High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment

Byungil Jeong, Luc Renambot, Ratko Jagodic, Rajvikram Singh, Julieta Aguilera, Andrew Johnson, Jason Leigh
Electronic Visualization Laboratory, University of Illinois at Chicago
{bijeong, luc, rjagodic, rsingh, julieta, aej, spiff}@evl.uic.edu

Abstract

The Scalable Adaptive Graphics Environment (SAGE) is specialized middleware for enabling data, high-definition video and extremely high-resolution graphics to be streamed in real-time from remotely distributed rendering and storage clusters to scalable display walls over ultra-high-speed networks. In this paper, we present the SAGE architecture, focusing on its dynamic graphics streaming capability. In the SAGE framework, multiple visualization applications can be streamed to large tiled displays and viewed at the same time. The application windows can be moved, resized and overlapped like any standard desktop window manager. Every window movement or resize operation requires dynamic and non-trivial reconfiguration of the involved graphics streams. This approach has been successfully shown to scale to support streaming on the LambdaVision 100 Megapixel display wall. SAGE is now being extended to support distance collaboration with multiple endpoints by streaming visualization to all the participants.

1. Introduction

A fundamental goal of visualization and collaboration on Grids interconnected by high-bandwidth deterministic networks (also called LambdaGrids [1,2]) is to enable users to collectively interpret enormous datasets in real-time at extremely high resolutions. An increasingly important model is to conduct the visualization using large pools of computing resources (such as clusters of powerful computers equipped with high-performance graphics processors) and streaming the results to the collaborating end-points. These end-points may range from PDAs all the way up to ultra-high-resolution display walls such as those built by stitching together dozens of LCD panels. The image streams shown on these display devices may consist of offline rendered movies as well as real-time visualizations and high-definition video. This approach provides significant advantages: firstly, the pooling of computing resources increases utilization, especially when they are cast as Grid services that can be combined with

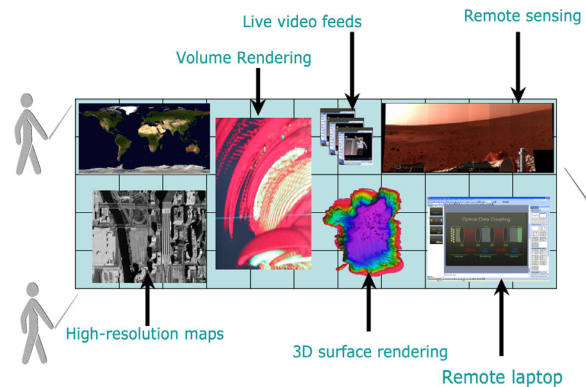


Figure 1. An Example of SAGE Session

other services to form a pipeline that could link large-scale data sources with visualization resources. Secondly, since networking is diminishing in cost at a rate exceeding that of computing and storage, it becomes more cost-effective for users to build low-cost, networked thin clients than to have to purchase and maintain their own rendering farms, storage repositories, etc.

We developed the Scalable Adaptive Graphics Environment (SAGE) to put this model into practice. SAGE allows the seamless display of various networked applications over ultra-high-resolution displays. Each visualization application (such as real-time or offline rendered visualizations, remote desktop, high-definition video streams, 2D maps etc.) streams its rendered pixels (or graphics primitives) to the virtual high-resolution frame buffer of SAGE, allowing user-definable window position and size on the displays (see Figure 1,9). Furthermore, SAGE enables users to freely move, resize and overlap the application windows by dynamically reconfiguring pixel streams.

SAGE has successfully supported our high-resolution display LambdaVision that is an 11x5 tiled display with a total resolution of 100 Megapixels. A high-resolution display like LambdaVision is essential to render complex geometric models without losing details. Even though a geometric model has a million triangles, if it is rendered into a window of 250,000 pixels (500x500), at most 25 percent of those triangles could contribute to the final image [10]. Also, geoscientists working with aerial and satellite imagery (365Kx365K pixels maps) and neurobiologists imaging the brain with montages consisting of thousands of pictures from high-resolution microscopes (4Kx4K pixels sensor) are good examples of SAGE and LambdaVision users.

In addition, SAGE has an important role in our

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SC2006 November 2006, Tampa, Florida, USA
0-7695-2700-0/06 $20.00 © 2006 IEEE

OptIPuter project [1,2]. It is a National Science Foundation funded project between the University of Illinois at Chicago and the University of California, San Diego to interconnect distributed storage, computing and visualization resources using a backplane constructed from LambdaGrids. SAGE is the graphics middleware of this project providing a unified environment to support various data exploration tools (visualization applications).

The main contributions of this paper are as follows:

I. It proposes the SAGE architecture and the procedure of dynamic pixel stream reconfiguration for multiple applications.
II. It shows the proposed architecture can support remote visualization applications scaling the resolution up to multi-ten Megapixels without losing interactivity.
III. It shows the proposed architecture can stream graphics data over 10gigabit wide area networks utilizing almost 90% of available bandwidth.

2. Related Work

There are several existing systems with parallel or remote rendering schemes related to SAGE. The simplest case of remote rendering uses remote desktop methods such as VNC, Microsoft Remote Desktop or Xmove. These 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. Access Grid [5] is a system that supports distributed collaborative interactions over Grids. Although it enables remote visualization sharing, the major focus of Access Grid lies in distributed meetings, conferences and collaborative work-sessions. Furthermore, the display resolution of remote desktop and Access Grid is limited to a single desktop resolution (at most 1600x1200 usually). On the other hand, SAGE can support 100 Megapixel display walls and include these systems in the SAGE framework by adding a simple SAGE API to them.

Perrine et al [9] and Klosowski et al [10] presented the merits of high-resolution display for various visualization applications. They used Scalable Graphics Engine (SGE) developed by IBM to drive their high-resolution displays. SGE is a hardware frame buffer for parallel computers. Disjointed pixel fragments are joined within the SGE frame buffer and displayed as a contiguous image [9]. SGE supports up to sixteen 1GigE inputs and can drive up to eight displays with double buffering for a total of 16 Megapixels. SAGE and SGE are similar in receiving graphics data from multiple rendering nodes and routing to high-resolution displays.

However, SAGE differs from SGE in that the former is a software approach that is much more flexible and scalable than the latter. SAGE does not require any special hardware. New network technology like 10GigE and new network protocols can be easily applied to SAGE. SGE, on the other hand, is bound to 1GigE inputs and an SGE-specific network protocol. There is no theoretical limitation in scaling the performance of SAGE by adding rendering and display nodes. In contrast, network bandwidth, the number of inputs and memory capacity all limit the performance of SGE.

There are several parallel rendering systems that can benefit from SAGE or SGE. WireGL [7] or parallel scene-

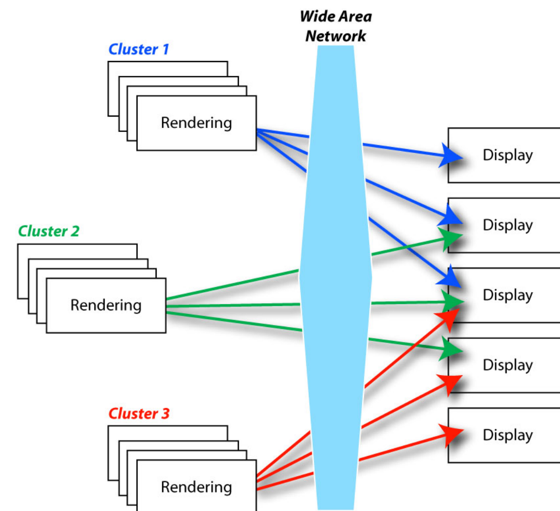


Figure 2. SAGE over Wide Area Network

graph rendering is a sort-first parallel rendering scheme from a single data source. This approach allows a single serial application to drive a tiled display by streaming graphics primitives that will be rendered in parallel on display nodes. However, it has limited data scalability due to its single data source bottleneck. Flexible scalable graphics systems such as Chromium [8] or Aura [11] are designed for distributing visualization to and from cluster driven tiled-displays. However, these systems enable a unique application at a time with a static layout on a tiled display. So they require a graphics streaming architecture such as SAGE or SGE to move, resize and overlap multiple application windows.

XDMX (Distributed Multi-head X11) is another system that can drive a tiled display. It is a front-end proxy X server that controls multiple back-end X servers to make up a unified large display [15]. XDMX also can support Chromium to display multiple applications on a tiled display. However, XDMX only supports non-parallel applications. This limits its scalability with large datasets.

The most unique feature of SAGE as compared with SGE, XDMX or Chromium, is the high-speed graphics streaming capability over wide area networks as shown in Figure 2. SAGE can use various streaming protocols designed for high-bandwidth and high-round-trip-time networks that are not considered in the streaming protocols of SGE and Chromium. We will discuss more about SAGE streaming protocols later. Moreover, we are extending SAGE to scalably support distance collaboration with multiple endpoints by streaming graphics to all the participating endpoints using various multicast approaches. In addition, SAGE considers mullions (borders) of each LCD panel of tiled displays when displaying application windows. Hence, the mullions appear to be placed on top of a large continuous image. This was also not considered in SGE and Chromium.

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 also can stream graphics data over wide area networks. However, it has a static application layout on a tiled display. It is suitable for streaming a single desktop to

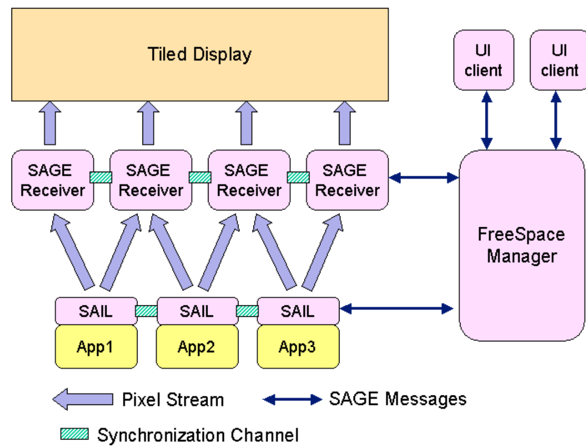


Figure 3. SAGE Components

a high-resolution tiled display, but unsuitable for supporting parallel applications or multiple instances of applications. To overcome these drawbacks, we developed SAGE.

3. SAGE Architecture

SAGE consists of various components: the Free Space Manager (FSManager), SAGE Application Interface Library (SAIL), SAGE Receivers, synchronization channel, and UI clients as shown in Figure 3. To dynamically reconfigure pixel streams, SAGE includes a component to control the whole procedure by communicating with all the other components. The FSManager is designed for this purpose. SAIL is the thin layer between an application and the high-speed network that is needed to capture the output pixels of the application and stream them to the display. A SAGE receiver on each display node receives and displays multiple pixel streams independently to allow multiple applications to be shown concurrently on the tiled display.

3.1. Free Space Manager (FSManager)

The Free Space Manager (FSManager) is the window manager for SAGE. This 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 Mega-pixel displays. The FSManager receives from UI clients various user commands such as application execution, window move, resizing or z-order change (overlapping windows) and then executes the commands by sending control messages to SAIL nodes or SAGE Receivers. To deliver control messages among SAGE components, we use our cross-platform adaptive networking toolkit QUANTA [12].

The FSManager collects and maintains the information needed for dynamic pixel stream reconfiguration from other SAGE components. Its most important role is to control the reconfiguration procedure based on the information whenever application windows are repositioned or resized. The detailed reconfiguration procedure will be described in the section 4.

Another important role of the FSManager is to execute applications based on user-defined configurations as shown in Table 1. ‘nodeNum’ is the number of rendering nodes. The line starting by ‘Init’ specifies the initial position and size of the application window. ‘exec’ is followed by an IP address and commands. The IP address specifies the machine on which the commands are executed, and the commands are directly used for executing the application. ‘nwProtocol’ specifies the name of the network protocol library to be used for pixel streaming. ‘syncMode’ specifies the synchronization mode of the application, on the rendering and display sides. Since multiple configurations are possible for each application, users can execute multiple instances of an application with different configurations by giving different configuration ID.

Table 1. SAGE Application Configuration

```
atlantis {
  configName TCP
  nodeNum 1
  Init 100 100  1000 1000
  exec 127.0.0.1 atlantis 0 127.0.0.1
  nwProtocol tvTcpModule.so
  syncMode 0

  configName UDP
  nodeNum 1
  Init 1100 1100 2000 2000
  exec 127.0.0.1 atlantis 0 127.0.0.1
  nwProtocol tvUdpModule.so
  syncMode 2
}
```

3.2. SAGE Application Interface Library (SAIL)

SAGE applications communicate with the FSManager and stream pixels to SAGE receivers through SAIL, which provides application programmers with a very simple interface to the SAGE framework. The SAGE API allows the programmers to describe pixel buffers and the position of the buffers in the application output image. The latter is needed when programming a 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 keeping the same total resolution) or to achieve higher resolution (increasing the number of processors while each generates the same amount of pixels). The only SAIL function other than 'init()' in the application is the call to the ‘swapBuffer()’ function. Due to this minimal API, any application with uncompressed pixel output can be easily ported to SAGE.

Table 2 shows an example of a minimal SAGE

Table 2. Minimal SAGE application

```
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) {
  draw( rgbBuffer );
  sageInf.swapBuffer( rgbBuffer );
}
```

Table 3. Tiled Display Configuration

```
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)
```

application. This application registers itself with the name 'render' to the FSManager. It is a sequential application, and the unique process will generate the whole image as described in the 'renderImageMap' data structure. The application function, 'draw()', generates 24-bit image in the RGB format onto the 'rgbBuffer' memory buffer where the pixels are laid out from bottom to top. The 'swapBuffer()' call delivers the image contained in the 'rgbBuffer' to the SAIL library. Then SAIL splits the image into sub-images and streams them to the proper SAGE Receivers based on the information given by the FSManager. It is completely transparent to users to partition and stream the images to the display using various protocols.

3.3. SAGE Receiver

A SAGE Receiver receives multiple pixel streams from SAIL nodes to drive the screens attached to each display node. The received streams may belong to different applications if multiple application windows are overlapped on the screens. This means the SAGE Receiver may have to update each application window at different times if the applications are independently synchronized at various refresh rates. We cannot use multi-threads to update the windows independently because conventional display control libraries like GLUT or SDL are not designed for multi-threading. Thus, we designed SAGE

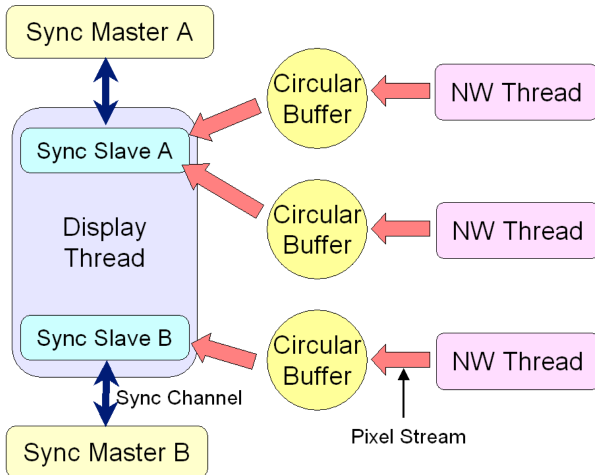


Figure 4. Architecture of SAGE Receiver

Table 4. Four Synchronization Modes

Sync Mode	Rendering Sync	Display Sync
0	On	On
1	On	Off
2	Off	On
3	Off	Off

Receiver as shown in Figure 4.

Each network thread receives a single pixel stream and stores the pixels in a circular buffer. If any synchronization slaves receive a synchronization signal, the display thread updates the screens with new images and start downloading the next frames from the circular buffers to the graphics card memory so that the next frames can be instantly updated to the screens by the next synchronization signal. The detailed synchronization procedure will be discussed in the next section.

A SAGE Receiver can drive multiple tiles (monitor screens). The layout of the tiled display is specified in a text configuration file that describes the association and the physical arrangement between the displays and the computers. Table 3 describes a 2x2 display driven by two computers. Here we assume LCD panels have uniform mullion (border of LCD panels) widths. PPI (pixel per inch) and mullion widths are used to calculate how many pixels should be hidden by mullions.

3.4. Synchronization Channels

When an application is displayed on a tiled display, each tile needs to be synchronously updated for the sub-images to be shown as one large consistent image. From our prior work on the TeraVision [3], we have learned that that not only display nodes but also rendering nodes need to be synchronized to yield better synchronization results in the case of a parallel application. So two synchronization channels were implemented: the display synchronization channel among SAGE Receivers and the rendering synchronization channel among SAIL nodes. If a parallel application synchronizes its output, the rendering synchronization is unnecessary. Also, users may want to turn off synchronization to remove the overhead in case synchronization is not critical for the application content. SAGE provides four synchronization modes as shown in Table 4. This allows users to freely turn on or off synchronization on each side per each application instance as shown in Table 1.

The procedure of rendering synchronization is very simple. The synchronization master thread resides on one of the SAIL nodes. Each SAIL node sends update signals to the synchronization master once it finishes transferring an image frame. Then, the synchronization master sends synchronization signals to all SAIL nodes after receiving update signals from all. After receiving the synchronization signal, SAIL nodes start to transfer new frames as soon as the frames are ready by the application.

For display synchronization, SAGE creates a synchronization master thread on one of SAGE Receivers and a synchronization slave object per each SAGE Receiver whenever an application is launched. The detailed synchronization procedure is as follows:

(1) The display thread waits until the images of frame N of the application A have arrived in the circular buffers.
(2) Once arrived, the display thread downloads the images into the graphics card memory.
(3) The synchronization slave A sends an update signal to its master (synchronization master A).
(4) The synchronization master A sends synchronization signals to all its slaves after receiving the update signal from all.
(5) When the synchronization slave A receives a synchronization signal, the display thread clears the screens and draws the frame N of the application A and the current frame of the application B.
(6) Repeat steps (1) – (5) for frame N+1

The same procedure is repeated for the application B in parallel. Even if only one application receives a synchronization signal, current images of the other application have to be redrawn, however, the overhead is minimal because drawing a rectangle with the texture already downloaded into the graphics card is extremely fast. The TCP out-of-band data channel is used for transferring synchronization signals in order to reduce the latency and to increase their priorities.

3.5. User Interaction

UI Clients can be a Graphical User Interface, text-based console or tracked devices [6], which send user commands to the FSManager and show the status of SAGE to the users. Any UI client can execute, shutdown, move, and resize SAGE applications in a manner very similar to a typical contemporary windowing system. Furthermore, UI clients can reside on any machine (laptop, tablet, desktop etc.) that can be connected to the FSManager over any network. Since SAGE is well suited for use in collaborative environments, several tools have been incorporated into the SAGE GUI to facilitate joint work. Users could, for example, have discussions and meetings in front of a tiled display where each of them is running an instance of the SAGE GUI connected to the same or even different displays. For basic communication, a chat capability and a list of users currently connected to the display are available via a server that manages user connections to every SAGE display. Every user can also be connected to multiple displays at the same time and control applications on any of them. This could prove especially useful when multiple sites are working together. At the end of a meeting, users could save the session and the state of the tiled display so that they can quickly resume their work at a later time.

4. Dynamic Pixel Stream Reconfiguration

We discuss here how SAGE dynamically reconfigures pixel streams. The procedure of dynamic pixel stream reconfiguration consists of the initial phase, the configuration phase and the streaming phase. The initial phase consists of static preparation procedures. The other two phases are being performed dynamically.

In the initial phase, an application is started and network connections are established.

(1) A SAGE application starts on a rendering cluster R.
(2) The application initializes a SAIL object per each node to be connected to the FSManager.

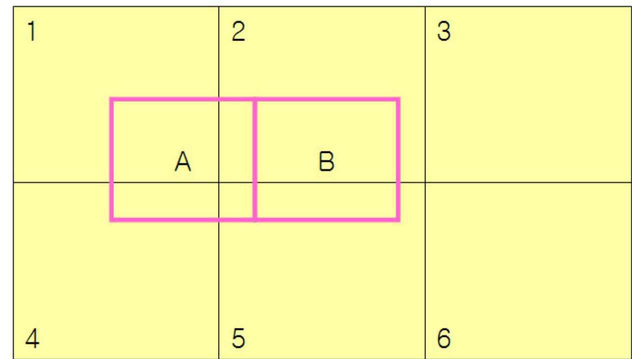


Figure 5. An Example of Application Layout

(3) The FSManager sends the SAIL objects the IP addresses and port numbers of the display nodes in cluster D.
(4) Every SAIL object on R is connected to all the display nodes in D so that |R| x |D| pairs of network connections are established (|R|,|D| is the number of nodes in each cluster).

In the configuration phase, the active connection set that will be actually used for the pixel streaming is decided according to user-defined application layout. The display and rendering nodes are dynamically reconfigured for the new streams. An example in Figure 5 is used for explaining the procedure. In this example, a parallel application is launched on rendering nodes A and B. The display cluster consisting of node 1, 2, 3, 4, 5 and 6 is driving a six-tile display.

(1) Overlap the application layout and the tiled display layout as in Figure 5.
(2) Generate the set of all sub-images of the application divided by tile borders. In the example, {A1, A2, A4, A5, B2, B5} (Mn: the intersection of the image rendered by M and the tile driven by n).
(3) Find the active connection set. Each sub-image generated in (2) corresponds to an active connection. In the example, {A-1, A-2, A-4, A-5, B-2, B-5} (M-N : network connection between M and N)
(4) Configure rendering nodes to split the application image into the sub-images generated in (2).
(5) Find the active display node set that will be actually used for showing application images. In the example, { 1, 2, 4, 5 }.
(6) Configure active display nodes to draw streamed images within the boundary of the sub-images generated in (2).

In the streaming phase, SAGE starts or resumes streaming using the configuration done in the previous phase. All streams are synchronized as described in 3.4.

(1) Grab the application image and split it as configured.
(2) Stream each sub-image over the corresponding active connection.
(3) Display the streamed images at the configured position on the tiles driven by the active display nodes.
(4) When users request window move or resizing, the application pixel streams are paused and go to the configuration phase.

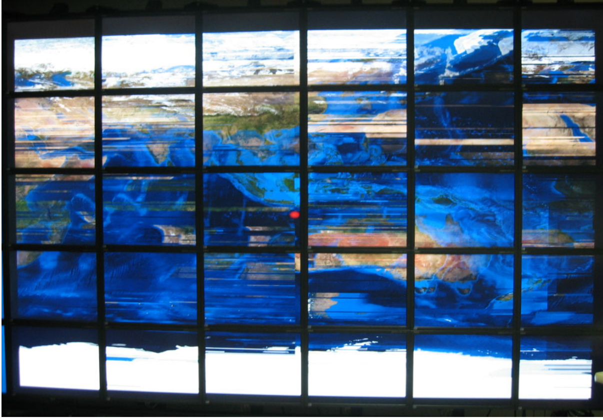


Figure 6. UDP Packet Loss Artifacts (30% Loss)

Ongoing streams need to be paused in order to be safely reconfigured. The FSManager pauses the streams by sending control messages to SAIL nodes and then starts the configuration phase.

5. Experimental Results

In this section, we will discuss the network streaming protocol we used for this experiment, the benchmarks, a real application test and the remote pixel streaming latency of SAGE. We used two 28-node LambdaVision cluster in San Diego (UCSD) and Chicago (UIC). A 10gigabit dedicated optical network connects the two clusters. Each cluster node has dual AMD 64bit 2.4Ghz processors, an Nvidia Quadro3000 graphics card, 4GB of main memory, and 1gigabit Ethernet (GigE) network interface fully connected to each other through a gigabit local network switch. We excluded other traffic on the network during our experiments. We used the standard MTU size (1.5KB) for local area tests and jumbo frames (9KB) for wide area tests.

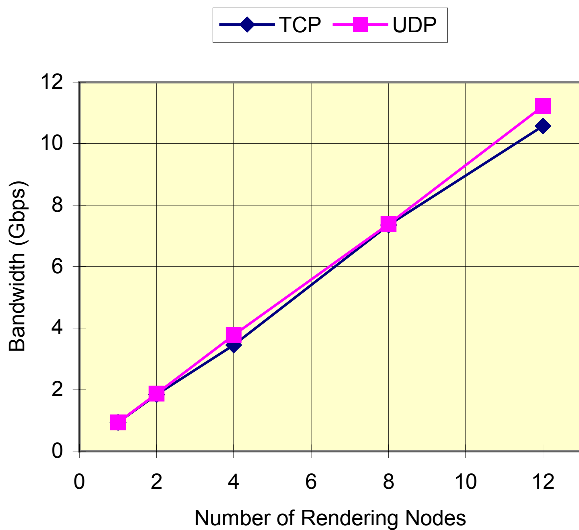
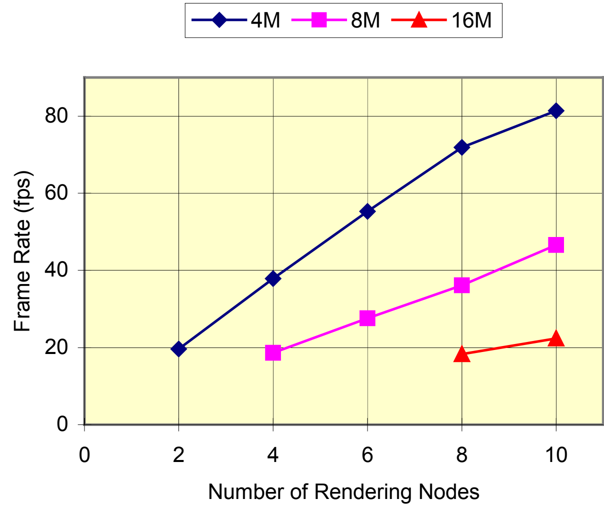
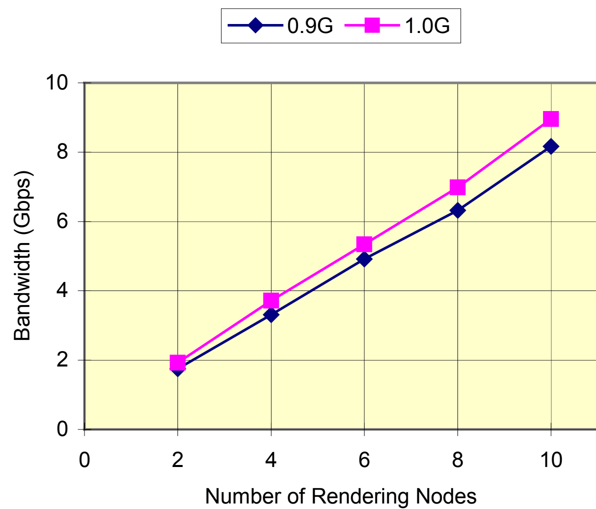


Figure 7. SAGE Throughput over LAN

5.1. Network Streaming Protocol

For SAGE we exploited the network streaming protocol of TeraVision [3], which provides simple interfaces to stream image frames over wide area networks. It has two network modules: one based on TCP and the other based on UDP. Since TCP is designed for short round-trip time networks, it works well over local area gigabit networks. However, it shows low or unstable performance over high-performance wide area networks with long round trip times, in our case 78ms. For this reason, we are using UDP network module for wide area network transfer. This module attaches a header for each UDP packet that includes the frame number and the position information of the pixel data stored in the packet. Even though several





Loss(%)	2	4	6	8	10
0.9G	0.01	0.00	0.00	0.00	0.32
1.0G	0.40	0.72	0.09	0.05	0.45

Figure 8. SAGE Performance over WAN and Packet Loss

packets are lost, pixel data contained in the following packets can be placed at the right position in the image frame using the information in the header. However, since the UDP module of TeraVision does not have data flow control, applications may blast pixel data exceeding available network bandwidth. It causes packet loss resulting in serious artifacts in the streamed image (see Figure 6). To fix this problem, we set the upper bound of the data transfer rate for each sender (rendering node) and extended the UDP module to control the data transfer rate so not to exceed the upper bound. In addition, we added a user command to change the upper bound interactively. By controlling the data transfer rate at an appropriate level, we were able to completely eliminate or reduce packet loss below 1.0% and the artifacts disappeared as shown in Figure 9.

5.2. Benchmarks

We wrote a benchmark application for SAGE, *Checker*, which keeps streaming white pixels stored in the main memory to the SAGE display. For the local area test, we ran Checker on 1 to 12 rendering nodes. Each rendering node streamed 1Mpixel images to the tiled display driven by 12 displaying nodes. In the case of UDP, we configured the upper bound of the transfer rate to 1Gbps for each sender (rendering node). Figure 7 shows the throughput increased to 11.2Gbps using UDP without packet loss with the number of rendering nodes increased and up to 10.5Gbps using TCP, sustaining the frame rates at 36~37fps (UDP) and 34~37fps (TCP). The pixel transfer rate of each node was 34~37Mpixel/sec. This result clearly shows the scalability of SAGE and its excellent network bandwidth utilization: 93.5% (UDP) and 90.3% (TCP) on average.

If only considering memory bandwidth, the peak performance of the SGE is 720Mpixel/sec [9]. However, to achieve this performance, the network bandwidth of each 1GigE input has to reach 45Mpixel/sec (1.08Gbps assuming RGB pixels are streamed) because the SGE has 16 inputs in total. This means the bottleneck of the SGE performance is in the network bandwidth utilization or the pixel transfer rate per rendering node. When the SGE was tested with the Chromium, the transfer rate was 12Mpixel/sec per rendering node [8], though, if measured separately, they showed the rate exceeding 23Mpixel/sec per node, which means the network utilization is around 50%. In contrast, SAGE was able to utilize the network over 90% (albeit on the different hardware used for rendering during the experiments).

Furthermore, SAGE showed the wide area performance consistent with the local area result. For the wide area performance test, we used a real SAGE application called MagicCarpet which is a cluster-based ultra-high-resolution image viewer for scalable tiled displays [14]. Figure 8 shows the frame rate and the network bandwidth of SAGE when MagicCarpet was rendering and streaming its pixels through SAIL from San Diego and Chicago. To evaluate the bandwidth and packet loss, we configured the application and SAIL as each rendering node streamed 1Mpixel images with the 0.9 and 1.0Gbps transfer rate upper bound. For the 1.0Gbps upper bound, we were able to achieve up to 8.95Gbps with at most 0.72% packet loss sustaining 36~37fps. For the 0.9Gbps upper bound, up to 8.16Gbps was achieved but with no or less packet loss than the former (see the table at the bottom of Figure 8). It is natural in that reducing the transfer rate increases packet

Figure 9. Typical SAGE Demonstration

intervals and decreases the load on the receivers. The former relieves network congestion and the latter reduces packet loss at the receivers. We also evaluated the display frame rates of SAGE while varying the resolution of the image data. This time we limited the transfer rate of each sender to 0.9Gbps. The curves in the graph above depict the frame rates when 4, 8 and 16Mpixel images are displayed respectively. This graph tells us that SAGE can scale the frame rate and the resolution by adding the appropriate number of rendering nodes. In other words, the frame rate linearly increases with the number of rendering nodes if we keep the image resolution the same, and the resolution which SAGE can support with the same frame rate linearly increases with the number of rendering nodes. When 20 and 38Mpixel images are streamed using 10 rendering nodes, we achieved 17.4 and 9.3fps respectively, but the packet loss increased 4.35% and 7.59% in spite of the transfer rate control. It seems the increase in resolution raised the load on each receiver, which increased the packet loss.

5.3 Real Application Test

Figure 9 shows four real SAGE applications used for a typical SAGE demonstration at EVL/UIC. MagicCarpet, on the right, was used to stream the Blue Marble dataset, created by NASA, from San Diego to Chicago using UDP. JuxtaView [4], in the middle, is a high-resolution image viewer that can migrate over huge image datasets such as 356Kx356K aerial photography. It was used to locally stream the aerial photography of downtown Chicago using TCP. Bitplayer, on the top left, is an uncompressed animation player developed by the National Center for Supercomputing Applications (NCSA). It was used to stream an animation of a tornado simulation from StarLight in downtown Chicago to UIC using UDP. Scalable Visualization Consumer (SVC), on the bottom left, developed by Gwang-ju Institute of Science and Technology (GIST) was used to locally stream HD camera live feed using TCP. For SVC, we used a 16-bit RGB pixel format rather than the 24-bit RGB format we used for the other applications.

Table 5 shows the sustained performance, total rendering resolution, and the number of rendering nodes used by these applications in this experiment. We set the transfer rate upper bound of MagicCarpet and Bitplayer to 0.8Gbps per sender to reduce packet loss to 0.31 and 0.38% respectively. When we increased the upper bound to

Table 5. Real Application Performance

Application	Bandwidth (Mbps)	Frame Rate (fps)	Rendering Resolution	Node Num
MagicCarpet	6737.3	33.7	3200x3000	10
JuxtaView	850.6	4.0	3200x3200	8
Bitplayer	516.8	11.3	1920x1080	1
SVC	538.4	24.9	1440x1080	1

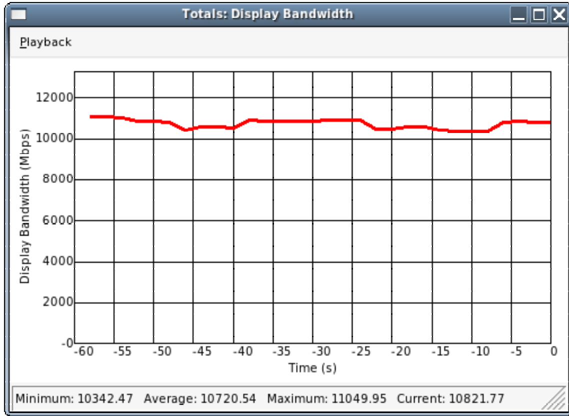


Figure 10. Network Bandwidth of SAGE Demo

1.0Gbps per node, the total network bandwidth was raised to 11Gbps as shown in the Figure 10. However, the packet loss of MagicCarpet and Bitplayer also increased to 1.5 and 9.1% respectively. This experiment shows that SAGE can support multiple remote and local applications using different network protocols at a time.

5.4 Remote Pixel Streaming Latency

We evaluated the pixel streaming latency of SAGE by running MagicCarpet again at San Diego and streaming its pixels to Chicago while varying the rendering resolution (X axis) and the number of rendering nodes (one or eight nodes). The average latencies and frame rates are plotted in Figure 11. The numbers shown in parenthesis in the legend are the number of rendering nodes used in each case. The annotated numbers at the top of the bars are the plotted values of the cases with eight rendering nodes. To check the latency, we sent a message from the FSManager to SAIL. SAIL attached a label on an image frame to be streamed when receiving the message. A SAGE Receiver reported to the FSManager upon receiving the image frame with the label. Then, the FSManager computed the latency by comparing the times of the message sent to SAIL and the message received from the SAGE Receiver. So the latency values depicted in Figure 11 included the message passing delay from Chicago to San Diego and pixel streaming latency from San Diego to Chicago. From the average 78ms round trip time for the network between San Diego and Chicago, we can estimate the message passing delay from Chicago to San Diego to be 39ms. Then, the actual pixel streaming latency between San Diego and Chicago ranged from 85ms to 311ms. These latency values and frame rates show that SAGE can support remote rendering applications without losing interactivity. In addition, Figure 11 shows the clear inverse correlation

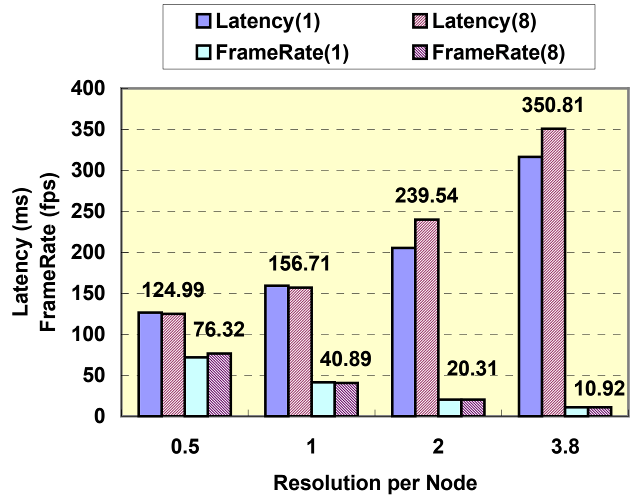


Figure 11. Remote Pixel Streaming Latency and Frame Rate

between the frame rate and latency. As the rendering resolution increased and the frame rate decreased, it takes more time to transfer an image frame. SAIL may also need to wait longer for the message from the FSManager before actually starting the transfer of the labeled frame. This explains the latency increase in Figure 11.

6. Future work

One of the main future goals of this research is to support distant collaboration with multiple end-points by streaming the same visualization at the same time. This enables groups of collaborators to share their visualizations, and see each other via the streaming of HD camera live feed. As windows on the tiled display are resized or repositioned on the walls, SAGE must reconfigure the multiple streams from the rendering source to the PC nodes that drive the displays. This problem becomes much more complex when SAGE is required to support independent window operations (such as reposition or scaling) at each display sites with different configurations.

A number of solutions are envisioned with various trade-offs. We are now designing a high-speed bridging system which receives pixel streams from rendering clusters to duplicate and split the streams for each end-point. In this case, each rendering node can stream full image frames without considering the window layouts and tiled display configurations of multiple end-points. This bridging system will be deployed on high-performance PCs equipped with 10gigabit network interfaces located in the middle of the collaboration sites. As more capacity is needed, more nodes can be added to sustain the desired throughput. Another approach we want to try is reliable layered multicast for tiled displays. Although a variety of techniques exist for supporting reliable multicast, high-bandwidth (on the order of tens of gigabits/s) and low-latency, reliable multicast is an unsolved problem and an active area of research within the Grid community [13]. It is a particularly challenging problem that the endpoints must distribute the multicast traffic over networked tiled displays.

Furthermore, we are working on new streaming protocols and real-time compression techniques that will improve the SAGE pixel streaming capacity. Eventually, we will extend SAGE to stream other graphics data types such as polygons, voxels or progressive mesh so that SAGE can support a wider range of applications, and utilize networks and computing resources more effectively.

7. Conclusion

In this paper, we showed that SAGE could support scientific visualization at an extremely high display resolution with an interactive frame rate. The dynamic pixel stream reconfiguration capability enabled the user to run multiple applications and to move and resize application windows freely. Our experiments showed the low latency, high throughput and scalability of SAGE, both over local-area and wide area networks. The peak performance of SAGE was 11.2Gbps on a local area network using UDP (without packet loss). Using a real-world application, we achieved 9.0Gbps over a 10Gbps dedicated link between San Diego and Chicago. Wide-area distributed visualization is now possible at the highest resolution while maintaining interactivity.

8. Acknowledgements

We would like to thank numerous people involved in the development of SAGE: Larry Smarr and Tom Defanti provided the endpoints at Calit2 in San Diego to perform the experiments. Javid Alimohideen, Allan Spale and Tae Jin Kim were involved in the SAGE UI development. Nicholas Schwarz, Arun Rao and Dmitri Svistula developed the applications for SAGE. Alan Verlo, Lance Long, and Pat Hallihan provided us with diligent support to help debug and solve various network and system issues. We also want to give special thanks to Laura Wolf for editing this paper. The valuable input by Xi Wang, Eric He, Cole Krumbholz, Charles Zhang and Venkatram Vishwanath are also greatly appreciated.

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.

9. References

[1] L. L. Smarr, A. A. Chien, T. DeFanti, J. Leigh, and P. M. Papadopoulos, "The OptIPuter," *Communications of the ACM*, volume 46, issue 11, pp. 58-67, November 2003.

[2] J. Leigh et al, "An experimental OptIPuter architecture for data-Intensive collaborative visualization," in *Third Workshop on Advanced Collaborative Environments*, 2003.

[3] R. Singh, B. Jeong, L. Renambot, A. Johnson, and J. Leigh, "TeraVision: a distributed, scalable, high resolution graphics streaming system," in *Proceedings of IEEE Cluster*, 2004.

[4] N. K. Krishnaprasad et al, "JuxtaView – a tool for interactive visualization of large imagery on scalable tiled displays," in *Proceedings of IEEE Cluster*, 2004.

[5] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi, "Access Grid: immersive group-to-group collaborative visualization," in *Proceedings of Fourth International Immersive Projection Technology Workshop*, 2000.

[6] C. Krumbholz, J. Leigh, A. Johnson, L. Renambot, and R. Kooima, "Lambda table: high resolution tiled display table for interacting with large visualizations," in *Proceedings of Fifth Workshop on Advanced Collaborative Environments*, 2005.

[7] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan, "Distributed rendering for scalable displays," in *Proceedings of ACM/IEEE Conference on Supercomputing*, 2000.

[8] G. Humphreys et al, "Chromium: a stream-processing framework for interactive rendering on clusters," in *Proceedings of SIGGRAPH*, 2002.

[9] K. A. Perrine, D. R. Jones, and W. R. Wiley, "Parallel graphics and interactivity with the scaleable graphics engine," in *Proceedings of ACM/IEEE Conference on Supercomputing*, 2001.

[10] J. T. Klosowski, P. Kirchner, J. Valuyeva, G. Abram, C. Morris, R. Wolfe, and T. Jackman, "Deep view: high-resolution reality," *IEEE Computer Graphics and Applications*, volume 22, issue 3, pp. 12–15, May/June 2002.

[11] D. Germans, H.J.W. Spoelder, L. Renambot, and H. E. Bal, "VIRPI: a high-level toolkit for interactive scientific visualization in virtual reality," in *Proceedings of Immersive Projection Technology/Eurographics Virtual Environments Workshop*, 2001.

[12] E. He et al, "Quanta: a toolkit for high performance data delivery over photonic networks," *Journal of Future Generation Computer Systems*, volume 19, issue 6, pp. 919-933, August 2003.

[13] M. D. Burger, T. Kielmann, and H. E. Bal, "Balanced multicasting: high-throughput communication for Grid applications," in *Proceedings of ACM/IEEE Conference on Supercomputing*, 2005.

[14] D. Svistula, J. Leigh, A. Johnson, and P. Morin, "MagicCarpet: a high-resolution image viewer for tiled displays," http://www.evl.uic.edu/cavern/mc.

[15] "Distributed multi-head X project," http://dmx.sourceforge.net/.