Preliminary Prospectus

Visualcasting - Scalable Real-Time Image Distribution in Ultra-High Resolution Display Environments

Byungil Jeong

bijeong@evl.uic.edu

July 15, 2006

Main Advisor:

Jason Leigh

Committee Members:

Jason Leigh

Andrew Johnson

Luc Renambot

Thomas A. DeFanti

Erik C. Hofer

1 Executive Summary

The Scalable Adaptive Graphics Environment (SAGE) is a specialized middleware that enables real-time streaming of extremely high-resolution graphics and high-definition video from remotely distributed rendering and storage clusters to scalable display walls over ultra high-speed networks. I propose Visualcasting in order to extend SAGE to support distant collaboration between multiple endpoints by streaming visualization to all participants.

In the SAGE framework, each visualization application streams its rendered pixels to the virtual high-resolution frame buffer of SAGE, allowing users to freely move, resize and overlap the application windows on the display. Every window movement or resize operation requires dynamic and non-trivial reconfigurations of the involved graphics streams. These reconfigurations become even more complex when SAGE is required to support multiple collaboration endpoints with different tiled display configurations and application window layouts.

Visualcasting is a scalable real-time image distribution service to address this problem. Visualcasting includes a high-speed bridging system which receives pixel streams from rendering clusters to duplicate and distribute pixels to each end-point. This approach will be compared with the traditional router-based multicasting approach and the reliable layered multicasting approach. Several issues in optimizing Visualcasting will be also addressed.

Intellectual Merit:

Visualcasting addresses the problem of reliable graphics multicasting for tiled displays. Although a variety of techniques exist for supporting reliable multicasting, reliably multicasting graphics data onto remote tiled displays with different display configurations is still an unsolved challenging problem. Furthermore, Visualcasting addresses the heterogeneity and scalability problems. Since each Visualcasting endpoint has heterogeneous network and display capacity, the Visualcasting service needs to adapt the resolution or quality of images to the capacity of each endpoint to prevent it from being overloaded. Also, the Visualcasting service should be scalable based on the number of endpoints and applications.

Broader Impact:

Visualcasting enables SAGE to scalably multicast real-time, high-definition video and ultra-definition visualizations across globally distributed research centers. This capability will demonstrate new ways in which high-performance networking and visualization can be used in a broad range of research, academic and commercial applications. Furthermore, understanding the requirements, benefits and limits of Visualcasting and alternative approaches will provide valuable input into future Internet system design.

2 Table of Contents

3 Introduction

In a decade's time, high-performance computing has proven its value in Science, Homeland Security, Medicine, Engineering, Education, and Filmmaking. These data-intensive domains rely on Grid technology to process the terabytes of raw data so to produce meaningful insight, often enabled by high-quality visualizations. As research and development become increasingly global and multidisciplinary, the need for a computing infrastructure to support collaborative work among distributed users has grown dramatically [Leigh06].

Up until the last five years, however, the predominant model for supporting data-intensive collaborative science and engineering involved replicating expensive systems at each collaboration site to visualize the highly distilled computation results from remote computing resources, because bandwidth was a scarce commodity. Recently, it has become viable for scientists to connect to ultra-high-speed networks more cheaply than they maintain large computing, storage and visualization systems, because the rate of decline in the cost of bandwidth far exceeding that of computing and storage [Stix01]. This is the fundamental premise behind the concept of shared cyber-infrastructure, and is being deeply researched by the international partners of the OptIPuter project [Smarr03, Leigh03].

In this context, an increasingly important visualization model is to conduct visualization using large pools of computing resources (such as clusters of powerful computers equipped with high-performance graphics processors) and to stream the results to the collaborating end-points. These end-points may range from PDAs to ultra-high-resolution display walls such as those built from 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 (see Figure 1). This enables users to collectively interpret enormous data-sets in real-time at extremely high resolutions.

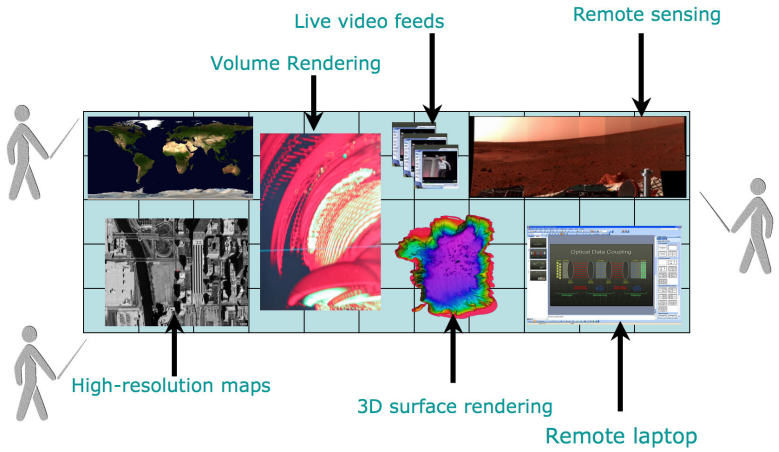


Figure 1. Collaborative Visualization on Tiled Display

A fundamental requirement of these high-resolution collaborative visualization systems is an ability to broadcast or multicast visualizations to all collaborating sites so that all participants can simultaneously see and interact with the data. Multicasting in these high-resolution environments poses a significant challenge because potentially tens of gigabits of network bandwidth are needed to support collaborative visualization. This thesis will investigate this problem in detail and propose a scheme called Visualcasting that is specifically designed to provide the kind of image multicasting service needed for ultra-definition visualization. This service will be implemented on top of SAGE.

SAGE enables multiple visualization applications to be streamed to large tiled displays and viewed at the same time (see Figure 1, 2). The application windows can be moved, resized and overlapped like any standard desktop window manager. This approach can be scaled to support streaming on the LambdaVision that is an 11x5 tiled display with a total resolution of 100 Mega-pixels (see Figure 2). SAGE and LambdaVision can greatly support 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).

Two distributed visualization models are supported. One involves running visualization applications at multiple remote rendering sites and streaming their pixels to one big tiled display (see Figure 3). The other supports distance collaboration between multiple endpoints by streaming the same visualization to each endpoint simultaneously (see Figure 4). The former model is already implemented as a SAGE prototype. Visualcasting extends SAGE to support the latter or the combination of both models.

Figure 2. LambdaVision driven by SAGE

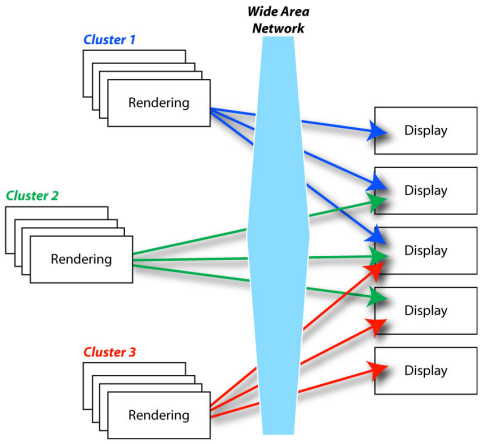


Figure 3. Multiple Rendering Sites

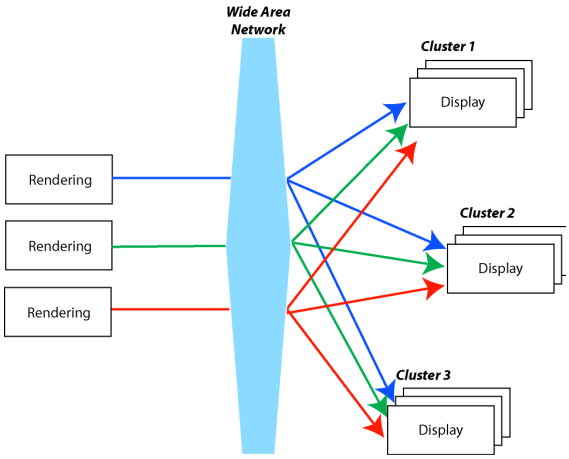


Figure 4. Multiple Display Sites

As windows on the tiled display are resized or repositioned on the walls, SAGE must reconfigure multiple streams from rendering sources to the display nodes. This problem becomes much more complex when SAGE is required to support independent window operations (such as reposition or scaling) at each display sites having different display configuration. For the first model, illustrated in the Figure 3, each sender can calculate the final destination (display node) of each pixel based on the application window layout on the tiled display. However, this approach is not scalable to the Visualcasting model in Figure 4, because the required network bandwidth and computation keeps increasing with the number of display sites until the senders become overloaded.

The approach to solve this problem is a high-speed bridging system called SAGE Bridge which duplicates and splits pixel streams received from rendering clusters for each end-point. This allows each rendering node to stream full image frames without considering the window layouts and tiled display configurations of multiple end-points. The SAGE Bridge will be deployed on high-performance PCs equipped with 10gigabit network interfaces. When more capacity is needed, the SAGE Bridge can dynamically allocate additional nodes to sustain the desired throughput. This approach will be compared with the traditional IP multicast [Deering91] approach.

Another important issue in Visualcasting is dealing with the heterogeneity of display resolutions and network bandwidth of the endpoints. Some endpoints may have a 100 megapixel tiled display and multi-ten gigabit network bandwidth which are enough to support multi-ten megapixel visualizations. Such a high resolution visualization, however, will not fit into other end-points with a small tiled display and only a gigabit network. To resolve this issue, the SAGE Bridge has to control applications to adapt the visualization to an appropriate resolution, or the SAGE Bridge itself must be able to adapt the resolution of the pixel streams to each end-point. This requires an appropriate application interface that dynamically

changes the visualization resolution and the effective real-time pixel stream compression and sampling techniques. Layered multicast [McCanne96] is an alternative to be compared with this approach.

In addition, two issues to enhance Visualcasting performance are discussed in this thesis: the investigation of optimal pixel data block size and the design of an optimal interface between SAIL and network protocols. Pixel data block size affects network streaming performance and pixel download performance to graphics memory. On the other hand, it decides the flexibility in pixel data distribution over tiled displays. Determining the requirements and options of the interface between SAIL and network protocols is critical in designing the interface. I will evaluate network streaming performance with varying options in order to find an optimal interface.

The major contribution of this research will include the following:

1. Proposing Visualcasting as a scalable solution for real-time image multicasting for tiled displays supporting distant collaboration with multiple end-points.
2. Addressing the heterogeneity problem among collaboration end-points.
3. Identification of the metrics for Visualcasting performance and evaluation of a few possible approaches to Visualcasting.
4. Approach selection strategy based on the given situation, such as available network bandwidth, desirable resolution and affordable latency.

The fundamental research questions are:

1. How to arbitrarily scale simultaneous data distribution to multiple receivers?
2. What happens if the receivers are heterogeneous? Are any special considerations needed?
3. What parameters define ‘arbitrarily’ and how do those parameters affect the distribution performance?
4. If multiple approaches are possible, how to decide the most appropriate approach based on the parameters?

In the following sections, I will discuss latest prototype of SAGE, other related research, the details of Visualcasting and the plans for completing this thesis.

3.1 Prior Accomplishments

I implemented a prototype of SAGE based on the distributed visualization model (multiple rendering sites) in Figure 3. The feasibility of this visualization model was successfully demonstrated in iGrid2005 and Supercomputing 2005 using the prototype. It has been successfully shown to scale to support streaming on the LambdaVision 100-Megapixel display wall. The architecture, dynamic pixel stream reconfiguration, pixel block based streaming and results of the prototype are to be discussed in this section.

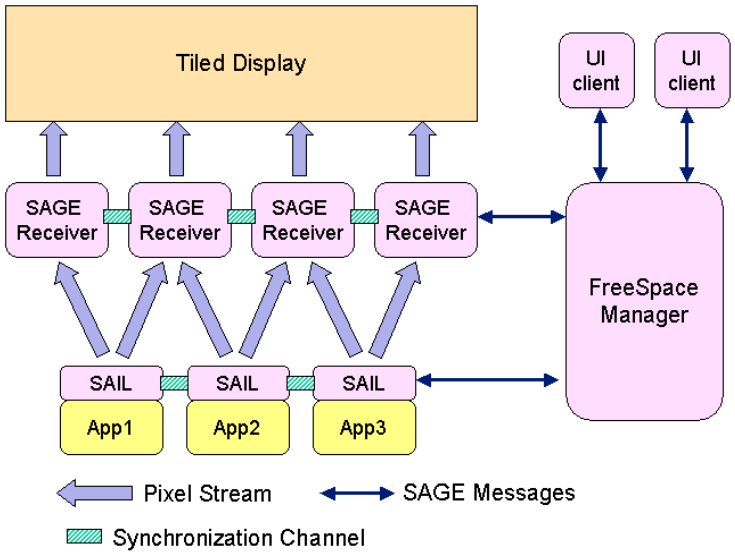


Figure 5. SAGE components

3.1.1 The architecture of the SAGE Prototype

The SAGE prototype consists of various components: Free Space Manager (FSManager), SAGE Application Interface Library (SAIL), SAGE Receiver, synchronization channel, and SAGE UI as shown in Figure 5. The FSManager communicates with all the components to control the procedure of dynamic pixel stream reconfiguration. SAIL is a thin layer between an application and the network that captures the output pixels of an application and stream them to the displays. A SAGE Receiver on each display node enables the display of multiple applications at the same time by receiving independent pixel streams from each application.

3.1.1.1 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 a large tiled display. The FSManager receives various user commands from UI clients 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 the SAGE Receivers. QUANTA, a cross-platform adaptive networking toolkit [He03], is used for delivering control messages to SAGE components.

The FSManager collects and maintains the information needed for dynamic pixel stream reconfiguration from other SAGE components. The most important role of the FSManager is to control the reconfiguration procedure based on the information whenever windows are repositioned or resized. Another important role

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
}
```

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)
  sageInf.swapBuffer( rgbBuffer );
```

of the FSManager is to execute applications based on user-defined configurations, including the number of rendering nodes, the initial position and the window size, application parameters, the network protocol for pixel streaming and the synchronization mode. Table 1 shows an example of a SAGE application configuration. Users can execute multiple instances of an application with different configurations by specifying different configuration entries.

3.1.1.2 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 programmer 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 ‘sageSwapBuffer()’. 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 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 will generate 24-bit RGB images in the ‘rgbBuffer’ where the pixels are laid out from bottom to top. After the initialization phase,

the application sends 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. Partitioning and streaming of the images to the display using various protocols is completely transparent to users.

3.1.1.3 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 display. To keep each application synchronized independently, the SAGE Receiver should update each application window at different times. But 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. Hence, we designed the SAGE Receiver as shown in Figure 6.

Each network thread receives a single pixel stream and store pixels in a circular buffer. If any synchronization slaves receive a signal, the display thread updates the screen with new images and downloads the next frames from the circular buffer to graphics card memory so that the next frames can be instantly displayed on 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 which describes the association and the physical arrangement between displays and computers. Table 3 describes a 2x2 display driven by two computers. Here we assume LCD panels have

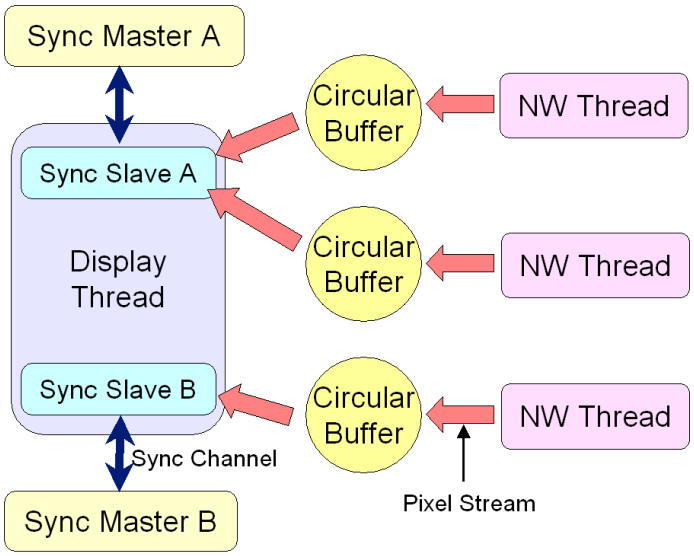


Figure 6. Architecture of SAGE Receiver

uniform mullion (border of LCD panels) widths. PPI (pixels per inch) and mullion widths are used to calculate how many pixels should be hidden by the borders of screens.

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)
```

3.1.1.4 Synchronization channels

When an application is displayed on a tiled display, each tile needs to be synchronously updated in order for the sub-images to be shown as one large consistent image. From TeraVision [Singh04], EVL researchers learned that not only display nodes but also rendering nodes need to be synchronized to yield better synchronization results for parallel applications. Hence, two synchronization channels were implemented: the display synchronization channel between SAGE Receivers and the rendering synchronization channel between SAIL nodes. The rendering synchronization is unnecessary, if a parallel application synchronizes its outputs. Also, users may want to turn off synchronization in order to remove the overhead if synchronization is not critical for the application. SAGE provides four synchronization modes as shown in Table 4. This allows users to freely turn on or off synchronization on each side per application instance.

Table 4. Four synchronization modes

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

Rendering synchronization is a very simple procedure. The synchronization master thread resides on one of the SAIL nodes. Each SAIL node sends an update signal to the synchronization master when it finishes transferring an image frame. Once the synchronization master receives an update signal from every node, it sends then a synchronization signal to each SAIL node. Each SAIL node starts to transfer new frames as soon as the frames are provided by the application.

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

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

In parallel the same procedure is repeated for application B. Even if only one application receives a synchronization signal, current images of the other applications have to be redrawn. The redrawing 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 latency and to increase priority.

3.1.1.5 User Interface

UI Clients can be a Graphical User Interface, text-based console or tracked devices [Krumbholz05], which send user commands to the Free Space Manager 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.) connected to the Free Space Manager over any network. Since SAGE is well suited for use in collaborative environments, several tools have been incorporated into the SAGE GUI to facilitate collaborative work. Users could, for example, have discussions and meetings in front of a tiled display where each user 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 from a server managing user connections to every SAGE display. Every user could 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. SAGE UI has been developed by other EVL researchers.

3.1.2 Dynamic Pixel Stream Reconfiguration

Multiple pixel streams of SAGE must be dynamically reconfigured whenever users order window resize or reposition. This reconfiguration procedure consists of three phases: the initial phase, the configuration

phase and the streaming phase. 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 connected to the FSManager.
(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 used for pixel streaming is decided, and the display and rendering nodes are reconfigured for streaming. Figure 7 explains the procedure with an example. A parallel application is launched on rendering nodes A and B. The display cluster consisting of nodes 1, 2, 3, 4, 5 and 6 is driving a six-tile display.

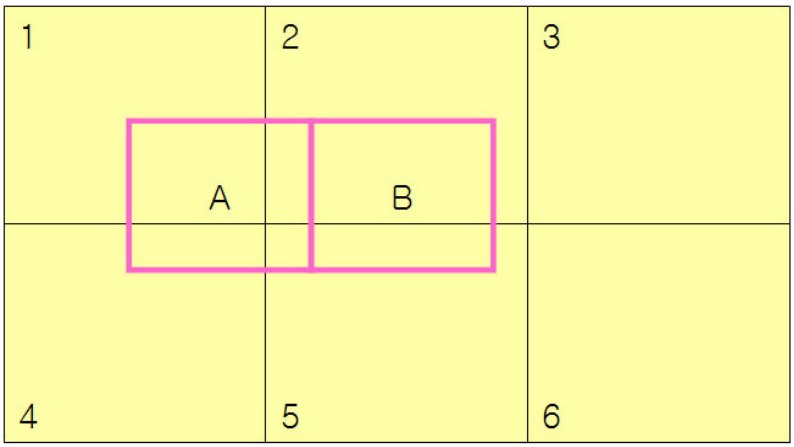


Figure 7. An example of application layout

(1) Overlap the application layout and the tiled display layout as in Figure 7.
(2) Generate the set of all sub-images of the application divided by tile borders. In the example, {A_1, A_2, A_4, A_5, B_2, B_5} (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 which will be 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 determined in the previous phase. All streams are synchronized as described in 3.1.1.4.

(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 perform window operations, the application pixel streams are paused and go to the configuration phase.

3.1.3 Pixel Block Based Streaming

The prototype described so far streams application images frame by frame. Each image frame is split into sub-images according to the application layout on the tiled display. The generated sub-images are copied to a network buffer and streamed to a tile. Since the generation of the sub-images closely depends on the application layout on the tiled display, this approach is unsuitable for Visualcasting where an application may have different application layouts on tiled displays at each endpoint. Pixel block based streaming was introduced to overcome this problem.

In the new approach, adjacent pixels in an application image are grouped as a pixel block and streamed over networks. Pixel blocks are regularly sized sub-images irrespective to tile configuration and application window layout. This allows SAGE to deliver the same pixel blocks to every Visualcasting endpoint and to independently distribute them over tiled displays at each endpoint. Figure 8 illustrates pixel block streaming compared with image frame streaming for a tiled display.

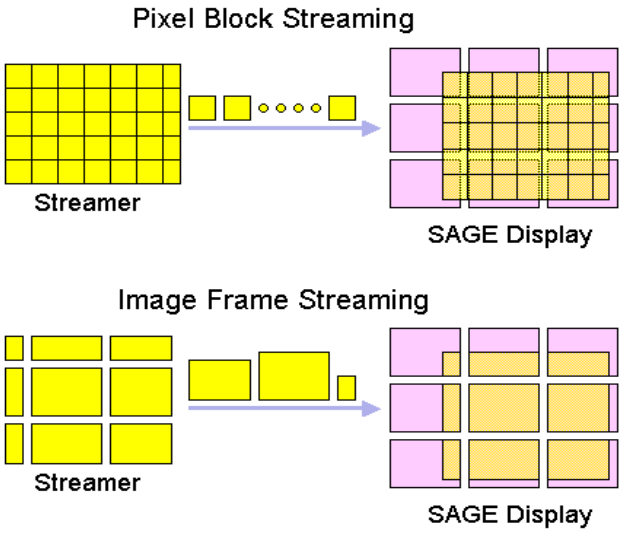


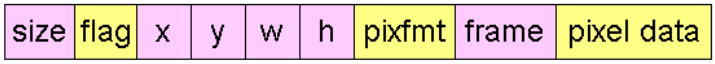
Figure 8. Pixel Data Streaming

Other advantages lie in image buffer management and non-rectangular image streaming. Image resizing requires freeing and reallocating buffers in the image frame streaming. On the other hand, it changes the number of pixel blocks in the new approach. Moreover, it is now possible to stream non-rectangular images. Selectively streaming a subset of pixel blocks generated from a rectangular image and combining the streamed pixel blocks at receivers can result in streaming a non-rectangular image.

The configuration phase of the dynamic pixel streaming reconfiguration procedure was executed by control messages sent from the FSManager to SAIL objects and SAGE Receivers. These control messages were not synchronized with pixel streams. Asynchronous reconfiguration of active streams might cause system crashes. Hence, the streams had to be paused until the reconfiguration was done safely, but this increased the latency of the reconfiguration by up to two seconds.

Pixel block streaming was designed to incorporate stream control information into pixel block streams so that SAGE can reconfigure the streams without stopping them. When the stream control message from the FSManager arrives, SAIL objects wait until pixel blocks from the current image frame are completely transferred and stream control blocks to SAGE Receivers. Then, SAIL objects reconfigure themselves and resume pixel streaming. SAGE Receivers are reconfigured not by the messages from the FSManager but by the control blocks inserted into pixel streams. In this way, both sides can be reconfigured synchronously and safely.

Figure 9 shows the format of the SAGE block. It consists of a 36-byte header and pixel data of variable size. The header contains the total size of a block in the beginning so that SAGE Receivers can get the size information first and then read the rest of the block. The flag field specifies the function of the block as listed in Figure 9. The fields x, y, width and height store the lower left corner and size of the pixel block in the whole image. The frame ID is used for synchronizing pixel blocks to be drawn in a single image.



- x, y : the location pixel block in the image frame
- w, h : width and height of pixel block
- pixfmt, frame : pixel color format, frame ID
- Flags

0001	Pixel Block	1002	Update Block
1000	Init Block	1003	Stop Block
1001	Final Block	1004	Clear Block

Figure 9. SAGE block format

Pixel blocks (flag=0001) deliver pixel data, and control information for pixel streaming is delivered by control blocks including:

(1) Init Block, flag=1000: sent before streaming begins to have SAGE Receivers ready to receive pixel block streams. It initializes the screen layout, the color format and the displaying textures of the pixel stream.
(2) Final Block, flag=1001: informing SAGE receivers that an image frame is completely transferred.
(3) Update Block, flag=1002: forcing SAGE Receivers to asynchronously update screens after window move or resizing operations.
(4) Stop block, flag=1003: informing SAGE Receivers that the pixel stream will be stopped due to a window move or resize.
(5) Clear Block, flag=1004: used for closing pixel streams before an application shutdown.

3.1.4 Achieved Results

The SAGE prototype could support scientific visualization at extremely high display resolutions with interactive frame rates. The dynamic pixel stream reconfiguration capability enabled users to run multiple applications and to move and resize application windows freely. Experimental results showed low latency, high throughput and scalability of SAGE over local-area and wide-area networks [Jeong06]. The peak performance of SAGE was 11.2Gbps on a local area network using UDP (without packet loss). 9.0Gbps was achieved using a real-world application from San Diego to Chicago over a 10Gbps dedicated link. The prototype successfully demonstrated wide-area distributed visualization at the highest resolution possible while maintaining interactivity. It was enhanced by pixel block based streaming which is well suited for Visualcasting. This new approach reduced dynamic pixel stream reconfiguration latency and increased flexibility in pixel streaming. The success of the SAGE prototype gives much confidence in this approach.

3.2 Prior 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. They were designed to transmit 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. AccessGrid [Childers00] is a system that supports distributed collaborative interactions over Grids. Although it enables remote visualization sharing, the major focus of AccessGrid lies in distributed meetings, conferences and collaborative work-sessions. Furthermore, the display resolution of remote desktops and AccessGrid 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 and Klosowski et al presented the merits of high-resolution display for various visualization applications [Perrine01], [Klosowski02]. They used the Scalable Graphics Engine (SGE) developed by IBM to drive their high-resolution displays. SGE is a hardware frame buffer for parallel computers. Disjoint pixel fragments are joined within the SGE frame buffer and displayed as a contiguous image [Perrine01]. 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 that they receive graphics data from multiple rendering nodes and route that data to high-resolution displays.

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

There are several parallel rendering systems that can benefit from SAGE or SGE. WireGL [Humphreys00] or parallel scene-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 [Humphreys02] or Aura [Germans01] are designed for distributing visualizations to and from cluster driven tiled-displays. However, these systems enable only one 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) [DMX] 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. 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.

An unique feature of SAGE compared with SGE, XDMX or Chromium is its high-speed graphics streaming capability over wide-area networks. 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. In addition, SAGE considers the 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 feature was also not considered in SGE and Chromium.

Table 5. Comparison between SAGE and other approaches

	SAGE	SGE	XDMX	Chromium	WireGL	TeraVision
multitasking (multiple windows)	Y	Y	Y	-	-	-
window reposition and resizing	Y	Y	Y	-	-	-
display-rendering decoupling	Y	Y	-	-	-	Y
high-performance WAN support	Y	-	-	-	-	Y
scalable parallel application support	Y	Y	-	Y	-	-
scalable image multicasting	Y	-	-	-	-	-

TeraVision [Singh04] developed by EVL is a scalable platform-independent solution that is capable of transmitting multiple synchronized high-resolution video streams between single workstations and/or clusters. TeraVision can also 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 a high-resolution tiled display but not suitable for supporting parallel applications or multiple instances of applications.

Table 5 compares SAGE with other systems. This table clearly shows that scalable image multicasting (Visualcasting) which will be addressed in this thesis is the most unique feature of SAGE. No other approach solves the problem.

4 Proposed Research

I propose Visualcasting – a scalable real-time image distribution service for ultra-high resolution display environments, which enables high-definition video and ultra-definition visualizations to be scalably multicasted in real time across distributed sites. Although there are several possible approaches to solve the same problem, this thesis will focus on SAGE Bridge, a high-speed bridging system. SAGE Bridge will be deployed on a cluster of PCs to replicate the pixel streams at core hubs in the network. These replication services can be dynamically scaled in proportion to the number of remote sites that are participating in a collaborative session. The replicated visualization streams are then properly re-routed and resized to fit on the tiled displays at the destinations. Figure 10 illustrates a Visualcasting session using SAGE Bridge.

4.1 Research and Development

This section presents the details of the proposed research including architecture, possible approaches, heterogeneity support and optimization issues. Several alternative approaches are compared with Visualcasting.

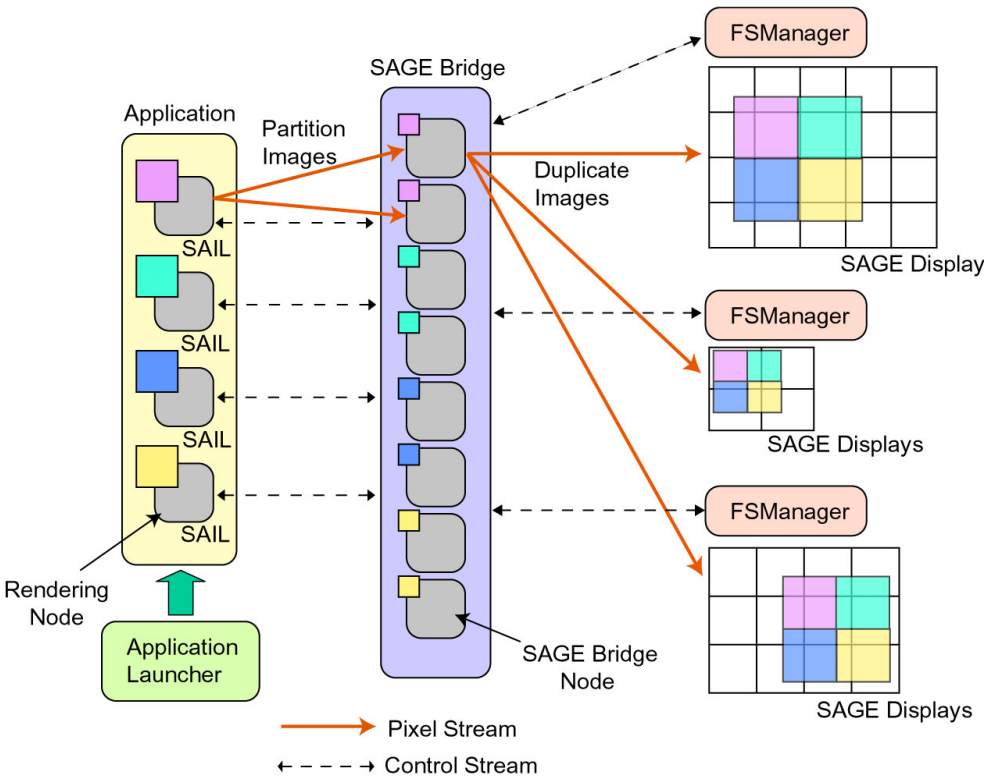


Figure 10. The SAGE Architecture for Visualcasting

4.1.1 The SAGE Architecture for Visualcasting

Visualcasting requires significant changes in the SAGE architecture as shown in Figure 10. Multiple FSManagers i.e. multiple SAGE sessions exist in this architecture. The SAGE Bridge is introduced between SAIL and SAGE Displays to intercept pixel streams from SAIL and to duplicate them for each SAGE session. These changes require a new procedure to execute applications as well. Figure 11 compares the old and new application launch procedures. FSManagers no longer executes applications in the new procedure. A new SAGE component, the Application Launcher, is introduced to launch applications instead of FSManagers. The new procedure consists of nine steps:

(1) A SAGE UI sends commands with application parameters and information about the SAGE Bridge and the first FSManager to the Application Launcher.

(2) The Application Launcher executes an application on appropriate rendering nodes using information from the SAGE UI.

(3) SAIL creates a control channel with the SAGE Bridge when the application is launched. The SAGE Bridge allocates SAGE Bridge nodes for the application and configures streams between SAIL and the SAGE Bridge.

(4) The SAGE Bridge connects to the first FSManager in order to configure the streams between the SAGE Bridge and the SAGE Displays.

(5) SAIL starts streaming pixels once all configurations are completed.

(6) Application images are displayed in the first SAGE session.

(7) In order to make the second SAGE session join the Visualcasting session, a SAGE UI sends a message that has information about the second FSManager to the first FSManager.

(8) The first FSManager directs the SAGE Bridge to connect to the second FSManager.

(9) The pixel streams between the SAGE Bridge and the second SAGE session are configured and started.

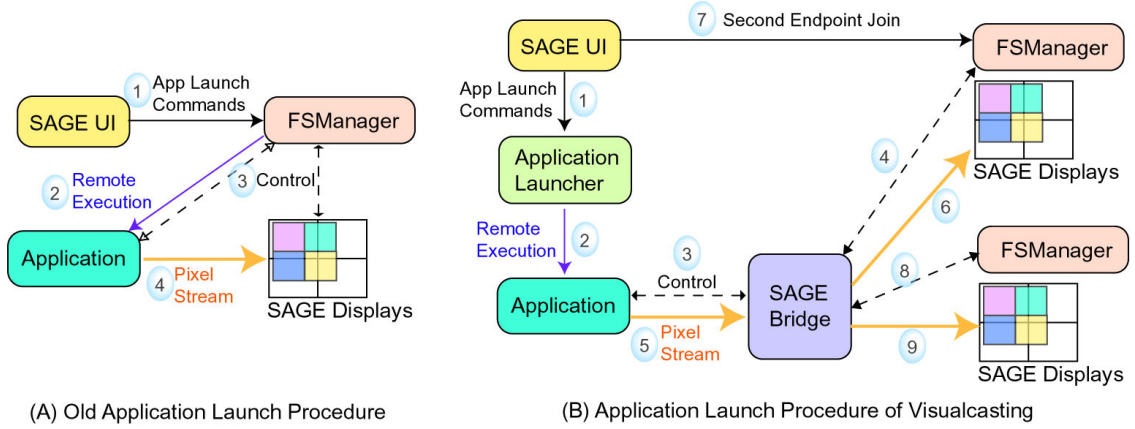


Figure 11. Old and New Application Launch Procedures

As the number of Visualcasting endpoints increases like this, initially assigned SAGE Bridge nodes become overloaded. The SAGE Bridge allocates additional nodes for the Visualcasting session. SAIL re-partitions images considering load balancing and streams to the SAGE Bridge nodes. This requires establishing new network connections and dynamically reconfiguring existing streams. Two research questions to be addressed are:

1. What are the conditions for adding or removing a SAGE Bridge node?
2. What SAGE Bridge architecture minimizes jitter on existing streams during this procedure?

If no additional nodes are available in the SAGE Bridge, it requests SAIL to reduce the rendering resolution or to compress the pixel blocks at the expense of the image quality.

4.1.2 Candidate Approaches to Visualcasting

Figure 12 shows four possible approaches to distribute pixels from the SAGE Bridge to tiled displays.

(1) The SAGE Bridge directly distributes pixel blocks to their final destinations as SAIL does in the latest SAGE prototype (see Figure 12-A).

(2) Placing a local bridge at each endpoint so that the SAGE Bridge duplicates pixel blocks and the local bridge distributes them over the tiles according to the application window layout (see Figure 12-B).

(3) Equally distributing pixel blocks over all the SAGE display nodes. Each node works as a local bridge node (see Figure 12-C). Some Visualcasting endpoints do not have enough nodes for a local bridge. It is economical if SAGE display nodes also work as a local bridge.

(4) Only the selected nodes receive pixel blocks from the SAGE Bridge and distribute them over the tiles (see Figure 12-D). The FSManager smartly chooses the local bridge nodes among the SAGE display nodes based on the application window layout and directs the SAGE Bridge to stream pixel

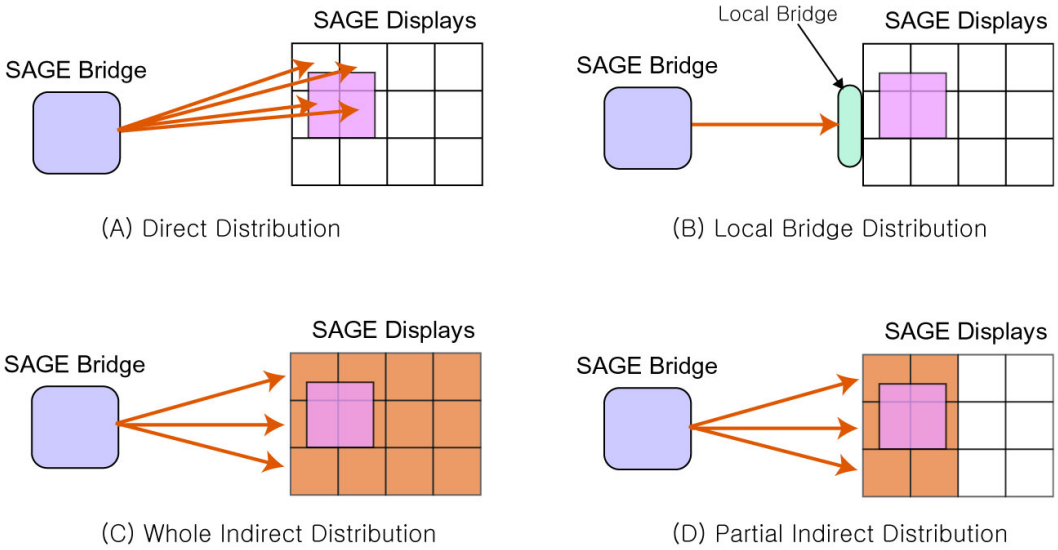


Figure 12. Visualcasting Approaches

blocks only to the selected nodes. This approach is useful for a large tiled display with multiple applications.

The second and third approaches have an advantage over the first one (Direct Distribution) in window operation latency because window reposition and resizing are performed locally. Direct Distribution, on the other hand, requires the reconfiguration of wide-area network streams for the operations. But Direct Distribution excels over other approaches in network streaming latency because indirect distribution schemes introduce additional delay in the network streams. In view of load balancing, the third approach excels beyond the others. But this approach may have huge re-routed traffic which affects the performance of other applications, so the third and fourth approaches are suitable for display clusters with high computing power or private network connections between the nodes.

The fourth approach is intended to be an optimal solution with a combination of the advantages of the other approaches: less re-routed traffic (the first approach), less hindering display processes (the second approach), load-balancing (the third approach) and avoiding reconfiguration of wide-area network streams (the second and third approach). So the algorithm to decide local bridge nodes for each application has to reflect the following strategies:

(1) Include the nodes now displaying the application on them (to reduce re-routed traffic).
(2) Exclude the nodes heavily used by other applications (for load balancing).
(3) Preferably include the nodes adjacent to the nodes selected by (1).
(4) Preferably avoid the change of the node set.

The purpose of (3) and (4) is to avoid reconfiguration of wide-area network streams. When application windows are moved or resized, (1) may incur frequent changes of the local bridge node set. (3) compensates this drawback by preventing node set changes due to small changes in the size or position of a window. There is a conflict between (1) and (4), so their priorities should be decided according to the current window position and size, and the amount of their changes relative to the total tiled display size.

4.1.3 Support for Heterogeneous Endpoints

As discussed in the introduction, each Visualcasting endpoint has heterogeneous network bandwidth, computing capacity and display resolution. Figure 13 shows the diversity of the tiled displays used by the international collaborators of EVL. An ultra-high-resolution visualization affordable by the 55-tile display at UIC in Figure 13 may not fit into the four-tile display at TAMU. A simple solution for this problem is to have the application adapt its resolution to the smallest tiled display. The latest prototype of SAGE assumes the application statically configures its rendering resolution. An appropriate application interface to dynamically change visualization resolution needs to be implemented for this solution. A more intelligent solution is to have the SAGE Bridge adapt the resolution of image streaming to each endpoint. This

Figure 13. Various Tiled Displays at International Collaboration Sites

requires the SAGE Bridge to resample received pixel blocks and to stream the new blocks with reduced resolution to the endpoints with less display resolution. If an endpoint has insufficient network bandwidth in spite of high display resolution, the compression of pixel blocks should be an appropriate solution. The compression can be performed by SAIL or the SAGE Bridge.

In an ideal case, the SAGE Bridge distributes pixel data at the same rate as it receives data. But the endpoints consume the data at different rates due to the heterogeneity discussed above. If the bridge distributes the data as it is, the global data transfer rate will drop down to the rate of the slowest endpoint. To avoid this the bridge needs to adaptively reduce the data size for slow endpoints by resampling or dropping image frames while streaming whole image frames to fast endpoints. In summary, the SAGE Bridge has to adaptively control the data transfer rate to each endpoint considering the display resolution, network bandwidth and data consuming speed of the endpoint by resampling, compressing or dropping image frames.

4.1.4 Pixel Block Size and Network Protocol Interfaces

As discussed in Section 3.1.3, pixel block based streaming will be used for Visualcasting. Since SAIL delivers pixel data to a network protocol block by block, decreasing pixel block size increases network API function calls which may incur network streaming overhead. It increases OpenGL API function calls in SAGE Displays as well because SAGE Displays send pixel data to the graphics card block by block. This may incur another overhead in the display process. However, increasing pixel block size incurs network streaming overhead in the case of the first and second approaches in Section 4.1.2. Since a pixel block is

regarded as an indivisible unit in Visualcasting, the whole pixel block is streamed even though it is partly shown on the display (pixel blocks on the edges of the tiles). Using larger pixel blocks may have more pixel data streamed that is not displayed. Using small pixel blocks also increases image splitting flexibility which is preferable for the third and fourth approaches in Section 4.1.2 and non-rectangular image streaming.

The trade-offs between large and small pixel blocks will be investigated more deeply. Some modifications to the assumptions and the approaches used in Visualcasting may help resolve some of these issues. For example, aggregating small pixel blocks to make a large block before streaming over networks or downloading into the graphics card may increase the performance. Allowing a few exceptions to the assumption of the indivisible pixel block also can reduce the network streaming overhead.

How to fetch pixel data from SAGE applications and pass it to network protocols is another problem. SAIL currently uses double buffering – the applications fill one buffer while SAIL transports the other buffer over the network. The two buffers are swapped once both sides are ready to go to the next frame. SAIL generates pixel blocks from the fetched image frame and streams them using the blocking network send function calls. The blocking send, however, may slow down wide area reliable network streaming because it blocks the sender until it confirms that the receiver gets all the data. A non-blocking send using LambdaStream (a new network protocol for high-performance wide area network streaming developed by EVL) [Xiong05, Vishwanath06] can improve performance. It requires an interface between SAIL and the network protocol to check if the pixel block buffers are ready to be overwritten before they are filled with new pixel data. Another interface is needed to implement adaptive pixel data streaming for which SAIL requests a desirable network bandwidth for an application from LambdaStream, and the available bandwidth for the request is returned. SAIL adapts the data streaming rate to the available bandwidth by reducing the frame rate or image resolution. These interfaces are very important to improve Visualcasting performance and afford a large number of pixel streams.

4.1.5 Pixel Stream Compression

During SAGE performance evaluation, SAGE applications typically used around 600Mbps network bandwidth per rendering node to stream their output to a remote tiled display (e.g. 1600 x 1200 x 24bit x 15fps = 691Mbps). The 1Gbit network interface on each node was easily saturated if multiple applications run on one rendering cluster. One parallel application utilized over 9Gbps from 10Gbps available WAN bandwidth when it streams extremely high resolution images such as 6400 x 4800 aerial photography. This means 10Gbps WAN bandwidth is not enough for multiple users to share by running their own applications in the SAGE frame work. Pixel stream compression is a very reasonable solution for this problem. Even if the network interfaces are upgraded to 10Gbit Ethernet and the WAN evolves to a Terabit network, the compression will allow more people to run more diverse applications in the SAGE frame work.

However, it is almost impossible to generate pixel blocks from a compressed image. It cannot be split and distributed over a tiled display. Hence, compressing an entire image is inappropriate for SAGE streaming. Instead, the image data of each pixel block has to be compressed separately. This makes sense for Visualcasting because the SAGE Bridge can send the compressed pixel blocks to the proper destinations by just checking their header without decompressing them. Furthermore, the pixel block compression reduces the load of the SAGE Bridge, though it increases the load on SAIL and SAGE Displays. It can be a good solution when the SAGE Bridge is overloaded due to the increase of the number of Visualcasting endpoints. A few real-time image compression techniques will be used including Run Length Encoding (RLE), the color format transformation between RGB and YUV, DXT texture compression and Wavelet transforms.

4.1.6 Comparison with Other Approaches

Visualcasting can be implemented using conventional multicast techniques, such as traditional IP multicast, photonic multicast or reliable layered multicast. These approaches, however, can be applied only on multicast-enabled networks with special configurations. In contrast, the SAGE Bridge approach is applicable on any network with a normal configuration. This is one straightforward advantage of the SAGE Bridge approach over multicast approaches.

The SAGE Bridge approach is more flexible than multicast approaches in that it can perform various operations on incoming pixel data including partitioning, resampling, compression and frame dropping as needed by each endpoint. On the other hand, such intermediate pixel data processing and stream control that considers the heterogeneity of endpoints is not feasible using conventional multicasting. All necessary pixel data processing should be done at the sending side. Layered multicast [McCanne96] can address heterogeneity of endpoints, but the source image has to be transformed appropriately for each layer. This may place an excessive computational load on the rendering nodes.

In addition, the source image has to be partitioned considering the display configurations and window layouts of all collaboration endpoints. Since each partitioned image should be streamed to a distinct multicast address, the number of available multicast addresses restricts the precision of image partitioning. An improper image partition may incur network streaming overhead. Window reposition or resize changes the membership of the multicast groups. In some cases, this change takes so much time that SAGE cannot perform window operations interactively.

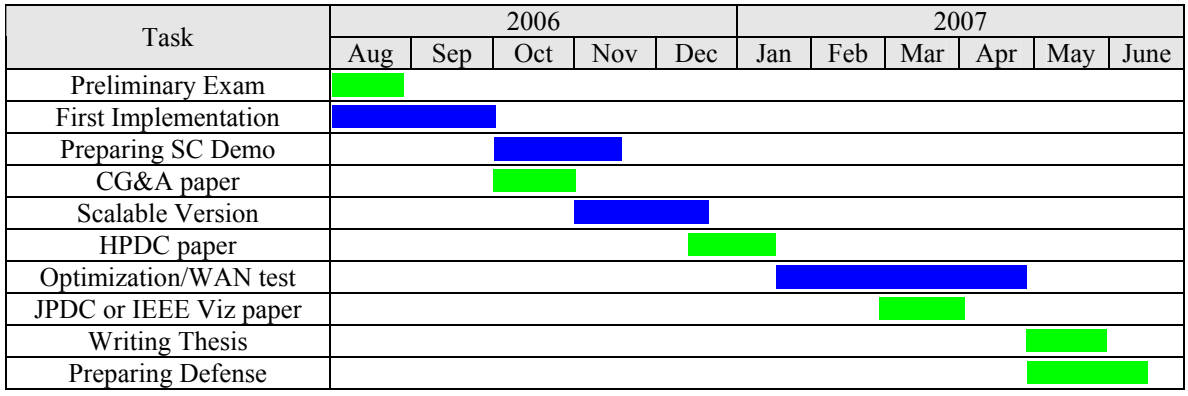
I will survey those multicasting techniques more deeply and compare them with the SAGE Bridge approach in their scalability, performance and functionality.

4.2 Metric for Success

The proposed approach will support distant collaboration between heterogeneous endpoints by distributing visualization at extremely high resolutions. The bandwidth and latency of this approach will be measured as the number of collaborating sites increases. The limits of this approach will be identified as well. Scalability, achievable bandwidth and latency as Visualcasting supports heterogeneous endpoints will decide the success of this approach.

4.3 Timeline

Deadline	Task Completed
June 29th, 2007	Submit all the paperwork needed to officially complete my thesis with UIC
June 15th, 2007	Thesis defense
May 25th, 2007	Hand in draft of thesis
April 27th, 2007	Real-world test over WAN, Schedule defense date with committee members
March 31st, 2007	Write a paper for JPDC or IEEE Visualization
March 15th, 2007	Optimization of Visualcasting (compression, network API)
Jan 15th, 2007	Write a paper for HPDC
Dec 31st, 2006	Evaluate performance and scalability and compare candidate approaches
Dec 15th, 2006	Implement a scalable version including all candidate approaches
Nov 11th, 2006	SC demo and paper presentation
Oct 31st, 2006	Write a paper for CG&A, Survey of multicast approaches
Oct 15th, 2006	Evaluate and analyze the performance of the first implementation
Sep 30th, 2006	Implement basic image distribution and heterogeneity support
Aug 24th, 2006	Preliminary Exam



4.4 Deployment Plan

The Visualcasting approaches will be implemented in future versions of SAGE. These versions will be released at the SAGE web site: http://www.evl.uic.edu/cavern/sage.

4.5 Experiment Plan and Equipment Needed

An initial test bed of the SAGE Bridge will consist of four high-performance PCs each equipped with two 10Gbit network interfaces that are connected to a LambdaVision cluster through local 10Gbit networks. The LambdaVision cluster will be configured as one rendering cluster and several display clusters. For example, 28 cluster nodes each equipped with a gigabit interface can be configured as an 8-node rendering cluster, and a 2-node (4-tile), 6-node (12-tile) and 12-node (24-tile) display cluster. A 10Gbit switch will connect the rendering/display cluster and SAGE Bridge nodes (see Figure 14). This test bed will be used to evaluate scalability and performance of Visualcasting supporting heterogeneous endpoints on local area networks.

Once the basic capabilities of SAGE Bridge is fully tested on the local test bed, a SAGE Bridge cluster will be placed at StarLight in downtown Chicago for real-world tests including remote collaboration sites. The

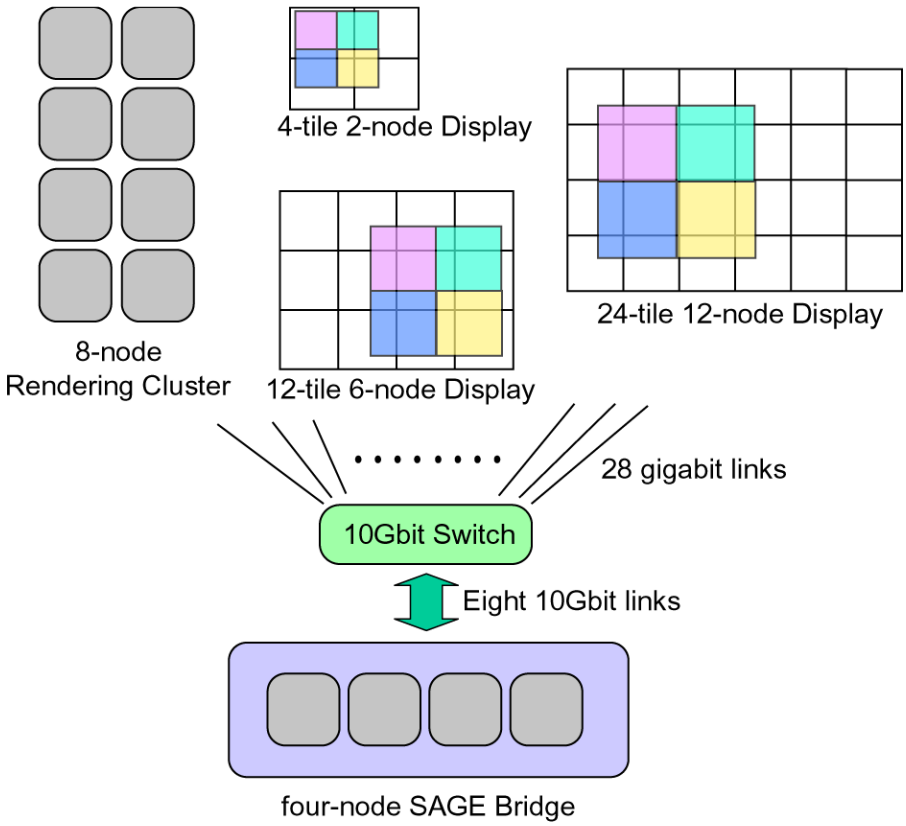


Figure 14. SAGE Bridge Test bed on a local network

cluster can be the same as in the initial test bed, or considering the fact that outgoing traffic from the SAGE Bridge is usually several times bigger than its incoming traffic the cluster may consist of four high-performance PCs each equipped with a 10Gbit (output) and two gigabit (input) network interfaces. The gigabit interfaces are connected to EVL/UIC through a 10Gbit network and the 10Gbit interfaces are connected to remote collaboration sites including University of Michigan (over Internet2), Calit2/UCSD (over CAVEWave/National Lambda Rail), SARA in Amsterdam (over SURFNet) and KISTI in Korea (over KREONet2). This is a good test case for heterogeneity support because these collaboration sites have tiled displays with various display resolutions, computing power, network latency and bandwidth.

5 References

[Childers00] Childers, L, Disz, T., Olson, R., Papka, M. E., Stevens, R., and Udeshi, T., "Access Grid: Immersive Group-to-group Collaborative Visualization," Proceedings of Fourth International Immersive Projection Technology Workshop, 2000.

[Deering91] Deering, S. E., "Multicast Routing in a Datagram Internetwork," PhD thesis, Stanford University, December 1991.

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

[Germans01] Germans, D., Spoelder, H.J.W., Renambot, L., and Bal, H. E., "VIRPI: a High-level Toolkit for Interactive Scientific Visualization in Virtual Reality," Proceedings of Immersive Projection Technology/Eurographics Virtual Environments Workshop, 2001.

[He03] He, E., et al, "Quanta: a Toolkit for High Performance Data Delivery over Photonic Networks," Journal of Future Generation Computer Systems, Volume 19, Issue 6, August 2003, pp. 919-933.

[Humphreys00] Humphreys, G., Buck, I., Eldridge, M., and Hanrahan, P., "Distributed Rendering for Scalable Displays," Proceedings of ACM/IEEE Conference on Supercomputing, 2000.

[Humphreys02] Humphreys, G., et al, "Chromium: a Stream-processing Framework for Interactive Rendering on Clusters," Proceedings of SIGGRAPH, 2002.

[Jeong06] Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A., and Leigh, J., "High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment," accepted by ACM/IEEE Supercomputing 2006.

[Klosowski02] Klosowski, J. T., Kirchner, P., Valuyeva, J., Abram, G., Morris, C., Wolfe, R., and Jackman, T., "Deep View: High-resolution Reality," IEEE Computer Graphics and Applications, Volume 22, Issue 3, May/June 2002, pp. 12–15.

[Krumbholz05] Krumbholz, C., Leigh, J., Johnson, A., Renambot, L., and Kooima, R., "Lambda table: High Resolution Tiled Display Table for Interacting with Large Visualizations," Proceedings of Fifth Workshop on Advanced Collaborative Environments, 2005.

[Leigh03] Leigh, J., Renambot, L., DeFanti, T. A., et al, "An Experimental OptIPuter Architecture for Data-Intensive Collaborative Visualization", Third Workshop on Advanced Collaborative Environments, Seattle, WA, June 2003.

[Leigh06] Leigh, J., Renambot, L., Johnson, A., Jeong, B., et al, "The Global Lambda Visualization Facility: An International Ultra-High-Definition Wide-Area Visualization Collaboratory," Journal of Future Generation Computer Systems, Volume 22, Issue 8, October 2006, pp. 964-971.

[McCanne96] McCanne, S., Jacobson, V., and Vetterli, M, "Receiver-driven Layered Multicast," ACM SIGCOMM, 1996.

[Perrine01] Perrine, K. A., Jones, D. R., and Wiley, W. R., "Parallel Graphics and Interactivity with the Scaleable Graphics Engine," Proceedings of ACM/IEEE Conference on Supercomputing, 2001.

[Renambot04] Renambot, L., Rao, A., Singh, R., Jeong, B., et al, "SAGE: the Scalable Adaptive Graphics Environment," Proceedings of WACE 2004, Nice, France, 09/23/2004 - 09/24/2004.

[Singh04] Singh, R., Jeong, B., Renambot, L., Johnson, A., and Leigh, J., "TeraVision: a Distributed, Scalable, High resolution Graphics Streaming System," Proceedings of IEEE Cluster, 2004.

[Smarr03] Smarr, L., Chien, A. A., DeFanti, T., Leigh, J., and Papadopoulos, P. M., "The OptIPuter" Communications of the ACM, Volume 46, Issue 11, November 2003, pp. 58-67.

[Stix01] Stix, G., "The Triumph of the Light," Scientific American, January 2001.

[Vishwanath06] Vishwanath, V., Leigh, J., He, E., Brown, M. D., Long, L., Renambot, L., Verlo, A., Wang, X., DeFanti, T. A., "Wide-Area Experiments with LambdaStream over Dedicated High-bandwidth Networks," IEEE INFOCOM, April 2006.

[Xiong05] Xiong, C., Leigh, J., He, E., Vishwanath, V., Murata, T., Renambot, L., and DeFanti, T., "LambdaStream – a Data Transport Protocol for Streaming Network-intensive Applications over Photonic Networks," Proceedings of The Third International Workshop on Protocols for Fast Long-Distance Networks, Lyon, France, Feb. 2005.