C_a + (1 - \alpha_s)(\alpha_s C_s) \\ \alpha_a &\leftarrow \alpha_a + (1 - \alpha_s)\alpha_s \end{aligned} \quad (2)$$

Parallelization breaks (1) into two or more concurrent subcomputations, for example

$$\begin{aligned} a_1 &= V(V(\dots V(V(0, s_1), s_2), \dots), s_{n/2}), \\ a_2 &= V(V(\dots V(V(0, s_{n/2+1}), s_{n/2+2}), \dots), s_n). \end{aligned}$$

The blending computation in the Sepia-2 firmware must compute a function  $F$  such that  $F(a_1, a_2) = a_T$ . Additional requirements on  $F$  are that it is associative, so that the computation can be parallelized into  $n$  concurrent pieces  $a_1 \dots a_n$ , and that 0 is an identity element for  $F$  corresponding to a transparent pixel. We take the standard approach [3] of pre-multiplication by  $\alpha$  and define a pre-multiplier function  $P$  on pixels  $s$  (which have components  $C_s$  and  $\alpha_s$ ),

$$P(s) = (\alpha_s C_s, \alpha_s). \quad (3)$$

We can easily verify that the following  $F$  is associative with identity element 0 (a transparent black voxel) and further that  $V(a, s) = F(a, P(s))$ . This  $F$  will be the compositing operator we implement in firmware. Note that it is not the same as the VolumePro blending operator (2).

$$F(a_1, a_2) = (C_{a_1} + (1 - \alpha_{a_1})C_{a_2}, \alpha_{a_1} + (1 - \alpha_{a_1})\alpha_{a_2}) \quad (4)$$

We can show that  $F(a_1, a_2)$  gives the same arithmetic result as the original ray casting computation  $a_T$  (1).

$$\begin{aligned} F(a_1, a_2) &= F(V(V(\dots V(V(0, s_1), s_2), \dots), s_{n/2}), \\ &\quad V(V(\dots V(V(0, s_{n/2+1}), s_{n/2+2}), \dots), s_n)) \\ &\quad \{ \text{by (3)} \} \\ &= F(F(\dots F(F(0, P(s_1)), P(s_2)), \dots P(s_{n/2})), \\ &\quad F(\dots F(F(0, P(s_{n/2+1})), P(s_{n/2+2})), \dots P(s_n))) \\ &\quad \{ \text{identity} \} \\ &= F(F(\dots F(F(0, P(s_1)), P(s_2)), \dots P(s_{n/2})), \\ &\quad F(\dots F(P(s_{n/2+1}), P(s_{n/2+2})), \dots P(s_n))) \\ &\quad \{ \text{associativity} \} \\ &= F(\dots F(F(\dots F(0, P(s_1)), \dots P(s_{n/2})), \\ &\quad P(s_{n/2+1})), \dots P(s_n)) \\ &\quad \{ \text{by (3) and (4)} \} \\ &= V(V(\dots V(V(0, s_1), s_2), \dots), s_n) \\ &\quad \{ \text{by (1)} \} \\ &= a_T. \end{aligned}$$

Since  $F$  is associative, we can extend this result inductively to any number of terms  $a_1 \dots a_n$ . This concludes our proof that this  $F$  is correct for blending together the result of any number of concurrent subcomputations.

### 3.2.2 Throughput and latency analysis

We have a model for processing latencies from the initial stage of image generation through blending and final display. This model predicts an end-to-end latency of 102 milliseconds, or roughly two and a half frame intervals at 25 frames per second. Of this figure 59 milliseconds are due to the normal VolumePro characteristics when it operates in a single workstation.

The data flow can be modeled in four stages. In the first stage the VolumePro generates a viewpoint dependent BP image and writes it to host memory through PCI bus number 0. We have observed that the time required for this operation, as measured by time spent inside the VolumePro API, varies between 36 and 42 milliseconds, for a sustainable image rate between 24 and 28 frames per second. This first stage is overlapped with all the succeeding stages so that image generation for frame  $i + 1$  occurs concurrently with copying, blending, and display of frame  $i$ .

In the second stage the rendering client software copies the RGBA subimage of the  $256^3$  local data volume from the native BP into a larger 1024x1024 pixel SBP. This stage consistently takes under 5 milliseconds regardless of viewpoint.

In the third stage the Sepia-2 hardware and firmware read the SBP from host memory through PCI bus number 1. This local SBP is blended with network traffic containing the intermediate result of blending SBPs from the other rendering clients. The ultimate blended result is an SBP that is delivered into host memory of the display node. The rate of blending is determined by the rate at which the SBP can be read into the Sepia-2 card through the host PCI interface. This is between 120 and 150 MB/s when the host PCI is not otherwise contended. This is much less than the Sepia-2 network transport (ServerNet-2) which can sustain 180 MB/s. At the slower rate of 120 MB/s this blending operation for a single 4 MB image will take 33.33 milliseconds, plus a sub-millisecond synchronization barrier and time to drain the pipeline. Taking all this into account the blending operation can be estimated at 34 milliseconds after which time the blended SBP result is in RGB form on the display node.

In the fourth stage the display process uploads the fully blended SBP into OpenGL texture memory and waits at most one monitor

refresh interval for the image to appear. In a well tuned GL driver this upload will occur at over 900 MB/s through an AGP-4x interface. We have verified such an upload rate in our workstations. At this rate the 1024x1024 RGB blended SBP texture uploads in under 4 milliseconds. At 60 Hz a monitor refresh interval is under 17 milliseconds.

We can model the worst case of these accumulated latencies. The original image generation takes no more than 42 milliseconds. In normal operation in a single workstation the VolumePro accumulates latency due to image generation and a single video refresh interval, for an estimated worst-case total of 59 milliseconds. In our demonstration system the subsequent stages of copying, blending, and texture upload take no more than 5, 34, and 4 milliseconds respectively. These figures total to a worst-case estimate of 102 milliseconds. Additional latencies of a few milliseconds may potentially result from operating system scheduling artifacts. A variety of factors may result in better latencies than these worst-case estimates. Parallelization thus adds approximately one additional frame of latency, which is roughly equal to the time required for SBP blending. We expect similar latencies at larger scales.

### 3.3 Results

Figures 3 through 5 show our demonstration equipment and a series of images generated by that equipment. Images were generated from computer models and simulations, with data sets of  $512^3$  and  $1024 \times 256 \times 256$  voxels. All of the data sets were visualized interactively at between 24 and 28 frames per second. We are unaware of any other demonstrations that have achieved these frame rates on data volumes of these sizes. The frame rates were determined by the VolumePro rendering times, which varied with different viewpoints. The Sepia-2 board was not the bottleneck in the computation and had considerable unused sustainable throughput. Detailed runtime measurements confirmed the principal performance bottlenecks for this application were the image generation process itself and the data rate through the host PCI interface into the Sepia-2 board.

## 4 Scalability

We have presented a small scale demonstration and claim that it validates a large scale result. In this section we show how to expand the hardware configuration from a single crossbar to arbitrarily large configurations while preserving linear scaling in complexity, and the latency and routability required for dynamic mapping. We then discuss the problem of scaling the image size to higher pixel resolutions.

### 4.1 Object space scaling

We have shown that a single-stage crossbar system satisfies requirements for latency, routability, and scaling. We argue by induction in the size of this crossbar that the demonstration validates the Sepia architecture at larger scales. Since physical crossbars have a scaling limit we have to show that equivalent results can be obtained using more elaborate multi-stage structures.

In this section we present a model for such structures based on the symmetric *Clos* topology [5] and show that this model has the scaling, latency and routability that we require. The model scales in component complexity as  $(X + Y)$ , and in some cases has a constant as small as  $(p-2)^{-1}$  with  $p$  the number of ports per switch. The maximum length path in this model adds only a few stages of latency more than a single crossbar. As a result a logical pipeline with thousands of stages will accumulate only a few milliseconds of network path latency.

### 4.1.1 Recursive Clos structures

The Clos model is one of the most widely studied network topologies [5]. We are interested in the specific case of two-layer symmetric Clos structures. These are constructed from two rows of switches with  $p$  ports per switch. Each row has the same number of switches, and each switch dedicates half its ports to external connections, and half to internal links shared with switches in the opposite row. Any given switch shares the same number of internal links with each switch in the opposite row.

For example, with  $p = 16$  it is possible to build a two-layer structure with 128 external ports and 64 internal links, populated by two layers of 8 switches each. Each top (bottom) layer switch supports 8 external ports, and 8 internal ports that share links with the 8 switches in the bottom (top) layer. In general a two layer structure will afford  $(p^2/2)$  external ports and will require  $p$  switches and  $(p^2/4)$  internal links. The two-layer structure scales linearly in link and switch complexity with increasing numbers of external ports.

The Slepian-Duguid (SD) theorem [25] proves that the two layer symmetric Clos structure provides full bisection-bandwidth between external ports in the top and bottom layers. In particular the theorem guarantees that a set of deterministic routing tables exist that can simultaneously route traffic among any set of pairings of external ports, where each port in a pairing is taken from a different layer (top or bottom) of the structure. The theorem also provides an algorithm to construct these routing tables.

To obtain more than  $(p^2/2)$  external ports the construction may be repeated recursively by replacing each individual crossbar with a two layer structure. For example, the limit of a two layer structure with  $p = 16$  is 128 external ports, and with  $p = 32$  this limit is 512. We can use this structure recursively to build a four layer structure from  $(p^2/2)$  repeated two layer structures. A four layer structure with  $p = 16$  requires 2K (2,048) switches and provides 8K (8,192) external ports. When  $p = 32$  these numbers are 16K (16,384) and 128K (131,072). Scaling is linear up to these maximum sizes.

The SD theorem applies to the four layer structure, and by induction in the number of recursion levels, to recursive structures built in this way with eight, sixteen, and more layers. In principle when the number of layers is taken into account the scaling is  $\mathcal{O}(n \log n)$ . In practice two or four layer structures should be large enough for any currently realistic configuration, and so in practice the scaling is linear.

### 4.1.2 Sepia pipeline mapping

We want to preserve the property that runtime mapping may be accomplished by routing image streams through the network according to network address. In the restrictive case of commutative operators the Sepia mapping problem is trivial. Routing in this case can be achieved using a simple serial chain of operators. An example might be a flight simulator application that involves only surface rendering and depth compositing. Such cases can be supported by a simple daisy chain of  $X/(p-2)$  switches with  $(p-2)$  external ports per switch.

In the case of general operators with non-commutative arithmetic properties the Sepia mapping problem requires a more complex topology. The SD theorem established that the two-layer symmetric Clos topology supports arbitrary mappings between two equal-sized sets of external ports. The theorem assumed half-duplex network links. The Sepia mapping problem requires routing image streams in a predetermined sequence among compositing operators (Sepia-2 boards) attached to external full-duplex network ports.

We can transform the Sepia mapping problem into two instances of the all-pairs assignment problem addressed by the SD theorem. The full-duplex nature of the physical network permits us to treat two such assignment problems simultaneously, one in each direction. Every path between stages in the mapping problem corre-

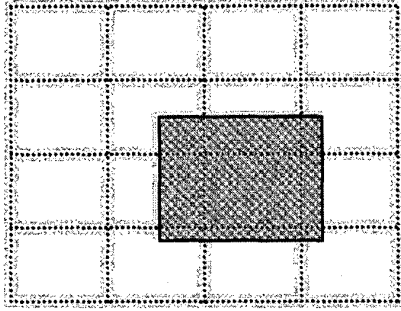


Figure 2: A viewport of 1600x1200 pixels intersecting 9 projectors in a 4Kx3K pixel tiled display wall. Each projector has 1024x768 pixel resolution.

sponds to a special case of two pairings in the assignment problem. These pairings have the form  $(AB, BC)$  where  $A$  and  $C$  are external ports in the bottom layer of the Clos topology, and  $B$  is an external port in the top layer. Thus every pair of pipeline stages is joined by a network path that travels from the bottom layer to the top, and then back to the bottom layer.

This proves Sepia routability in the symmetric Clos model for any number of levels of recursion and thus for any scale. The preceding discussion has already demonstrated linear scaling in complexity and negligible increase in interactive latency. Taken together these conclude our argument that the demonstration at small scales with a single crossbar validates the Sepia architecture at larger scales.

We finally note that a simple optimization can reduce the switch cost of a two-layer structure by 25%. Sepia-2 boards are only attached to external ports on the bottom layer of a multi-layer structure. External ports on the top layer are unused and as a result the number of top layer switches can be cut in half by eliminating these unused ports. The corresponding changes to routing tables are straightforward.

## 4.2 Image space scaling

The ultimate goal of the TeraVoxel project is to visualize  $1024^3$  voxels interactively. We expect to achieve this using eight nodes due to the rapidly increasing performance levels of texture mapping hardware in off-the-shelf OpenGL accelerators [9]. The Sepia-2 board blends a 1024x1024 SBP containing RGBA pixels at 24 to 28 frames per second with time to spare. This performance should be enough for higher resolution images produced by OpenGL accelerators. For example, 1600x1200 RGBA images could be blended at 24 frames per second, and 1280x1024 RGBA images at 36 fps. As a result we believe that a single image stream is adequate for this application and we don't require scaling to multiple displays.

In general we expect image scaling to involve the situation illustrated in figure 2 in which viewports are not constrained to coincide with tile boundaries. For example a 1600x1200 pixel viewport driving a wall of 1024x768 resolution projectors may intersect as many as 9 projectors. We have experimented with image tiling and reassembly through firmware and believe the model in figure 2 poses no substantial difficulties. The impact on our architecture is a requirement for a new routability proof to account for image sources contributing to more than one logical pipeline, and a dynamic progress property for frame reassembly. The resulting usage model is natural and may facilitate anti-aliasing and multi-resolution mechanisms similar to those described for the MetaBuffer.

## 5 General discussion

We expect forthcoming generations of OpenGL accelerators to provide the performance needed to render larger volumes on our 8 node cluster. The only practical way to incorporate such accelerators is to acquire images efficiently through a DVI interface, as the Lightning-2 project has explained. Unfortunately such acquisition is not yet practical at workstation display rates. Immature DVI transmitter and receiver components restrict sustainable data rates to well below the DVI specified maximum of 1600x1200 pixels at 60 Hz using dual channels (six bytes per pixel, 660 MB/s total). And driver software for graphics accelerators must be modified to allow double buffering, non-standard DVI transmitter configuration, and other features. Obtaining the needed cooperation from leading graphics accelerator manufacturers has up until now been impossible. Despite this difficulty we believe the required features will eventually be supported in mainstream accelerators and we intend to continue to work on this problem.

Only double-buffered DVI acquisition strategies will suffice for workstation display rates with non-RGB content. Any single buffered strategy requires the graphics pipeline to stall until the single buffered data has been acquired. If this data is acquired through DVI the stall is for an entire DVI interval. In current components this penalty ranges from 8.333 milliseconds per frame for 1280x1024 images at 120 Hz, to as long as 16.666 ms per frame at 1600x1200 resolution and 60 Hz.

This is roughly the same amount of idleness as incurred by pixel readback strategies at 300 MB/s. We have verified such readback rates in a mainstream accelerator (Matrox G400 under Linux). Because this idleness occurs on every frame the severity of the penalty increases at higher frame rates. For example, acquiring 60 frames per second at 1280x1024 pixel resolution with a 120 Hz refresh would require idling the graphics pipeline at least 50% of the time. The problem is more severe at higher resolutions which require lower monitor refresh rates and therefore longer DVI intervals. For example, acquiring 1600x1200 images at 60 Hz results in a 100% idled pipeline with a single buffered strategy. Thus unless we can obtain double buffering in graphics drivers, frame buffer acquisition through DVI at workstation display rates will be scarcely more efficient than pixel readback.

## 6 Acknowledgements

We are indebted to Glenn Lupton and Jack Alexander of Compaq's visualization group for assistance with performance analysis and debugging of the application software. John McCorquodale prepared the data for the Visible Human in figure 5. Brian von Herzen contributed to a discussion of image space scaling. Hugh Lauer, Andy Vesper, and Larry Seiler of RTViz have been helpful in understanding details of the VolumePro system. This work was supported by National Science Foundation grants ACI-9982273 and EIA-0079871, by the U.S. Department of Energy's ASCI Center for Simulation of Dynamic Response of Materials, by Caltech's Center for Advanced Computing Research, and by Compaq Computer Corporation's Enterprise Systems Laboratories and Systems Research Center.

## References

- [1] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image based methods for rendering soft shadows. In *SIGGRAPH 00*, pages 375–384, 2000.
- [2] W. J. Blanke, C. Bajaj, D. Fussel, and X. Zhang. The MetaBuffer: a scalable multiresolution multidisplay 3D

- graphics system using commodity rendering engines. Technical Report TR2000-16, University of Texas at Austin, TICAM, 2000.
- [3] James Blinn. Compositing - theory. In *Jim Blinn's Corner*, pages 179–190. Morgan Kaufmann Publishers, 1998.
- [4] Shahid Bokhari. *Assignment problems in parallel and distributed computing*. Kluwer Academic Publishers, 1987.
- [5] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32:406–424, 1953.
- [6] Paul E. Dimotakis, Andrew W. Cook, Stefan B. Deusch, James M. Patton, and Santiago V. Lombeyda. Rayleigh-Taylor instability studies from direct numerical simulations. In *Proceedings of the 52nd Annual Meeting of the American Physical Society's Division of Fluid Dynamics (DFD99)*, 1999.
- [7] John Eyles, Steven Molnar, John Poulton, Trey Greer, Anselmo Lastra, Nick England, and Lee Westover. PixelFlow: The realization. In *Proceedings of the 1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 57–68, 1997.
- [8] Pemith Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In *SIGGRAPH 01*, 2001.
- [9] Joe Kniss, Patrick McCormick, Allen McPherson, James Ahrens, Jamie Painter, Alan Keahey, and Charles Hansen. Interactive texture-based volume rendering for large data sets. *IEEE Computer Graphics and Applications*, 21(4), 2001.
- [10] H. T. Kung and D. Stevenson. A software technique for reducing the routing time on a parallel computer with a fixed interconnection network. In *High Speed Computer and Algorithm Organization*, pages 423–433. Academic Press, 1977.
- [11] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transform. In *SIGGRAPH 94*, pages 451–457, 1994.
- [12] Tom Lokovic and Eric Veach. Deep shadow maps. In *SIGGRAPH 00*, pages 385–392, 2000.
- [13] Laurent Moll and Alan Heirich. Scalable distributed visualization using off-the-shelf components. In *Proceedings of the IEEE Symposium on Parallel Visualization and Graphics (PVG'99)*, pages 55–59, 1999.
- [14] Laurent Moll, Alan Heirich, and Mark Shand. Sepia: Scalable 3D compositing using PCI Palette. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'99)*, 1999.
- [15] Steven Molnar. Combining Z-buffer engines for higher-speed rendering. In *Proceedings of the Eurographics Third Workshop on Graphics Hardware*, pages 171–182, 1988.
- [16] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):22–32, 1994.
- [17] Kenneth Moreland, Bryan Wylie, and Constantine Pavlakos. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, 2001.
- [18] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide*. Addison-Wesley, 1997.
- [19] Keith Packard. Translucent windows in X. Technical report, XFree86 Core Team, <http://www.xfree86.org/~keithp/talks/KeithPackardAls2000/index.html>, 2000.
- [20] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro real-time ray-casting system. In *SIGGRAPH 99*, pages 251–260, 1999.
- [21] Thomas Porter and Tom Duff. Compositing digital images. In *SIGGRAPH 84*, pages 253–259, 1984.
- [22] Visible Human Project. Visible human data set. Technical report, U.S. National Library of Medicine, [http://www.nlm.nih.gov/research/visible/visible\\_human.html](http://www.nlm.nih.gov/research/visible/visible_human.html), 2001.
- [23] Arnold Rosenberg. Issues in the study of graph embedding. In *Graph Theoretic Concepts in Computer Science*, pages 150–176. Springer-Verlag, 1981.
- [24] Rudrajit Samanta, Thomas Funkhouser, and Kai Li. Parallel rendering with K-way replication. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, 2001.
- [25] D. Slepian and A. M. Duguid. In *Switching and traffic theory for integrated broadband networks*, pages 53–84. Kluwer Academic Publishers, 1990.
- [26] Paul H. Smith and John van Rosendale. Data and visualization corridors. Technical Report CACR-164, California Institute of Technology, Center for Advanced Computing Research, 1998.
- [27] G. Stoll, M. Eldridge, D. Patterson, A. Webb, S. Berman, R. Levy, C. Caywood, S. Hunt, and P. Hanrahan. Lightning-2: a high-performance display subsystem for PC clusters. In *SIGGRAPH 01*, 2001.
- [28] Brian Wylie, Constantine Pavlakos, Vasily Lewis, and Ken Moreland. Scalable rendering on PC clusters. *IEEE Computer Graphics and Applications*, 21(4), 2001.
- [29] Xiaoyu Zhang, Chandrajit Bajaj, William Blanke, and Donald Fussell. Scalable isosurface visualization of massive datasets on COTS clusters. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, 2001.