Remote Visualization of Large Scale Data for Ultra-High Resolution Display Environments

Sungwon Nam[1], Byungil Jeong[2], Luc Renambot[1], Andrew Johnson[1], Kelly Gaither[2], Jason Leigh[1]

[1] Electronic Visualization Laboratory, University of Illinois at Chicago
842 W. Taylor St., Chicago, IL 60607
snam5, renambot, spiff @ uic.edu
aej @ evl.uic.edu

[2] Texas Advanced Computing Center
10100 Burnet Rd. Bldg 196, Austin, TX 78758
bijeong, kelly @ tacc.utexas.edu

ABSTRACT

ParaView is one of the most widely used scientific tools that support parallel visualization of large scale data. The Scalable Adaptive Graphics Environment (SAGE) is a graphics middleware that enables real-time streaming of ultra-high resolution visual content from distributed visualization resources to scalable tiled displays connected by ultra-high-speed networks. Integrating these two technologies enables visualization of large-scale data at an extremely high resolution to be displayed on distantly located scalable tiled displays. The benefits, limitations, and future directions for this approach will be discussed.

Categories and Subject Descriptor

I.3.2 [**Computer Graphics**]: Graphics Systems – *Distributed/network graphics*

General Terms

Experimentation, Performance

Keywords

Large-scale data, remote visualization, ultra-high resolution visualization, ParaView, SAGE

1. INTRODUCTION

The volume of scientific data collected from sensors and simulations continues to grow exponentially. Individual data objects are now on the order of terabytes and will soon reach petabytes. This magnitude of data easily exceeds the capacity of a personal computer or even a modest compute cluster. Visualizing this large-scale data therefore is best achieved through the use of High Performance Computing resources to render the computationally intensive visualization, while allowing users to employ thin clients connected over emerging high-speed networks to view the results. A parallel visualization application that is well suited to this model is ParaView [5, 8]. It enables users to visualize and interact with the large-scale data on a remote visualization cluster via an interactive client running on the users' laptop or desktop computer. However, the resolution and interactivity of visualizations is limited by the clients' screen resolution and available network bandwidth.

On the other hand, the use of scalable tiled display walls to view visualizations of large-scale data at near native resolution is becoming increasingly popular due to its growing affordability. Furthermore the expansive size and exquisite resolution of the display has been conclusively demonstrated to positively impact the scientific discovery process by allowing researchers to juxtapose multiple high-resolution visualizations [6, 9, 18, 20]. We have developed an "operating system" for such ultra-high resolution display systems known as Scalable Adaptive Graphics Environment (SAGE) [13]. SAGE facilitates launching visualization applications on separately distributed remote clusters as well as streaming the resulting visualization directly to the users' tiled displays. The uniqueness of SAGE lies in its ability to stream the visualizations to any portion of a tiled display as individually managed windows. Such windows not only allow a user to manipulate and view the visualization, but also provide users a way to manage multiple streams to their liking. SAGE allows users to scale the traditional notion of the thin-client to extremely high resolution given sufficient network bandwidth.

ParaView itself can support visualization on a tiled display with two limitations: 1) the tiled display has to be directly connected to the rendering nodes – there is no remote visualization for a tiled display; 2) It only allows one dataset to be visualized at a time, occupying the entire display. Our approach integrates ParaView with the SAGE framework, to enable ParaView to overcome these limitations. Furthermore SAGE, through *Visualcasting*, provides the ability to replicate visualization streams to multiple destinations thereby enabling distance collaboration. The contributions of this paper are threefold:

- It provides users with a solution to visualize large-scale data that best leverages the use of ultra-high resolution scalable tiled displays and remote High Performance Computing resources.
- It demonstrates the viability of a scalable thin-client-based approach to remote visualization.
- It provides users of VTK-based [3, 5, 7, 17] applications with a means to perform remote visualization on scalable tiled displays.

2. RELATED WORK

Perrine et al. and Klosowski et al. presented the merits of high-resolution display for various visualization applications using IBM's Scalable Graphics Engine (SGE) [14, 16]. 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. SGE supports up to sixteen 1GigE inputs and can drive up to eight displays with double-buffering to support display systems of up to 16 megapixels. SAGE and SGE are similar in that they both 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 which is much more flexible and scalable than the latter. Since SAGE does not require any special hardware, and network technologies such as 10GigE and novel transport protocols are easily applied to SAGE. SGE, on the other hand, is bound to 1GigE inputs and the SGE-specific network protocol. There is no theoretical limitation to scaling the performance of SAGE by adding more rendering and display nodes. Conversely, 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 [11] 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 [12] or Aura [10] are designed for distributing visualizations to and from cluster driven tiled displays. However, since these systems enable only one application at a time with a static layout on a tiled display, they require a graphics streaming architecture such as SAGE or SGE to move, resize and overlap multiple application windows.

XDMX (Distributed Multi-head X11) [1] 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 can also support Chromium to display multiple applications on a tiled display. However, XDMX does not support parallel applications. This limits its scalability with respect to large datasets.

No other systems discussed so far were designed to stream graphics data over a high-speed wide-area network. In contrast, SAGE has both a TCP and a UDP-based high-speed pixel streaming architecture for wide-area networks that have multi-ten gigabits of network bandwidth. The architecture is open so that it may use new streaming protocols designed for high-bandwidth and high round-trip time networks that are not considered in the streaming architectures of SGE and Chromium. In addition, SAGE takes the mullions (borders) of each LCD panel of tiled displays into consideration when displaying application windows. Hence, the mullions appear to be placed on top of a large continuous image.

TeraVision [19] developed by EVL is a hardware-based scalable platform-independent solution that is capable of transmitting multiple synchronized high-resolution video streams between single workstations and/or clusters. While TeraVision can also stream graphics data over wide-area networks, 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 for supporting parallel applications or multiple instances of applications.

3. INTRODUCTION TO PARAVIEW AND SAGE

In this section, we will briefly introduce ParaView's modes of operation in multi-processor environments, and the underlying model of SAGE and its capabilities.

3.1 ParaView

ParaView is a scientific visualization tool designed to analyze large datasets by taking advantage of distributed memory computing environments [4]. While a user can perform the computations and rendering in a single machine by running a single instance of ParaView, multiple instances running in parallel is preferred for visualizing large datasets. ParaView supports various modes of operations on a distributed computing environment (Figure 1) [2].

- Client-Server Mode: a server (a single PC) which is located remotely, performs the computation and rendering, and the resulting pixels are streamed to a desktop client. This mode enables remote visualization but does not support parallel computation and rendering.
- Distributed Server Mode: a cluster of computers performs the computation and rendering in parallel. In this mode, the master node of the cluster composites the final image and sends it to the client. Hence, the resolution of the final image is limited to client's desktop resolution.
- Tiled-Display Mode: a cluster of computers performs the computation and rendering in parallel. Each cluster node renders and composites its image fragment (view frustum) and displays it on physically connected display. This mode does not support remote visualization.

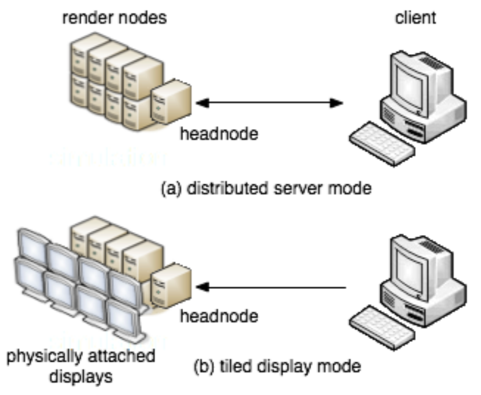

Figure 1. ParaView's two modes for supporting scalable visualization: (a) In the Distributed Server mode, the head node sends the final image to the client. (b) In the Tiled Display mode, the final image is displayed on monitors attached to rendering cluster.

While ParaView supports remote parallel rendering via its Distributed Server mode, and ultra-high resolution rendering via its Tiled Display mode, it does not support both at the same time- SAGE bridges this gap. The next section describes how this is accomplished.

3.2 SAGE

Unlike other tiled display approaches, such as Chromium, SAGE delegates the rendering of graphics to remotely located compute clusters, and relies on the use of high speed networks to stream the pixels of the visualization to the displays. This "thin-client" model has the advantage that large cluster farms or supercomputers can be brought to bear to render datasets that may be too large to fit on an individual graphics card. Furthermore, SAGE's Visualcasting service replicates and distributes the high-resolution visualization streams at interactive frame rates to multiple distantly located tiled displays thereby enabling collaborative visualization. Users can juxtapose and manipulate multiple high-resolution visualization windows on their tiled display.

4. PARAVIEW AND SAGE INTEGRATION

This section describes the integration of ParaView with SAGE using SAIL (the SAGE Application Interface Library).

4.1 Integrating Visualization Applications with SAGE

There are two ways to integrate a visualization application into the SAGE framework. First, visualization applications can be modified to use a thin API layer called SAIL (the SAGE Application Interface Library) that will capture the application's frame buffer and then stream it to the remote tiled display. SAIL is also capable of supporting parallel rendering applications where multiple nodes may be generating a sub-portion of the overall full image. SAGE takes each of the individual sub-images and stitches them together in real-time for presentation on the tiled display.

A second approach leverages VNC server to stream the entire computer desktop screen to the tiled display. In this model SAGE launches a VNC client that is enhanced so that the received pixels are placed in a frame buffer that are then routed to the tiled display. This approach enables any computer or laptop to "push" its screen onto the tiled display without modifying any application code. While using VNC to stream pixels to SAGE can give users easy access to tiled display, the resolution is limited to the users' desktop resolution.

4.2 SAIL in ParaView

To support parallel visualization at an ultra-high resolution on SAGE-driven tiled display, one must use SAIL. The integration of SAGE with ParaView involves applying SAIL to ParaView in its Tiled Display Mode. SAIL captures the graphics buffers on each node of the rendering cluster, and streams them to a SAGE-driven tiled display (see Figure 2).

ParaView is built on top of the Visualization Toolkit (VTK) [17], which includes a class called *vtkXOpenGLRenderWindow*. This class is responsible for creating and managing OpenGL windows on an X display. It is on this X display that the VTK renderers draw. And its member function *Frame()* is responsible for executing the graphics swapbuffer call. SAIL API calls are inserted in this member function to retrieve pixel data from the framebuffer and stream them to a tiled display. ParaView performs sort-last rendering, i.e. each renderer loads partial data into its main memory and renders it. The rendered images are composited into image fragments at each render node according to the viewport of each renderer. Each instance of SAIL reads the image at each rendering node and streams it to a tiled display driven by SAGE.

The main advantage of this approach is that the display and network streaming of the visualization are totally transparent from the application. Thus, ParaView can be optimally configured for rendering without having to be concerned about optimizing for network streaming, or the layout of the tiled display. SAGE manages the network streaming and the image scaling on the tiled display. Furthermore, multiple ParaView applications can be launched on other remote compute clusters, and their results can be simultaneously streamed for viewing and comparison on a tiled display as individual windows. Multiple ParaView sessions on SAGE-driven tiled display in action is shown in Figure 3.

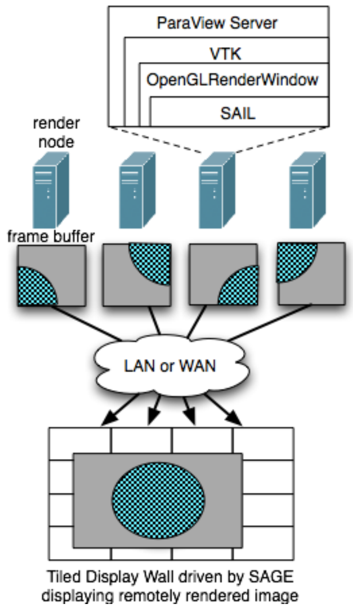

Figure 2. Diagram showing four render nodes streaming to a SAGE-driven tiled display using SAIL embedded in VTK's OpenGLRenderWindow.

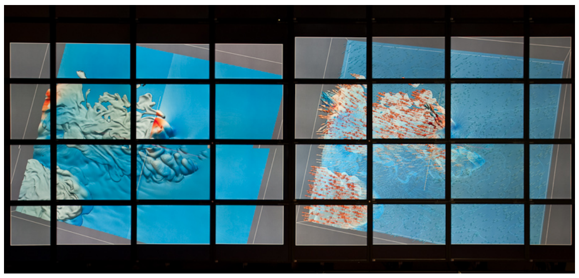

Figure 3. 100 million pixel tiled display at EVL displaying two ParaView sessions. On the left, two local rendering nodes stream to the display. On the right, four remote rendering nodes stream from TACC.

5. IMPACT OF STREAMING DELAY

Users can take advantage of remote ultra-high resolutions visualization by using ParaView integrated into SAGE at a cost of additional delay introduced by various SAGE components in the streaming pipeline. This delay could impede the user's ability to interact with the remote visualization. In this section, we will describe a model for predicting the delay, which is a combination of the delay at the sender (rendering node) and the receiver (display node).

5.1 Delay at the Render Node

Once rendering is completed, a rasterized image resides in the framebuffer of the render node, A SAIL object in the render node delivers pixels to tiled display. It first copies pixels from the framebuffer to main memory (incurring capture delay), waits for other render nodes to finish rendering and copies the pixels to main memory (synch delay). The image is then split into multiple pixel blocks (split delay), and sends groups of pixel blocks to their destined display nodes.

To simplify the model, we assume following:

- The rasterized image is divided evenly. The number of pixels generated by each render node is uniform. Thus the image size $S_{frustum}$ in each node is the final image size S_{image} divided by total number of render nodes N_{ren_node}. And the capture delay at a render node is proportional to $S_{frustum}$.

$$S_{frustum} = S_{image} / N_{ren\_node} \tag{1}$$

- Rendering delay at a node is uniform (assuming a well balanced parallel renderer). This simplifies the synchronization delay D_{sync} to be a function of the number of render nodes, i.e.:

$$D_{sync} = C_s N_{ren\_node} \tag{2}$$

Thus, the streaming delay at a render node D_{render_node} is the sum of these three components (capture delay $D_{capture}$, sync delay D_{sync}, and split delay D_{split}), and can be represented as follows.

$$D_{capture} + D_{sync} + D_{split} \tag{4}$$

where

$$D_{capture} = C_c S_{frustum} \tag{5}$$

and

$$D_{split} = C_b S_{frustum} \tag{6}$$

By substituting (2), (5), and (6) with (4), the streaming delay at a render node is therefore:

$$D_{render\_node} = \left(\frac{C_c + C_b}{N_{ren\_node}} \right) \cdot S_{image} + C_s N_{ren\_node} \tag{7}$$

Given the number of rendering node, the delay can be shown as a linear function of image size. Also, (7) can be seen as a function of the number of render node with given image size. We found that the effect of image size dominates other factors in streaming delay. However, the number of rendering nodes also affects parallel computation and rendering performance in ParaView. In many cases, the delay from ParaView's rendering dominates over SAGE's streaming delay as we will show in the Section 6. The number of rendering nodes should therefore be chosen to minimize the delay incurred by ParaView. In the Section 6, we will also show how the delay varies with respect to the final image size.

5.2 Delay at the Display Node

The pixel receiving and displaying procedures start when the first group of pixel blocks arrive from the network. Each display node copies pixel blocks to its texture memory, waits for the other display nodes to finish copying the pixels, and displays the image on the screen. The following are our assumptions upon which our model is based:

- An image is evenly distributed across display nodes. Thus the number of pixel block groups that each display node receives is uniform. The number of pixel block groups at each display node can be calculated by dividing the total number of pixel block groups for the final image (S_{image} / S_{grp}) by the number of display nodes $N_{display\_node}$.

$$N_{grp} = S_{image} / (S_{grp} N_{display\_node}) \tag{8}$$

- When pixel blocks arrive at a display node, they are queued until the display process retrieves and copies them into its texture memory. We assume this queueing delay D_Q is constant.

The delay at a display node $D_{display\_node}$ is sum of the queueing delay D_Q, delay to copy to texture memory D_{cpy}, and the synchronization overhead D_{sync}. D_{cpy} is of course propotional to the size of the image fragment that a display node receives and displays ($N_{grp} S_{grp}$); and D_{sync} is a function of the number of display node.

$$D_{display\_node} = D_Q + D_{cpy} + D_{sync} \tag{9}$$

where

$$D_{cpy} = C_t N_{grp} S_{grp} \tag{10}$$

and

$$D_{sync} = C_s N_{display\_node} \tag{11}$$

By substituting (8) with (10), and (11) with (9), we obtain the following.

$$D_{display\_node} = D_Q + \frac{C_t S_{image}}{N_{display\_node}} + C_s N_{display\_node} \tag{12}$$

The delay at a display node can be viewed as a function of the final image size, or a function of the number of display nodes. Similar to the delay at a render node, the final image size is a dominant factor contributing to the delay. Therefore, one can lower the delay by maintaining sufficient number of display nodes. However, as (12) suggests, the delay also increase as the number of display nodes increases. Therefore a carefully balance is needed for optimal performance. We will discuss this more in Sections 6.

6. EXPERIMENTAL EVALUATION

In this section, we will present our experimental results showing delay with respect to the size of the dataset and final image. We used Texas Advanced Computing Center (TACC)'s Spur visualization nodes for computation, and rendering. Each node has four quad core 2.4GHz AMD Opteron processors, 128GB main memory, and 4 Nvidia Quadro 5600 Graphics Hardware in PCI-Express slots. For the remote display we used Electronic Visualization Laboratory (EVL)'s 100-megapixel LambdaVision driven by a 30-node cluster. The cluster nodes are equipped with 64-bit dual 2.4GHz AMD Opteron processors, 4GB main memory, and Nvidia Quadro 3000 Graphics Hardware in an AGP slot. We connected the two sites using EVL's 10 Gigbit/s National Lambda Rail connection.

6.1 Frame Rate and Total Delay

A 7.7 million cell thunderstorm dataset was contour-filtered with ParaView to generate isosurfaces with varying numbers of polygonal mesh cells. The number of isosurface cells used for the experiment were 1, 2.5, 4.1, and 5.5 million cells. Four rendering nodes at TACC rendered the isosurfaces in parallel and 16 display nodes at EVL received and displayed the pixels. The frame rate and total delay while interacting with the ParaView client was measured. Figure 4 and 5 show the overhead imposed by SAGE. As data size increases, rendering delay dominates over SAGE streaming delay.

6.2 SAGE Delay

SAGE delay is dependent on image size (number of pixels to stream) and number of render/display nodes as described in (7) and (12). In this experiment, we fixed data size and varied other factors that affect SAGE delay. Figure 7 shows SAGE delay with respect to image size with 4 render nodes at TACC streaming to 16 display nodes at EVL. The curves with dotted lines represent expected delay according to the model described in section 5. The actual delay observed was much less than the model because we assumed capture (5) and split delay (6) at the render side and copy delay (10) are linearly proportional to the image size in the model. The capture delay is the dominant factor among other components and is highly dependent on hardware performance. At the display side, the copy delay is minimal compared to queueing delay D_Q. The queueing delay can be lowered by reducing the number of pixels that has to be delivered to each node. This is shown in Figure 6. In our experiment, a single render node evenly distributes pixels (its image fragment) to four display nodes.

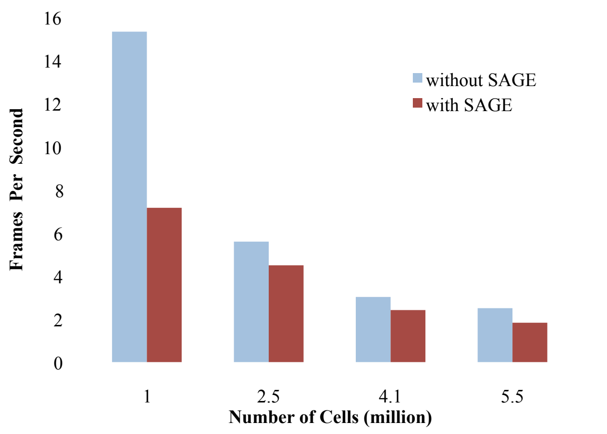

Figure 4. Graph showing the effect of frame rate vs number of cells in a polygonal mesh data. The size of the image streamed is 3200x2400. As data size increases, the delay incurred by SAGE becomes negligible.

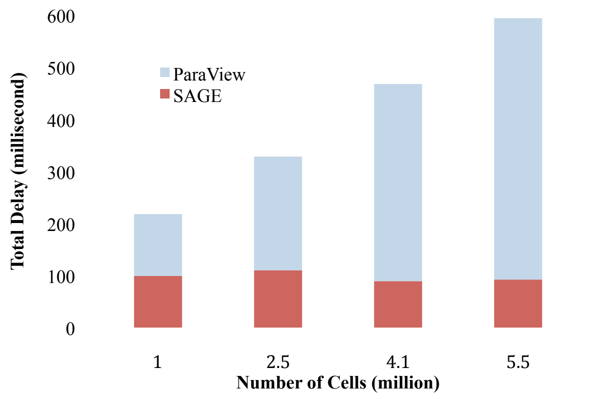

Figure 5. Graph showing ParaView and SAGE delay. The size of the image streamed is 3200x2400. SAGE delay remains constant because the image size is constant whereas ParaView delay increases as data size increases.

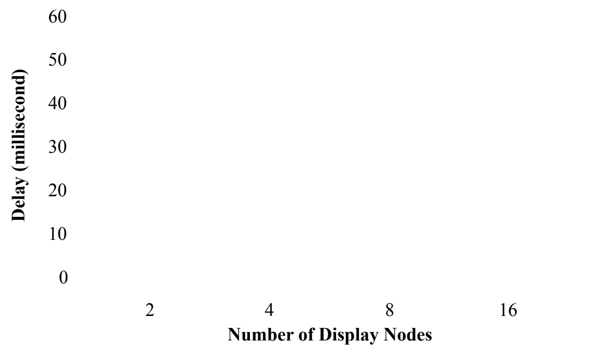

Figure 6. Graph showing the delay at the display side vs number of display nodes. Four rendering nodes stream an image of size 3200x2400. The delay is reduced as we add more display nodes.

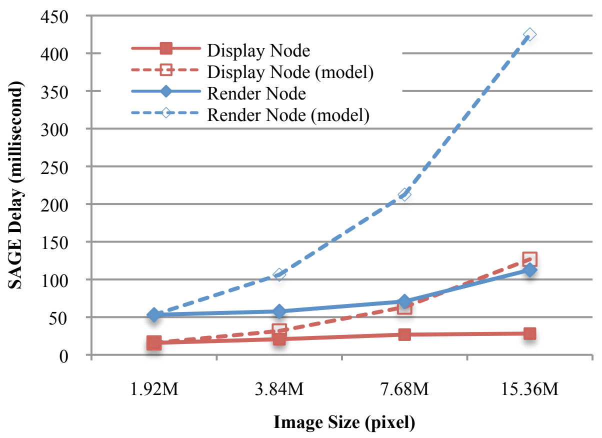

Figure 6. Delay incurred by SAGE is shown with model. This graph shows increasing deal from SAGE as the image size increases. When the image size is doubled, SAGE delay is increased by only small fraction.

7. CONCLUSIONS AND FUTURE WORK

The integration of ParaView and SAGE enables ParaView and VTK-based applications to stream ultra-high resolution visualizations from remote rendering servers to clients ranging from laptops to scalable tiled displays. This paper has shown that the overhead imposed by such a capability is negligible compared to the overall time typically needed for rendering large-scale data.

In the current ParaView/SAGE integration, when a user enlarges a window on the tiled display, the image is not rendered at higher resolution; rather the pixels are enlarged to fill the screen space. In the future we will enhance SAGE to send window dimensions to ParaView so that it can dynamically render images of greater resolution. Ideally to ensure continued scalability, ParaView should attempt to utilize more rendering nodes in order to keep frame rates up.

In the future, we intend to also integrate VisIt [3] and VisTrails [7] with SAGE. We expect this integration to be relatively straightforward as both of these visualization tools use VTK as their base visualization framework.

8. REFERENCES

[1] Distributed Multihead X Project, http://dmx.sourceforge.net

[2] ParaView on Multiple Processors, https://visualization.hpc.mil/wiki/Paraview_on_Multiple_Processors

[3] VisIt Visualization Tool, https://wci.llnl.gov/codes/visit/home.html

[4] J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. Law, and M. Papka: "Large-scale data visualization using parallel data streaming", *Computer Graphics and Applications, IEEE*, vol.21, no.4, pp.34-41, 2001.

[5] J. Ahrens, B. Geveci, C. Law, D. H. Charles, and R. J. Chris: ‘ParaView: An End-User Tool for Large-Data Visualization’: ‘Visualization Handbook’ (Butterworth-Heinemann, 2005), pp. 717-731

[6] R. Ball, and C. North: "Analysis of User Behavior on High-Resolution Tiled Displays," *Human-Computer Interaction, INTERACT 2005*, pp.350-363, 2005.

[7] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo: "VisTrails: enabling interactive multiple-view visualizations", *Visualization, 2005. VIS 05. IEEE*, pp.135-142, 2005.

[8] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre: "Remote large data visualization in the paraview framework", *Proceedings of the Eurographics Parallel Graphics and Visualization,* pp.162-170, 2006.

[9] M. Czerwinski, G. Smith, T. Regan, B. Meyers, G. Robertson, and G. Starkweather: "Toward characterizing the productivity benefits of very large displays," *Human-Computer Interaction, INTERACT 2003*, pp.9-16, 2003.

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

[11] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan: "Distributed rendering for scalable displays," *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, Dallas, Texas, United States, 2000.

[12] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski: "Chromium: a stream-processing framework for interactive rendering on clusters," *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, San Antonio, Texas, 2002.

[13] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, and J. Leigh: "High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment," *Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference*, pp.24-24, 2006.

[14] J. T. Klosowski, P. D. Kirchner, J. Valuyeva, G. Abram, C. J. Morris, R. H. Wolfe, and T. Jackman: "Deep view: high-resolution reality", *Computer Graphics and Applications, IEEE*, vol.22, no.3, pp.12-15, 2002.

[15] S. Nam, S. Deshpande, V. Vishwanath, B. Jeong, L. Renambot, and J. Leigh: "Multi-Application Inter-Tile Synchronization on Ultra-High-Resolution Display Walls," *Multimedia Systems*, Arizona, USA, 2010.

[16] K. A. Perrine, and D. R. Jones: "Parallel graphics and interactivity with the scaleable graphics engine," *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Denver, Colorado, 2001.

[17] W. J. Schroeder, L. S. Avila, and W. Hoffman: "Visualizing with VTK: a tutorial", *Computer Graphics and Applications, IEEE*, vol.20, no.5, pp.20-27, 2000.

[18] L. Shupp, R. Ball, B. Yost, J. Booker, and C. North: "Evaluation of viewport size and curvature of large, high-resolution displays," *Proceedings of Graphics Interface 2006*, Quebec, Canada, 2006.

[19] R. Singh, J. Byungil, L. Renambot, A. Johnson, and J. Leigh: "TeraVision: a distributed, scalable, high resolution graphics streaming system," *Cluster Computing, 2004 IEEE International Conference on,* pp.391-400, 2004.

[20] D. S. Tan, D. Gergle, P. Scupelli, and R. Pausch: "With similar visual angles, larger displays improve spatial performance," *Proceedings of the SIGCHI conference on Human factors in computing systems*, Ft. Lauderdale, Florida, USA, 2003.