High-Performance Scalable Graphics Architecture for High-Resolution Displays

Byungil Jeong+, Luc Renambot, Rajvikram Singh, Andrew Johnson, Jason Leigh
Electronic Visualization Laboratory, University of Illinois at Chicago

Abstract

We present the Scalable Adaptive Graphics Environment (SAGE), a graphics streaming architecture for supporting collaborative scientific visualization environments with potentially hundreds of megapixels of contiguous display resolution. In collaborative scientific visualization it is crucial to share high resolution visualizations as well as high definition video among groups of collaborators at local or remote sites. Our network-centered architecture allows collaborators to simultaneously run multiple visualization applications on local or remote clusters and share the visualizations by streaming the pixels of each application over ultra high speed networks to large tiled displays. This streaming architecture is designed such that the output of arbitrary M by N pixel rendering cluster nodes can be streamed to X by Y pixel display screens allowing for user-definable layouts on the display. This dynamic pixel routing capability of our architecture allows users to freely move and resize each application's imagery over the tiled displays in run-time, tightly synchronizing the multiple visualization streams to form a single stream. Experimental results show that our architecture can support visualization at multi-ten-megapixel resolution with reasonable frame rates using gigabit networks.

CR Categories and Subject Descriptors: I.3.2 [Computer Graphics]: Graphics Systems-Distributed/network graphics; C.2.2 [Computer-Communication Networks]: Network Protocols-Applications; C.2.4 [Computer-Communication Networks]: Distributed Systems-Client/server, Distributed applications

Additional Keywords: collaborative scientific visualization

1 Introduction

We envision situation-rooms and research laboratories in which all the walls are made from seamless ultra-high-resolution displays fed by data streamed over ultra-high-speed networks from distantly located visualization, storage servers, and high definition video cameras [1,6]. It will allow local and distributed groups of researchers to work together on large amounts of distributed heterogeneous datasets. From our prior work on the Continuum [2], we have learned that it is crucial for collaborators to have both local control (e.g. on a tablet or laptop) and the casual ability to share their work and see what others are working on (e.g. on a large tile display). We are taking the next steps toward this vision by building LambdaVision - an 11x5 tiled display with a total resolution of 100 megapixels and developing SAGE, the Scalable Adaptive Graphics Environment (see Figure 1). High-resolution displays like LambdaVision are necessary to support geoscientists working with aerial and satellite imagery (365Kx365K pixels maps) and neurobiologists imaging the brain with montages consisting of thousands of pictures from a high-resolution microscope (4Kx4K pixels sensor). SAGE allows the seamless display of various networked applications over the high resolution displays. Each visualization application (such as 3D rendering, remote desktop, video streams, 2D maps) streams its rendered pixels (or graphics primitives) to the virtual high-resolution frame buffer of SAGE, allowing for any given layout onto the displays.

The graphics streaming architecture of SAGE address two non-trivial problems in scientific visualization. One is heterogeneity: since most visualization applications are closely tied to their graphics environment, it is difficult to integrate various visualization applications into a unified graphics environment. For example: visualization applications that are developed for desktop computers are rarely able to take advantage of the processing power of a cluster of graphics computers; conversely visualization applications developed for clusters rarely function on desktop computers. The other is scalability: the ability of visualization software and systems to scale in terms of the amount of data they can visualize and the resolution of the desired visualization [5]. SAGE addresses the heterogeneity problem by decoupling graphics rendering from graphics display so that visualization applications developed on various environments can easily migrate into SAGE by streaming their pixels into the virtual frame buffer. Also, SAGE provides scalability by supporting any number of rendering and displaying nodes, number of tiles, and screen resolution, and the SAGE visualization applications have extremely fast access to huge database at remote or local sites taking advantage of affordable ultra-high-bandwidth networks.

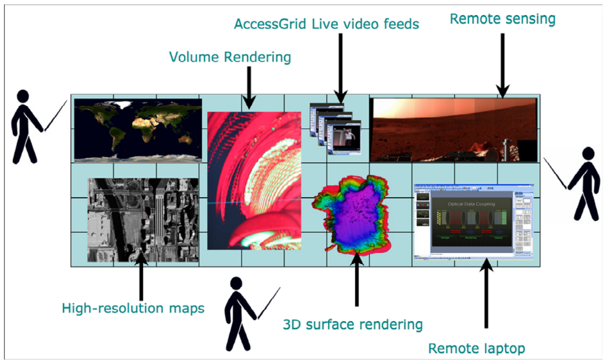

Figure 1. Example of a collaborative SAGE session

As for user interaction, SAGE's Free Space Manager (i.e. window manager) provides an intuitive interface for moving and resizing visualizations on the tiled display. When a visualization window is moved from one portion of the screen to another, the Free Space Manager informs the remote rendering clusters of the new destination for the streamed pixels, giving the user the illusion that they are working on one continuous computer screen, even though the systems performing the visualizations may be several thousand miles apart. The Free Space Manager is akin to a traditional desktop manager in a windowing system, except that it can scale from a single tablet PC screen to a desktop spanning over 100 million pixel displays.

The main contributions of this paper are:

- It presents a new paradigm to decouple rendering and display processes by dynamically routing pixels from applications to high-resolution displays,
- It describes the design and the implementation of the SAGE environment enabled by high-bandwidth networks,
- Using two benchmark applications, it proposes an evaluation of the scalability of SAGE,
- Finally, various applications ported to the SAGE environment are described.

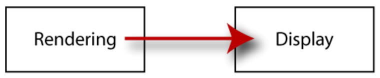

Figure 2. Remote rendering : VNC, Microsoft Remote Desktop

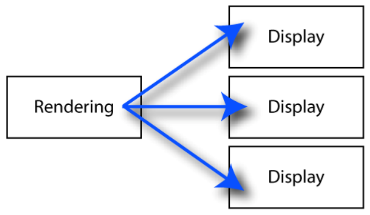

Figure 3. Parallel rendering from single source: WireGL

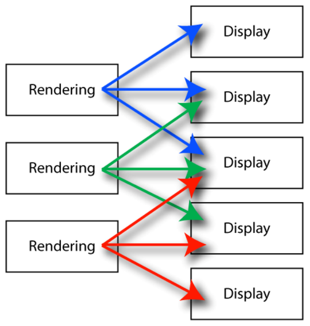

Figure 4. SAGE on local area network

2 Related Work

There are several existing systems with parallel or remote rendering schemes related to SAGE. Figure 2 shows a simple remote rendering scheme using a remote desktop network protocol such as VNC or Microsoft Remote Desktop. They were designed to transmit screens of single desktops to remote computers over slow networks operating on event triggered streaming mechanisms that are not suitable for real-time streaming of scientific visualization or collaborative applications.

Figure 3 shows a sort-first parallel rendering scheme from a single source exploited in WireGL [7] or parallel scene-graph rendering. This approach allows a single serial application to drive a tiled display and streamed graphics primitives to be rendered in parallel on display nodes, but it has poor data scalability due to its single source limitation.

Flexible scalable graphics systems such as Chromium [8] and Aura [11] are designed for distributing visualization to and from cluster driven tiled-displays. They have a similar parallel rendering scheme with SAGE on local area network as shown in Figure 4. However, Chromium is not designed to execute multiple applications on a tiled display, and its applications have a static layout on the tiled display – it can divide the tiled display into several parts and execute multiple applications, but each tile can support only one application. Chromium's DMX extension allows executing multiple applications and window moving and resizing but doesn't support parallel applications – it has a single source (serial application) as shown in Figure 3. Moreover, its design is not suitable for graphics streaming over wide-area networks, while SAGE is designed for distributed rendering over wide-area network, as shown in Figure 5, by using various protocols designed for high-bandwidth and high round-trip time networks. Also, we will extend SAGE to scalably support distance collaboration with multiple endpoints by streaming pixels to all

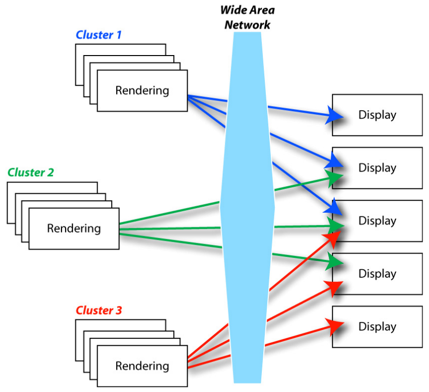

Figure 5. SAGE over wide area networks

the participating endpoints using either traditional router-based multicasting or photonic multicasting.

IBM's Scalable Graphics Engine [9] is a hardware-based approach that allows for reception of pixels streamed over networks and can drive an array of displays synchronously. Currently, it is limited to 16 1GigE network inputs, 4 DVI outputs, and SGE specific network protocol which prevents users from exploiting advanced network protocols. Several SGE devices can be ganged and synchronized to drive larger displays. As 10GigE (10 gigabit Ethernet) network becomes prevalent, 1GigE network interface becomes a drawback of SGE.

Our previous work, TeraVision [3], is a scalable platform-independent solution that is capable of transmitting multiple synchronized high-resolution video streams between single workstations and/or clusters. TeraVision was also designed for high-speed graphics streaming over wide-area network and SAGE directly exploits its network transport libraries. However, TeraVision doesn't have dynamic pixel routing capability - it has a static application layout on a tiled display and isn't suitable for supporting parallel applications and multiple instances of applications – we need the same number of TeraVision boxes as rendering nodes, and each TeraVision box can stream only one application's pixels.

3 SAGE

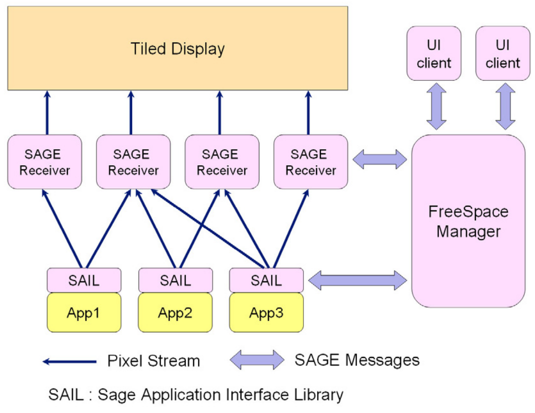

Figure 6. SAGE components

The Scalable Adaptive Graphics Environment (SAGE) consists of the Free Space Manager, SAGE Application Interface Library (SAIL), SAGE Receiver, and User Interface (UI client) as shown in Figure 6. The Free Space Manager receives user commands from UI clients and controls pixel streams between SAIL and the SAGE Receivers. SAIL captures application's output pixels and streams them to appropriate SAGE Receivers. A SAGE Receiver can receive multiple pixel streams from different applications and displays streamed pixels on multiple tiles – a SAGE Receiver can drive multiple tiles. A UI Client sends user commands to control the Free Space Manager and receives messages that inform users of current status of SAGE.

3.1 Free Space Manager

The Free Space Manager is a central control unit of SAGE. It communicates with UI clients receiving user commands and sending SAGE status messages. When the Free Space Manager receives user commands, it sends appropriate control messages to SAIL and the SAGE Receivers to perform various SAGE functions such as application execution, window movement and resize, and window z-order change (overlapping windows). When the functions are completed, SAIL or a SAGE Receiver informs the Free Space Manager to update SAGE status to the UI clients, for example changed window position or size. Functions such as window movement and resize require multiple messages to be passed between the Free Space Manager and SAIL or SAGE Receiver. In that case, the Free Space Manager becomes an information broker between SAIL and SAGE Receiver collecting information from SAIL and distributing it to SAGE Receiver or vice versa.

Also, the Free Space Manger is an information server for SAGE. It contains several possible execution configurations of SAGE applications, run-time information of each application instances, and the tile display configuration. Execution configurations consist of number of rendering nodes, command lines for executing applications, initial position and size of the application window, and the network protocol for pixel streaming. Multiple configurations are possible for each application so the user can execute multiple instances of an application with different configurations by giving different configurations in the application execution commands. Run-time application instance information includes current window position and size, z-order of the window, performance data of each instance and number of instances of each application. Any changes in this information are instantly updated to all UI clients so that all of them are synchronized with each other and the tiled display – This is very important when the UI clients have a GUI. The tiled display configuration consists of the number of tiles, number of display nodes, screen resolution, mullion information, and information for each display node – IP address, number of tiles it drives, positions of tiles. When the Free Space Manger is initialized, it sends this information to all SAGE Receivers to initialize them also.

3.2 SAGE Application Interface Library (SAIL)

SAIL is the library for SAGE applications to communicate with the Free Space Manager and stream pixels to SAGE Receivers. When a SAGE application is executed, it creates and initializes a SAIL object – if the application runs on multiple rendering nodes, one SAIL object per each node. For initialization, each application node has to give SAIL the pixel format – 24bit RGB, 32bit RGBA, 16bit RGB, etc - and information about the image rendered by the node – resolution of the image and the image's coordinates in the whole application image. Then, SAIL connects to the Free Space Manager and the Free Space Manager informs SAIL how it should split the application images and where to stream the split images. Whenever a SAGE application gives SAIL a new frame by a "sageSwapBuffer" call, SAIL splits and sends the new images to the proper SAGE Receivers.

The SAIL API allows the programmers to describe the pixel buffer and the position of the buffer into the application output image. The latter is needed when programming parallel application where each processor will generate a portion of the whole picture. This mode is used either to speed up the application (each processor generates less pixels) or to achieve higher resolution (each processor generates the same amount of pixels). The only extra SAIL function in the application is the call to the 'sageSwapBuffer' function. This minimal API makes very easy to port to SAGE any application producing images or pixels in an uncompressed format.

Table 1 shows the code for a minimal SAGE application: it is an application that registers with name `'render'` to the Free Space Manager, it is a sequential application (it's rank is 0), the unique process will generate the whole image as described in the `'renderImageMap'` data structure, the application will generate 24bit images in the RGB format, in a buffer where the pixels are laid out from bottom to top. After the initialization phase, the application sends the pixels contained in the `'rgbBuffer'` variable to the SAIL library.

```
sailConfig scfg;
scfg.cfgFile = "sage.conf";
scfg.appName = "render";
scfg.rank = 0;

sageRect renderImageMap;
renderImageMap.left = 0.0;
renderImageMap.right = 1.0;
renderImageMap.bottom = 0.0;
renderImageMap.top = 1.0;

scfg.imageMap = renderImageMap;
scfg.colorDepth = 24;
scfg.pixFmt = TVPIXFMT_888;
scfg.rowOrd = BOTTOM_TO_TOP;

sageInf.init(scfg);

while (1) sageInf.swapBuffer( rgbBuffer );
```

Table 1. Minimal SAGE application

The partition of the image and the streaming of the pixels to the display using various protocols are completely transparent to the user.

3.3 SAGE Receiver

SAGE Receivers receive pixel streams from SAIL and control tiled display residing on the display nodes. They are remotely launched by the Free Space Manager and receive all the information about the tiled display from it. Each display node has only one SAGE Receiver, while more than one SAIL objects can exist on a rendering node since they are created per each application instance. Hence, a SAGE receiver has to receive multiple pixel streams from different application instances or different application nodes (on multiple rendering nodes) in one instance and has a circular buffer for each stream so that stream receiving threads fill the buffers and displaying threads read the buffers and display pixels on tiled displays. The current version of SAGE does not require applications to change the rendering resolution when users resize the application windows. Instead, SAGE Receiver scale the applications' image to fit the new window size.

A SAGE Receiver can drive multiple tiles, if the display node on which it resides is connected to more than one tile like 2x1, 3x1, or 2x2 tiles. The layout of the tiled display is specified in a text configuration file, which describes the association and the physical arrangement between displays and computers. Table 2 describes a 2x2 display driven by two computers.

```
TileDisplay
        Dimensions 2 2
        Mullions 0.625 0.625 0.625 0.625
        Resolution 1280 1024
        PPI 90
        Machines 2
DisplayNode
        Name yorda1-10
        IP 10.0.8.121
        Monitors 2 (0,0) (1,0)
DisplayNode
        Name yorda2-10
        IP 10.0.8.122
        Monitors 2 (1,0) (1,1)
```

Table 2. Tiled display configuration

3.4 UI Clients

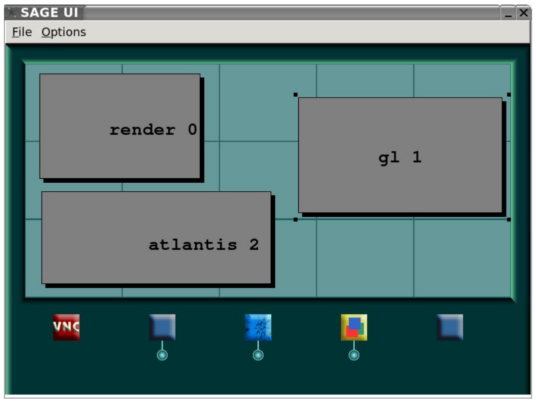

Figure 7. SAGE GUI for a 5x3 tiled display, running 3 applications

UI Clients send user commands to Free Space Manager and show the status of SAGE to the users. UI clients can be a Graphical User Interface (see Figure 7), text-based console, or some tracked devices. We have already implemented a text-based console and the first version of the SAGE GUI that can execute, move, and resize SAGE applications by sending commands to the Free Space Manager. UI clients can reside on any machine that can be connected to the Free Space Manager by network and multiple UI clients can connect to a Free Space Manager and interact simultaneously. This let multiple users connect to SAGE (a Free Space Manager) from their laptops, tablets, or desktops and control SAGE, interacting with each other.

4 Dynamic Pixel Routing

We discuss here how SAGE achieves dynamic pixel routing capability. Section 4.1, 4.3 and 4.4 describe three procedures of dynamic pixel routing. Section 4.2 shows how SAGE synchronizes pixel streams and the close relation between the three procedures and SAGE's synchronization method.

4.1 Establishing Network Connections

Due to SAGE's window move and resize functionality, the possible destinations of a pixel stream from any sender (SAIL object) include all receivers (SAGE Receivers). Since we don't want to kill and recreate the network connections between senders and receivers whenever a window is moved or resized, we establish network connections from all receivers to all senders of an application when the application is executed and pick a subset of them to use for streaming. If we have M tiles and N senders, the total connection number is MxN – when a SAGE Receiver drives multiple tiles, we send separate pixel stream for each tile, so the actual number of stream destinations are not the number of SAGE Receivers but the number of tiles. Also, we create a receiving thread and a sending thread for each connection. Initially all threads are idle, but the threads associated with selected connections are activated when SAGE starts streaming.

4.2 Synchronization of Streams

SAGE's synchronization method is based on TeraVision [3]'s synchronization method using a low latency and high priority TCP channel. The main difference is that SAGE has separate sync master threads on both the sending and receiving sides. In the case of TeraVision, one of sending threads and one of receiving threads are picked as sync master and send sync signals to other sending or receiving threads. However in the case of SAGE, all streams would be blocked if the master sending thread or master receiving thread became idle as a result of window move or resize. To avoid this, we can pick new sync masters and reconnect them to all sync slaves every time we move or resize windows, but this is a very inefficient and time-consuming method. Thus, we create a separate sync master thread on one of the senders and another on one of the receivers.

On the sending side, every active sending thread sends update signals to the sync master when it finishes transferring the previous frame. The sync master sends sync signals to all active sending threads after receiving update signals from every active sending thread – the sync master doesn't wait for idle sending threads' update. After active sending threads receive the sync signal, they start to transfer new frames if the frames are ready by the application's "sageSwapBuffer" call. On the receiving side, every active receiving thread stores pixels into its circular buffer and the associated displaying thread checks the buffer if a new frame arrives and sends an update signal to sync master when it finds a new frame. The sync master sends sync signals to all displaying threads after receiving update signals from every active displaying thread. Displaying threads don't display new frames until they receive sync signals so that all the application images are simultaneously updated on tiled displays.

4.3 Generating Stream Information

Before SAGE starts streaming, the Free Space Manager overlaps the application window's layout onto tiled display's layout as in the Figure 8 to decide which connections are needed for streaming and how to split the application nodes' image. In Figure 8, numbers from 1 to 6 indicate displaying nodes and A and B indicate rendering nodes. Active connections that are assigned to six pixel streams are 1-A, 2-A, 2-B, 4-A, 5-A, 5-B and idle connections are 1-B, 3-A, 3-B, 4-B, 6-A, 6-B. The Free Space Manager sends this information to senders and receivers to create network streaming buffers and activate the threads associated with the active connections and start streaming.

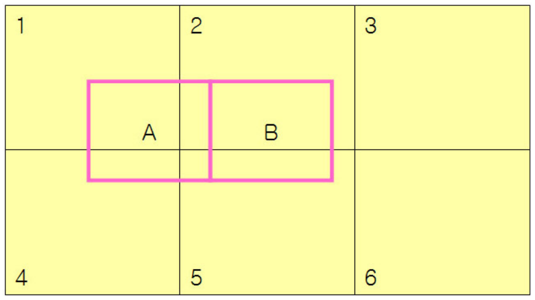

Figure 8. Before window movement

4.4 Modifying Streams

Through window moving operation, a window's layout is changed as in Figure 9 so that the active connections are 2-A, 2-B, 3-B, 5-A, 5-B, 6-B and the idle connections are 1-A, 1-B, 3-A, 4-A, 4-B, 6-A. Four connections (2-A, 2-B, 5-A, 5-B) are still active so the sending and receiving threads associated with these connections keep running, but the streaming buffer sizes and image information (displaying position, aspect ratio, image coordinates on the application image) should be changed. The previous imagery on the display 1 and 4 has to be cleaned, because two connections (1-A, 4-A) have become idle. Two connections (3-B, 6-B) are newly active to start pixel streaming synchronized with four existing streams. To safely apply such changes to pixel streams, we stop the streams momentarily by blocking the sync master threads on the sending side and receiving side. While the streams are stopped, the Free Space Manager generates new stream information (application image splitting and active connection lists) and updates the information on the sending and receiving sides to recreate the network streaming buffers and activates or deactivate threads according to the new information. Then, we unblock the sync master threads to resume streaming.

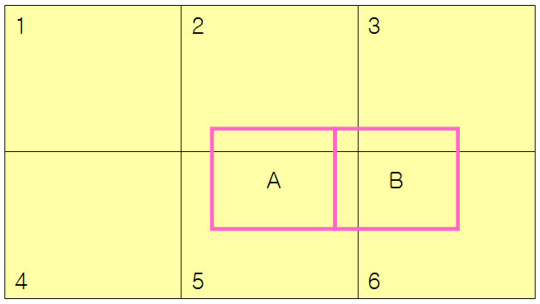

Figure 9. After window movement

5 Implementation

The current implementation of SAGE can execute multiple instances of various visualization applications on local or remote rendering clusters and dynamically stream the pixels of the applications to tiled displays so that multiple application windows can be freely moved or resized on the tiled displays. Each window has a unique z-value to decide which window is in front or behind the others when multiple windows are overlapping. Also, SAGE itself monitors the frame rate of each application and the network bandwidth used by it. Figure 10 shows the current implementation of SAGE, running three distinct application: a high-resolution aerial photography application (on the left), a volume rendering application (in the center) and a VNC session (on the right).

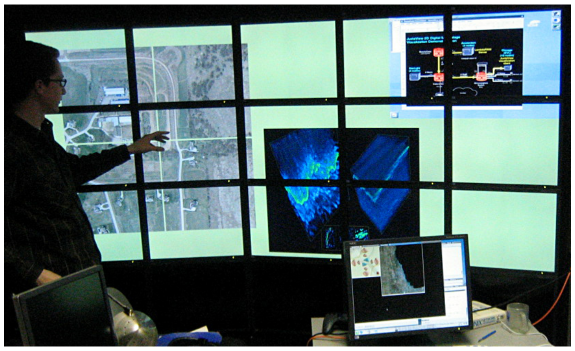
Figure 10. Current implementation of SAGE

5.1 Software

We use C++ as the main language for SAGE implementation. The Quanta [12] library is used for message passing between SAGE components, TeraVision's network protocol libraries are used for pixel streaming, and OpenGL/GLUT is used for displaying pixels on the tiled displays. The current implementation has been developed on Linux and ported to MacOS-X and MS windows. Any graphics application written in C/C++ requires minimal modifications: a few lines of code are added to link to the SAIL library. WxPython is used for SAGE GUI development.

5.2 Hardware Infrastructure

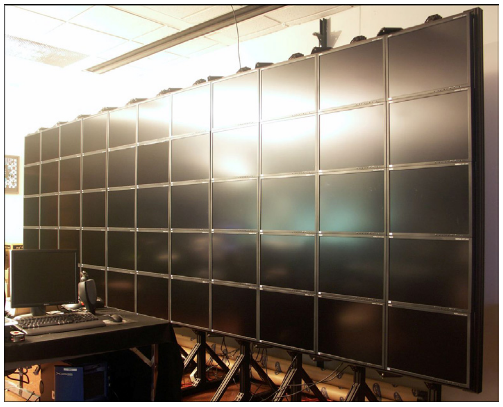
Figure 11. LambdaVision

The SAGE architecture consists of a number of rendering resources (from single desktop computers to clusters of local or distributed PCs capable of rendering graphics either with dedicated graphics hardware or software), connected over a high speed network to a scalable frame buffer, in our case the LambdaVision display (see Figure 11). LambdaVision pushes the concept of tiled-display technology [4,5,9] and anticipates the geosciences community's needs by coupling massive data storage (50TB of local storage) to a wall of 55 screens driven by 64-bit computers attached to each other by tens-of-gigabits of networking, for a total over 100Mpixels.

6 Experiments

We tested the SAGE performance on local area networks using 6-node display cluster and 6-node rendering cluster. Each node has dual AMD 64bit 2.4Ghz processors, Nvidia Quadro3000 graphics card, 4GB of main memory, and 1GigE network interface fully connected to each other through a gigabit network switch. We used a simple application, Checker, which keeps fetching images from main memory and streaming to the SAGE display and a typical OpenGL application, Atlantis, for this performance test. In the latter part of this section, we present a few real applications as examples of SAGE application.

6.1 Benchmarks

We designed a suite of benchmarks on local area networks. In all our experiment, total rendering resolution is equal to total display resolution. Table 3 shows the performance results of Checker running on one rendering node and one display node, while varying the frame size (RGB image).

		Checker		Checker w/o SAGE		Efficiency	
Resolution	Frame size (MB)	Frame rate (fps)	Bandwidth (Mbps)	Frame rate (fps)	Bandwidth (Mbps)	SAGE	Network
512x512	0.75	110	694	1359	8154	8.5%	73.8%
1024x512	1.50	60	760	670	8040	9.5%	80.8%
1024x1024	3.00	30	758	338	8112	9.3%	80.6%
2048x1024	6.00	15	757	168	8064	9.4%	80.4%
3200x1200	10.99	8	737	92	8086	9.1%	78.3%

Table 3. 'Checker' performance results

		Atlantis		Atlantis w/o SAGE		Efficiency	
Resolution	Frame size (MB)	Frame rate (fps)	Bandwidth (Mbps)	Frame rate (fps)	Bandwidth (Mbps)	SAGE	Network
512x512	0.75	85	510	96	576	88.5%	54.2%
1024x512	1.50	49	588	55	660	89.1%	62.5%
1024x1024	3.00	27	648	30	720	90.0%	68.9%
2048x1024	6.00	14	672	15	720	93.3%	71.4%
3200x1200	10.99	8	736	9	791	93.0%	78.2%

Table 4. 'Atlantis' performance results

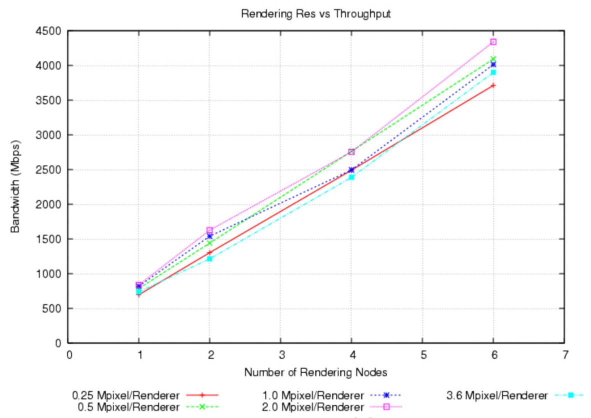

Figure 12. Throughput of Checker

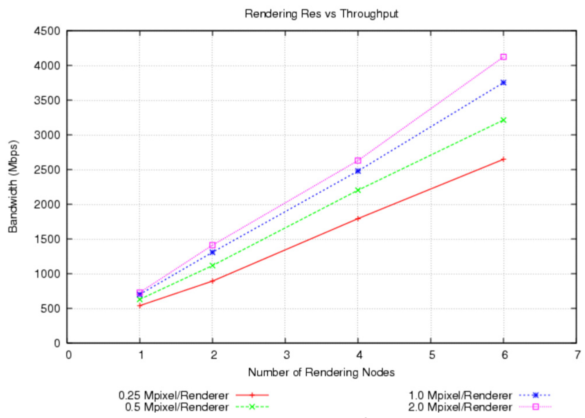

Figure 13. Throughput of Atlantis

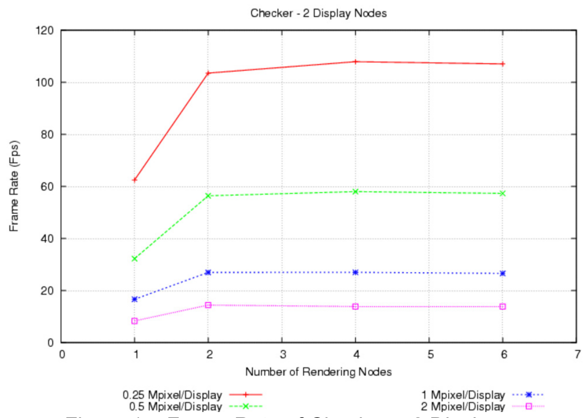

Figure 14. Frame Rate of Checker – 2 Displays

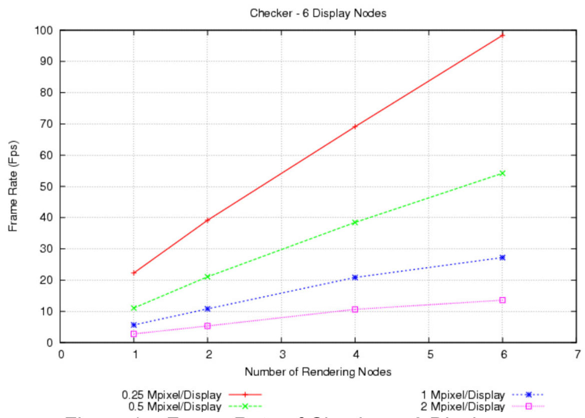

Figure 15. Frame Rate of Checker – 6 Displays

The result tells one SAGE stream can utilize up to 760Mbps network bandwidth which is 80.8% of 941Mbps - the maximum bandwidth monitored by the Iperf (a network benchmarking tool). The reason why the SAGE can't utilize the full network bandwidth is the SAGE overhead – memory copies inside SAIL, display overhead, and synchronization overhead. The results under "Checker without SAGE" shows the maximum performance numbers of the standalone version of Checker without network streaming. SAGE efficiency tells how much performance a SAGE application can achieve comparing to the application without SAGE. One gigabit networks limit the efficiency for Checker about 9%, but it means we can expect 80% of network bandwidth utilization when the network interface is upgraded to 10GigE.

Table 4 shows the performance results of Atlantis with the same experiment setup. Atlantis readbacks the pixels rendered in the frame buffer and streams them over networks using SAIL. Atlantis without SAGE does readback without network streaming. Unlike the case of Checker without SAGE, the bandwidth of Atlantis without SAGE is limited below 800Mbps due to readback overhead of OpenGL, and SAGE shows high efficiency(about 90%) for Atlantis. It means SAGE overhead is reasonably small if the networking bandwidth is not the bottleneck. The network efficiency is worse because the bottleneck is the OpenGL readback operation.

Figures 12 and 13 show the throughput of Checker and Atlantis on SAGE for various rendering resolution and number of rendering nodes. In both Checker and Atlantis, the throughput linearly increases over 4Gbps when we add new rendering nodes. It means we can scale the size of data we want to visualize by increasing the number of rendering nodes. As increasing the load of each node – increasing rendering resolution – Atlantis shows a distinct increase in the throughput, but Checker does not. The reason why Atlantis' throughput increases is the readback performance becomes better when we read larger chunk of pixels. For Checker, the throughput slightly increases with the rendering resolution up to a 2.0Mpixel resolution but decreases for a 3.6Mpixel resolution, because display overhead becomes the bottleneck for the 3.6Mpixel resolution to limit the throughput.

Figures 14 and 15 show we can get higher frame rates when we use more rendering nodes to display the same resolution of pixels. The data points on each curve have the same display resolution, but, as the number of rendering nodes increases, the load for each rendering node decreases so the frame rate increases. For six displays (Figure 15), the frame rates are linearly increases with the number of rendering nodes supporting a 12Mpixel resolution (2Mpixel x 6displays) at 14fps and a 6Mpixel resolution (1Mpixel x 6displays) at 28fps. For two displays (Figure 14), the frame rate does not increase for four and six rendering nodes because 2 gigabit network bandwidth limitation prevents rendering nodes from generating pixels at higher rate even though the load for each node decreases. We got similar results for Atlantis as shown in Figure 16, however, this time the frame rate increases for 2 displays and 4 rendering nodes because the bottleneck is the readback operation so the rendering performance has not reach the network bandwidth limit yet. When we use 6 rendering nodes and 6 displays, SAGE can run Atlantis at a 13fps frame rate and a 12Mpixel resolution.

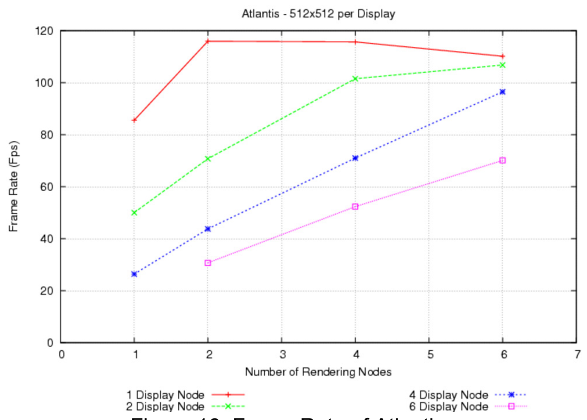

Figure 16. Frame Rate of Atlantis

6.2 Applications

To program SAGE-enabled applications, we used the very simple SAIL application programming interface. We describe briefly four applications which show the benefits of SAGE.

6.2.1 JuxtaView

JuxtaView [4] is a cluster-based application for viewing ultra-high-resolution images on scalable tiled displays. JuxtaView benefits from a new parallel computing and distributed memory approach for out-of-core montage visualization, so called LambdaRAM, a software-based network-level cache system. The data is distributed using LambdaRAM, on the memory of all nodes of a remote cluster. Aggressive pre-fetching schemes employed by LambdaRAM help to reduce latency involved in remote memory access. We ported JuxtaView to the SAGE environment, where each node of the cluster fetches images from LambdaRAM and generates a portion of the total image. The pixels are then streamed using SAGE to a high-resolution tiled display. JuxtaView is shown in Figure 10, on the left. Using SAGE, JuxtaView enables a user to interactively roam through potentially terabytes of distributed, spatially referenced image data sets such as those generated from electron microscopes, satellites and aerial photography. Through the use of a large amount of bandwidth, SAGE enables the domain scientist to bridge distributed resources including storage, rendering, and display clusters.

6.2.2 Vol-a-tile

Vol-a-Tile [13] is an interactive tool for exploring large volumetric data on scalable tiled displays. Hardware texture mapping and level-of-detail techniques provide interactivity. OptiStore is the data management server that provides the high performance I/O needed to stream data from storage to the nodes driving the tiled display over high-bandwidth photonic networks. It is designed to handle common data management operations, including loading data, maintaining meta-information about the data, such as dimension and scaling. In addition, OptiStore provides data processing capabilities, including run time gradient/histogram generation, sampling, and cropping by utilizing functionality in VTK. Network connectivity is provided through the QUANTA [12] network toolkit, which utilizes aggressive network protocols [10]. Since Vol-a-Tile is an OpenGL-based application, we added a pixel 'readback' step at the end of the rendering phase. These pixels are streamed for display using SAGE. This is a minimal modification to the source code, and could be applied to any OpenGL applications. This is an easy way to port various open-source applications to SAGE. Given the fill-limited performance characteristics of volume rendering applications (Vol-a-Tile uses OpenGL 3D textures and fragment shaders), this applications runs faster on a cluster of machines where each rendering node generates a sub-portion of the final image. SAGE recombines all the streams and provides the user with a coherent picture, which can be moved and scaled on any portion of the tiled display. Vol-a-Tile is shown in Figure 10, in the center.

6.2.3 OpenGL Wrapper

Numerous scientific applications and visualization packages are using the OpenGL API (OpenDX, VTK, or Paraview for instance). The success of Chromium and WireGL shows a need to support native OpenGL applications in binary mode (without source code modification). We developed an OpenGL wrapper library along the scheme used by WireGL: using a shared library mechanism, we only capture the calls to the 'glSwapBuffer' function and add a pixel readback step. The captured pixels are then streamed to SAGE. This is an efficient and extremely easy mode to port native OpenGL application to SAGE. The performance is sufficient to run 1280x1204 resolution application at an interactive framer rate. The new generation of PCI-express graphics card will even increase the performance several times.

6.2.4 VNC Viewer

Finally, we developed a Virtual Network Computer (VNC) protocol client that enables user to bring desktop content to the SAGE environment. Our VNC application is a regular VNC viewer program modified to serve as a proxy between a VNC server (of any size and pixel depth) and SAGE. Once the pixels are retrieved from the VNC server, the same pixels are given to the SAGE API for immediate display. The functionality is critical in a collaborative environment where each scientist with his/her laptop needs to share information (web browser, presentations…). SAGE supports any number of simultaneous VNC applications, making use of the large real estate offered by high-resolution tiled displays. VNC is shown in Figure 10, on the right.

7 Conclusions and Future Works

SAGE can support collaborative scientific visualizations at extremely high display resolution. Decoupling of rendering and display addresses heterogeneity and scalability. SAGE's dynamic pixel routing capability enables user to run multiple applications freely moving and resizing the application windows. In our experiment, we reached over 4.5Gpbs bandwidth and 14fps frame rate at a 12Mpixel resolution.

In the future, our high-bandwidth networks will be upgraded to 10gigabit networks : Infiniband on local networks and 10gigabit Ethernet on wide-area optical networks. When multicasting capability is added to SAGE, different endpoints in the collaboration may be seeing the same visualization on different devices with different display characteristics. Also, we are working on new streaming protocols and real-time compression which will improve SAGE's pixel streaming more reliable and fast over wide area networks. Eventually, we will extend SAGE to stream other graphics data types such as polygons, voxels, or progressive mesh so that SAGE can support wider range of applications and utilize networks, rendering and display resources more effectively.

8 Acknowledgments

We would like to thank numerous people involved in the development of SAGE: Alan Verlo, Lance Long, and Pat Hallihan provided us with diligent support to help debug and solve various network and system issues, Ratko Jagodic, Allan Spale, Julieta Aguilera for designing the GUI and logo for SAGE, valuable inputs by Nicholas Schwarz, Arun Rao, Charles Zhang, and Venkatram Vishwanath.

The Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago specializes in the design and development of high-resolution visualization and virtual-reality display systems, collaboration software for use on multi-gigabit networks, and advanced networking infrastructure. These projects are made possible by major funding from the National Science Foundation (NSF), awards CNS-0115809, CNS-0224306, CNS-0420477, SCI-9980480, SCI-0229642, SCI-9730202, SCI-0123399, ANI-0129527 and EAR-0218918, as well as the NSF Information Technology Research (ITR) cooperative agreement (SCI-0225642) to the University of California San Diego (UCSD) for "The OptIPuter" and the NSF Partnerships for Advanced Computational Infrastructure (PACI) cooperative agreement (SCI-9619019) to the National Computational Science Alliance. EVL also receives funding from the State of Illinois, General Motors Research, the Office of Naval Research on behalf of the Technology Research, Education, and Commercialization Center (TRECC), and Pacific Interface Inc. on behalf of NTT Optical Network Systems Laboratory in Japan. The GeoWall, GeoWall2, Personal GeoWall2 (PG2), and LambdaVision are trademarks of the Board of Trustees of the University of Illinois.

References

[1] Larry L. Smarr, Andrew A. Chien, Tom DeFanti, Jason Leigh, Philip M. Papadopoulos "The OptIPuter" Communications of the ACM, Volume 46, Issue 11, November 2003, pp. 58-67.

[2] Park, K., Renambot, L., Leigh, J. and Johnson, A., "The Impact of Display-rich Environments for Enhancing Task Parallelism and Group Awareness in Advanced Collaborative Environments", In Workshop on Advanced Collaboration Environments, June 22-24, 2003, Seattle, WA.

[3] Rajvikram Singh, Byungil Jeong, Luc Renambot, Andrew Johnson and Jason Leigh "TeraVision: a Distributed, Scalable, High Resolution Graphics Streaming System" , in the proceedings of IEEE Cluster 2004, San Diego, September 20-23, 2004.

[4] Naveen K. Krishnaprasad, Venkatram Vishwanath, Shalini Venkataraman, Arun G. Rao, Luc Renambot, Jason Leigh, Andrew E. Johnson, and Brian Davis "JuxtaView – a Tool for Interactive Visualization of Large Imagery on Scalable Tiled Displays" , in the proceedings of IEEE Cluster 2004, San Diego, September 20-23, 2004.

[5] W. Blanke, C. Bajaj, D. Fussell, and X. Zhang, "The Metabuffer: a Scalable Multiresolution Multidisplay 3-D Graphics System using Commodity Rendering Engines." Tr2000-16, University of Texas at Austin, February 2000.

[6] J. Leigh, L. Renambot, T.A. DeFanti, M.D. Brown, E. He, N.K. Krishnaprasad, J. Meerasa, A. Nayak, K. Park, R. Singh, S. Venkataraman, C. Zhang, D. Livingston, M. McLaughlin, "An Experimental OptIPuter Architecture for Data-Intensive Collaborative Visualization", 3rd Workshop on Advanced Collaborative Environments, Seattle, WA, June 2003

[7] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan, "Distributed rendering for scalable displays.", IEEE Super-computing 2000.

[8] G. Humphreys, M. Houston, Y. Ng, R. Frank, S. Ahern, P. Kirchner, and J. T. Klosowski, "Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters", Proceedings of SIGGRAPH 2002.

[9] Kenneth A. Perrine, Donald R. Jones, William R. Wiley "Parallel Graphics and Interactivity with the Scaleable Graphics Engine" Proceedings of the 2001 ACM/IEEE conference on Supercomputing.

[10] E. He, J. Leigh, O. Yu, T. A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer", IEEE Cluster Computing 2002, Chicago, IL, Sept 2002.

[11] D. Germans, H.J.W. Spoelder, L. Renambot, H. E. Bal, "VIRPI: A High-Level Toolkit for Interactive Scientific Visualization in Virtual Reality", Proc. Immersive Projection Technology/Eurographics Virtual Environments Workshop, Stuttgart, May 2001.

[12] Eric He, Javid Alimohideen, Josh Eliason, Naveen Krishnaprasad, Jason Leigh, Oliver Yu, Thomas A. DeFanti, "Quanta: A Toolkit for High Performance Data Delivery over Photonic Networks," Journal of Future Generation Computer Systems (FGCS), Elsevier Science Press, Volume 19-6, August 2003.

[13] Nicholas Schwarz, Shalini Venkataraman, Luc Renambot, Naveen Krishnaprasad, Venkatram Vishwanath, Jason Leigh, Andrew Johnson, Graham Kent, Atul Nayak, "Vol-a-Tile - a Tool for Interactive Exploration of Large Volumetric Data on Scalable Tiled Displays", IEEE Visualization 2004, Poster session, Austin, TX, October 2004.