DisplayCluster: An Interactive Visualization Environment for Tiled Displays

Gregory P. Johnson, Gregory D. Abram, Brandt Westing, Paul Navrátil and Kelly Gaither
Texas Advanced Computing Center (TACC)
The University of Texas at Austin
Austin, Texas
Email: {gregj, gda, bwesting, pnav, kelly}@tacc.utexas.edu

***Abstract*—DisplayCluster is an interactive visualization environment for cluster-driven tiled displays. It provides a dynamic, desktop-like windowing system with built-in media viewing capability that supports ultra high-resolution imagery and video content and streaming that allows arbitrary applications from remote sources (such as laptops or remote visualization machines) to be shown. This support extends to high-performance parallel visualization applications, enabling interactive streaming and display for hundred-megapixel dynamic content. DisplayCluster also supports multi-user, multi-modal interaction via devices such as joysticks, smartphones, and the Microsoft Kinect. Further, our environment provides a Python-based scripting interface to automate any set of interactions. In this paper, we describe the features and architecture of DisplayCluster, compare it to existing tiled display environments, and present examples of how it can combine the capabilities of large-scale remote visualization clusters and high-resolution tiled display systems. In particular, we demonstrate that DisplayCluster can stream and display up to 36 megapixels in real time and as many as 144 megapixels interactively, which is $3\times$ faster and $4\times$ larger than other available display environments. Further, we achieve over a gigapixel per second of aggregate bandwidth streaming between a remote visualization cluster and our tiled display system.**

***Keywords*-visualization; displays; high performance computing; human computer interaction; wide area networks;**

I. Introduction

In the age of big data, we will often need more pixels for data than a single display can provide. Device-generated data, such as from computed tomography (CT) machines and electron microscopes, can already yield images reaching into the gigapixels; petascale simulations can also require mega- or gigapixels to capture feature details; and digital media, such as digital documents or social media, can require significant screen space to display detailed relationship networks. In addition, these and other sources can contain distinct but correlated data views that benefit from simultaneous display. Any of these are difficult to accomplish on a single screen.

Fortunately, the continued development of commodity hardware has reduced prices to the point that many monitors can be used to create very high resolution displays at a reasonable cost. Recent displays constructed from commodity LCD monitors have eclipsed the three hundred megapixel barrier (Figure 1) [29], with a number of two hundred megapixel displays close behind [17]. Such displays are too large for conventional windowing software: DMX for X Windows on Linux can handle only sixteen separate clients [20], and Windows and MacOS handle only the displays connected to a single workstation.

A different windowing solution is necessary for these displays. Unfortunately, current windowing solutions for tiled displays [18], [34] can display images and movies at their native resolution, but they cannot display unmodified arbitrary applications. Custom API solutions [27], [11], [18], [19] enable an application to stream directly to a tiled display, though these require modification of source code which eliminates closed-source applications. Custom OpenGL libraries [23] allow distributed rendering and display, but only for a single OpenGL-based application at a time.

In this paper, we present DisplayCluster, a novel environment to drive tiled displays that combines the various features of previous display managers and adds novel features such as scriptability, smartphone and touchless interaction modes, massive pixel streaming from remote applications, multiple gigapixel image display and zoom and desktop application streaming to displays that allows any application to be shown interactively. We show that DisplayCluster can effectively display movies and gigapixel content simultaneously as well as stream hundreds of megapixels from a remote source interactively. We further demonstrate that DisplayCluster matches or exceeds the performance of other windowing environments, while providing more features. In particular, DisplayCluster can stream images, movies and applications up to 36 megapixels in real time and can stream up to 144 megapixels interactively, which represents a $3\times$ improvement in rendering speed and a $4\times$ improvement in size capability over other display environments. Further, we achieve over a gigapixel per second of aggregate bandwidth streaming between a remote visualization cluster and our tiled display system.

The rest of the paper is as follows: In Section II we discuss related work; we describe the architecture of DisplayCluster in Section III; then, in Section IV we discuss our tests and results; and we conclude with Section V.

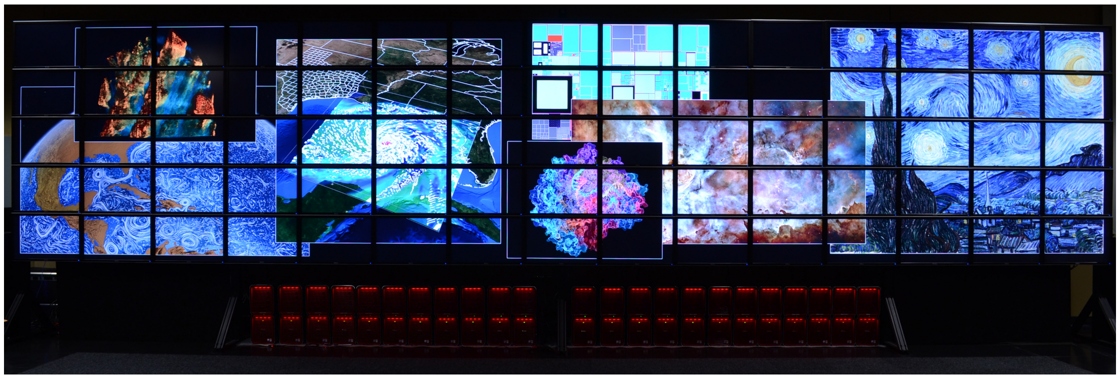

Figure 1. DisplayCluster running on Stallion, a 15 x 5 tiled display of 30-inch Dell LCD monitors providing 307 megapixels of resolution, at The University of Texas at Austin.

II. Related Work

Distributed display environments such as tiled wall or multi-projector systems have become common in laboratories, classrooms, and other settings. These environments are useful when a large display area is required [15], or a high number of aggregate pixels increase the user's ability to perceive detail and relationships in the data [10], [9], [13], [14], [36]. These benefits appear to scale with increased display size. Further, the physical navigation of large displays is a primary component of user productivity and satisfaction in large-scale visualization tasks [7], [8].

Below, we place DisplayCluster in the context of related work. We provide a comparative overview here, and we contrast specific features of DisplayCluster against other display environments in Section III.

A. Distributed Display Environments and Parallel Rendering Frameworks

A display environment for cluster-driven high resolution displays aims to provide a single usable workspace across the distributed system, in a way similar to the environment on a single workstation. One of the first distributed display environments was DMX [20], an extension to the X windowing environment. This enabled a fully-functional windowing environment across a cluster, but unfortunately DMX does not scale beyond sixteen tiles.

The Scalable Adaptive Graphics Environment (SAGE) [34] focused on a scalable but limited environment, departing from the full-featured approach of DMX. SAGE was developed with an architecture in which all pixels are streamed to the multi-node display from sources over the network. SAGE furthered this idea by providing the SAGE Application Interface Library (SAIL) [25] that enabled application instances to be streamed across multiple connections. While SAGE supports parallel streaming from multiple sources and provides for interaction from multiple input devices [24], the bandwidth requirements of its socket-based architecture limit its scalability over tens of megapixels, requiring high-speed dedicated network connections for interactive performance. Its streaming has been shown to scale up to 38 megapixels at 9.3 frames per second, on a dedicated 10-gigabit connection [25]. In addition, SAGE does not support dynamic zooming within images or videos.

In contrast to SAGE's streaming architecture, CGLX [18] provides a semi-transparent OpenGL-based graphics framework for distributed visualization systems. CGLX emulates a unified OpenGL context by intercepting and manipulating certain OpenGL directives. In this way, an application is run on each node within the cluster, and each node renders the portion of the view frustum according to the location of the node's displays within the total display structure. Another framework, Image Composition Engine for Tiles (IceT) [27], uses a sort-last parallel rendering library that can distribute image rendering in a distributed system. Equalizer [19] is a middleware library for parallel OpenGL-based applications that facilitates distributed rendering, load balancing, and virtual reality applications. In addition, Garuda [30] is a framework for transparently rendering any application written with OpenSceneGraph (OSG) across a tiled display. To use these middleware libraries, a user must modify the source of an application, which limits the set of applications to those where both the source is available and the user has the skill and time to implement the changes. An OpenGL intercept library, like WireGL [22] or Chromium [23], can distribute a single OpenGL application across multiple displays, but nothing else can be shown simultaneously and the available implementations have implemented only version one of the OpenGL standard.

The drawbacks of these display and rendering environments include limited scalability for streaming more than tens of megapixels, the necessity to modify application source code, and lack of support for displaying multiple applications simultaneously. DisplayCluster is designed to fill these gaps.

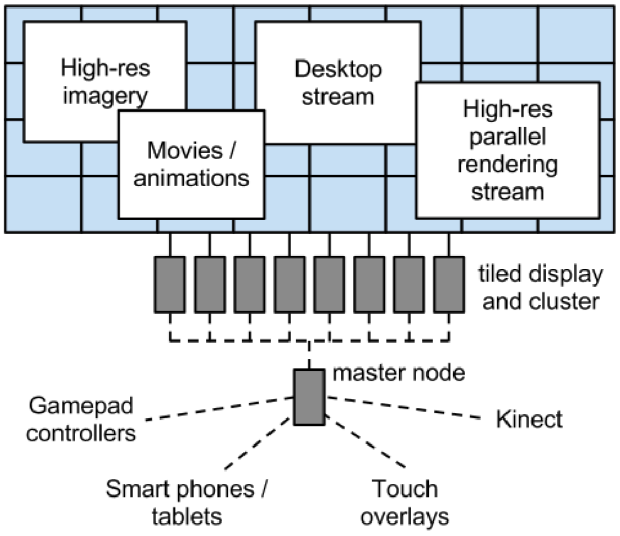

Figure 2. DisplayCluster supports media content and arbitrary applications through a streaming interface. Multi-user concurrent interaction is supported through several devices.

B. *Interacting with High Resolution Displays*

While studies show that large high-resolution displays have advantages in usability and productivity, unique concerns are raised in their use. These concerns stem from the user's ability to interact with the display system from both a position adjacent to the display with local interaction, and positions further from the display where the aggregate size of the display become important [21]. Solutions to these interaction problems and the study of their applicability have been proposed and implemented on many fronts. Touch-less interaction methods allow for the use of the system with gestures without physically touching the display surface or mouse [28], [38], [37]. Pointing techniques have been used to allow precise interaction at a distance by Lumipoint [16]. Furthermore, the application of touch interaction to large displays provides a natural usage scenario for effective interaction [39], [12], [32], [31]. DisplayCluster enables these display interaction modes, expanding the methods in which large tiled displays can be used.

III. Architecture

DisplayCluster provides a cross-platform, unified desktop-like environment for cluster-driven tiled displays. It executes in parallel across a cluster, taking control of the attached displays. A distributed windowing environment is then provided through which media content and arbitrary applications can be viewed and interacted with (Figure 2).

A. *Windowing Environment*

The windowing environment provided by DisplayCluster allows for content to be dynamically added, removed, and repositioned on the tiled display (Figure 3). This is similar to

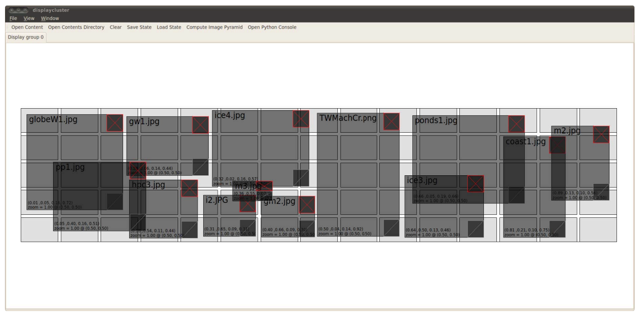

Figure 3. DisplayCluster's graphical user interface. The windowing environment allows content to be easily repositioned on a tiled display.

the approach used by SAGE [25], but differs from the static layouts provided by other tiled display environments [18], [23]. View transformations associated with window coordinates and tiled display layout are handled automatically. This simplifies rendering procedures for various content types, and also allows for window-level culling on individual DisplayCluster processes to improve performance and scalability.

Associated with each window in DisplayCluster is an object containing metadata describing the content it is to display and its windowing parameters. This metadata is managed by a master process, and synchronized to the other processes on the cluster. The individual cluster processes are then tasked with locating, processing, and rendering the content referenced by their visible windows. This separation allows for the compute-intensive workload to occur in parallel and asynchronously on the cluster while user interaction is taking place. Further, metadata for all windows can be saved to a state file, which can be later retrieved to restore a complete set of content on the tiled display, a feature lacking in other tiled display environments. This improves usability of the tiled display, allowing it to more easily be used in presentations and demonstrations.

Windows can also be zoomed and panned within for any supported content type. This is most widely used for high-resolution imagery, but also useful for animations and high-resolution streamed content (Figure 4).

B. *Media Viewing*

DisplayCluster provides built-in support for media content including imagery and video. Images of standard resolutions (up to 16 megapixels) are loaded by the appropriate image library and displayed directly. Similarly, video and animations are processed by the FFmpeg library [2].

High-resolution imagery (greater than 16 megapixels) is handled specially through a pyramiding process (Figure 5). The pyramiding process can either be done as a pre-processing step saving results to disk, or in real-time by the DisplayCluster cluster processes. An image pyramid consists of several groups of images, each group corresponding to

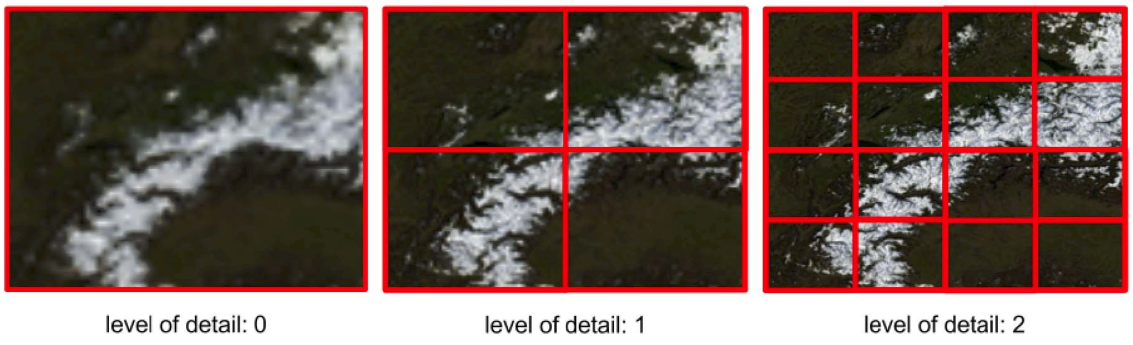

Figure 5. High-resolution imagery is handled through a pyramiding process. The appropriate level of detail is shown depending on the window size on the tiled display.

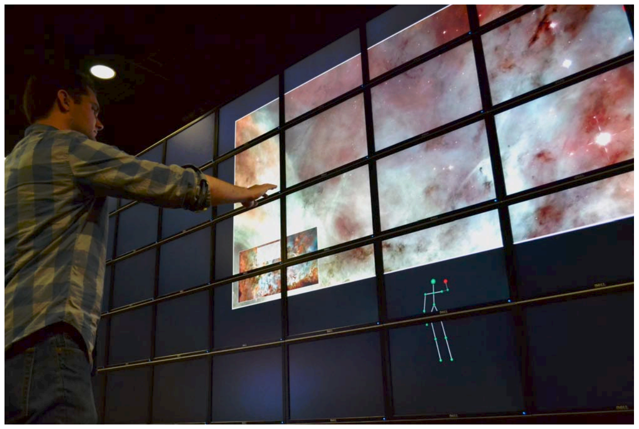

Figure 4. Windows can be zoomed and panned within for any supported content type. The zoomed area within the full image is shown to assist in navigation. The user shown is interacting with DisplayCluster via gestures and a Microsoft Kinect.

a level of detail. Each component image is of a fixed resolution (typically 512×512 pixels). The pyramiding process generates the images corresponding to these levels of detail from the original high-resolution image. The first (and lowest) level of detail is represented by a single image. Subsequent levels of detail divide each image region from the previous level of detail into quadrants, providing four times the resolution. The final level of detail contains the full resolution of the original image. An arbitrary number of such high-resolution images can be shown simultaneously. Other tools such as MagicCarpet (provided as a separate environment from the SAGE [25] developers) can only handle a single high-resolution image in a static layout. CGLX's image viewer can handle several high-resolution images simultaneously, but not together with other content types [18].

It is worth noting the scalability of the media viewing capability provided by DisplayCluster. Since the cluster nodes process the media content directly and off-screen windows are automatically culled, each process generally only handles a small subset of the displayed content. This approach takes advantage of the compute capabilities of the cluster nodes. Since each node decodes its own on-screen media content, more content can be shown on the entire tiled display simultaneously compared to streaming frameworks such as SAGE [25], where media content is decoded separately on a limited set of processes and then streamed over the network. Further, as opposed to SAGE, only compressed data is read from disk / network file storage, which reduces the required network bandwidth. This allows DisplayCluster to take advantage of high-speed parallel filesystems for sourcing content. This approach is similar to CGLX [18]; however since CGLX requires separate applications for viewing each content type, multiple content types can not be easily shown simultaneously as with DisplayCluster.

C. *Streaming*

DisplayCluster provides a network-based streaming interface that enables arbitrary applications to be shown on a tiled display. Applications provide image buffers which are compressed, sent over the network, and then received, decompressed and displayed by DisplayCluster. This pixel-based streaming approach can be integrated into existing applications with minimal modification. Further, unmodified applications can be streamed through a provided Desktop-Streamer application that captures a user-specified region of a desktop. The streaming interface supports resolutions upwards of a hundred megapixels, which makes it ideal for high-resolution scientific visualization applications.

Streamed content is displayed in a window on the tiled display in the same way media content is shown. Streams can provide content for a full window or for segments of a window. The set of window segments representing a full window can be sent over one or many network connections. This flexibility enables streamed content to be generated and compressed in parallel from a single process or from a set of distributed processes (Figure 6).

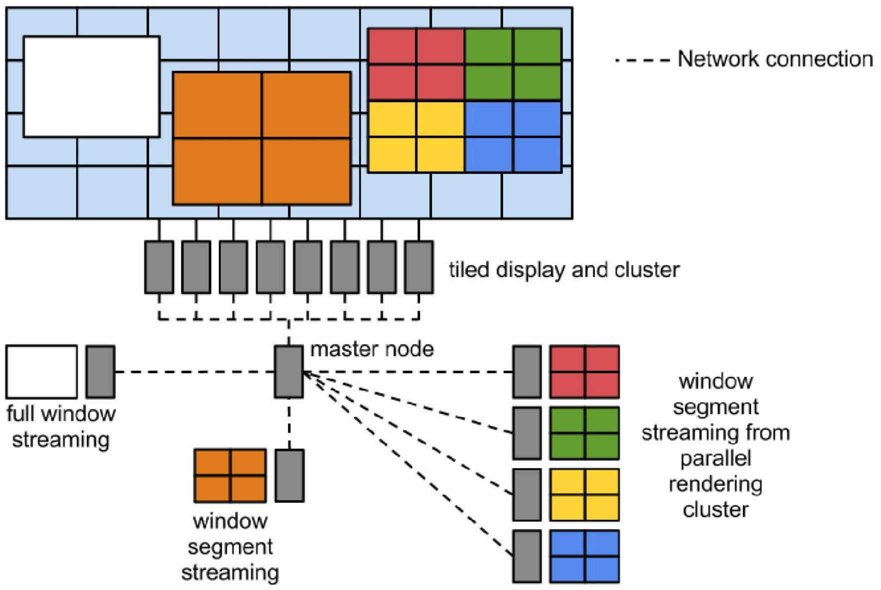
Network connection
tiled display and cluster
master node
full window streaming
window segment streaming
window segment streaming from parallel rendering cluster

Figure 6. DisplayCluster's streaming capability allows content to be streamed in segments and from one or many network connections.

Compression and decompression uses the libjpeg-turbo library [3], a derivative of libjpeg that uses SIMD instructions to accelerate JPEG conversion. JPEG compression reduces image data size significantly, lessening the required network bandwidth and allowing for higher resolutions to be streamed interactively. SAGE uses DXT compression, which has a fixed compression ratio of 6:1 [33]. In comparison, JPEG compression achieves a compression ratio of between 15:1 to 23:1 for high and medium image quality, respectively.

1) Parallel Visualization Applications: DisplayCluster provides a modified version of the Image Composition for Tiles (IceT) library [27] with streaming support. Visualization applications that use IceT can link with this modified version and stream directly to a running DisplayCluster instance. No modification of the underlying visualization application is required. This gives DisplayCluster direct support for existing scientific visualization applications that use IceT such as ParaView [6]. Additionally, IceT can be used as a framework for developing new DisplayCluster-enabled applications. These applications then have the flexibility to stream to DisplayCluster or be run independently. They can be run on the tiled display cluster, or on remote resources such as remote visualization clusters. Further, the applications can run using a single process at desktop resolution, or at scale using many processes giving resolutions of hundreds of megapixels. CGLX also provides an interface for developing OpenGL applications [18]. Although these applications can run interactively at the native resolution of the tiled display, they lack the data parallel capabilities of IceT, and cannot be run from remote resources. Further, such applications can only be run in the context of a running CGLX instance.

D. Interaction

DisplayCluster supports multiple types of control devices and allows for concurrent interaction from all connected devices. This allows multiple users to interact with DisplayCluster simultaneously, better supporting collaborative use of the tiled display. All interactions are handled by a master process, where each connected device has a corresponding cursor on the tiled display. When a device manipulates a window (moving, resizing, zooming or panning), a copy of the window object is created and manipulated. Changes to the window parameters are then communicated asynchronously to the primary window object and then to the tiled display cluster. In this way, user interactions do not conflict with each other. For example, one user can be zooming within a window while another pans within it at the same time.

DisplayCluster supports many types of input devices. Support for traditional mouse and keyboard interaction is provided at the console graphical user interface. Multi-touch interaction is supported through a TUIO library [26] network listener. Using TUIO as a layer of abstraction provides support for many types of multi-touch devices over the network. These devices include smart-phones, tablets, and large touch overlays. Gamepad controllers and other joystick devices are supported in DisplayCluster and provide a natural way to interact with the system in a wired or wireless modality.

Finally, touch-less interaction is supported in DisplayCluster through the OpenNI [4] natural interaction library

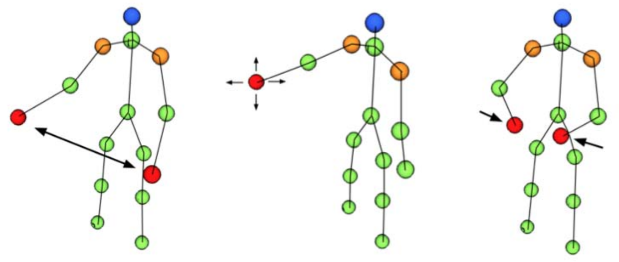

Figure 7. The skeletal representation of a user provides a basis for input gestures. Shown here are the zoom-out, pan, and zoom-in gestures, respectively.

and depth cameras such as the Microsoft Kinect. This interaction method provides for control of the windowing environment without the need to use any device other than the stationary sensor. The users present within the field of view (FOV) of the sensor are tracked and their skeleton joint locations are estimated in an automatic calibration step [35]. Gestures are interpreted from the skeletal representations (see Figure 7) to manipulate windows. These gestures allow users to move, resize, zoom and pan within windows. To provide visual feedback, skeletal representations of the users present within the FOV of the sensor are rendered on the tiled display as shown in Figure 4.

E. Scripting Interface

Tiled displays are often used as presentation or demonstration environments. For these use cases it is beneficial to have an interface to automate interaction with the tiled display software. DisplayCluster provides a fully-featured Python interface that allows users to script actions such as opening and removing windows of content, moving and resizing windows, and zooming and panning within windows. These actions can be timed to choreograph full demonstrations on the tiled display.

F. Platform Indepedence

In order to provide cross-platform portability of this wide range of features, DisplayCluster is built upon industry standard component libraries. The underlying framework of the DisplayCluster processes is built using features of the Boost and Qt libraries [1], [5]. In addition to providing the console graphical user interface for DisplayCluster, the Qt library is used for much of the internal framework including multi-threading, asynchronous messaging, and a platform independent interface to OpenGL. The OpenNI and TUIO libraries [4], [26] used for handling device interaction, including gamepad controllers, touch devices, and touch-less Kinect devices, are also platform portable.

The Message Passing Interface (MPI) is used for communication and synchronization between the DisplayCluster processes. This leverages the HPC community's efforts at

Table I
RESOLUTION, NUMBER OF PROCESSES, SEGMENT SIZE, FRAME RATE, AND PIXEL BANDWIDTH FOR TEST CASES CORRESPONDING TO FIGURE 8.

Resolution (megapixels)	Processes	Segment Size	Frames per Second	Megapixels per Second
1	1	1024^2	31	31
2	8	512^2	31	62
4	16	512^2	30	120
8	32	512^2	30	240
16	64	512^2	30	480
24	96	512^2	30	720
32	128	512^2	29	928
36	144	512^2	29	1044
48	48	1024^2	12.5	600
64	64	1024^2	10.4	665.6
96	96	1024^2	9.3	892.8
128	128	1024^2	8.4	1075.2
144	144	1024^2	4.9	705.6

optimizing message passing across high-performance networks and support for high-performance interconnects such as InfiniBand. MPI implementations typically get much higher performance than TCP- or UDP-based socket protocols across such networks. Further enhancements to networking technology will be addressed at the MPI level enabling DisplayCluster to take advantage. Other tiled display software environments such as SAGE and CGLX depend on socket level interfaces for communication, and therefore are limited in their support for higher speed technologies.

The use of these high-level component libraries enables DisplayCluster to be compiled across multiple operating systems. At The University of Texas at Austin, DisplayCluster runs tiled displays in both Linux- and Mac-based environments. Since these component libraries are also known to work in Windows, we expect DisplayCluster to work there as well. Additionally, the DesktopStreamer application used for remote streaming runs successfully on Windows, Mac and Linux operating systems.

IV. RESULTS

In this section we evaluate DisplayCluster's streaming performance for high-resolution scientific visualization applications from servers at remote sites to tiled displays.

We conduct our experiments on *Stallion*, a 307 megapixel tiled display located at the TACC ACES Visualization Laboratory on the campus of the University of Texas at Austin, and *Longhorn*, a large-scale remote visualization cluster located 9 miles away at the main TACC location on the UT Pickle Research Campus.

A. Hardware

Stallion, built in 2008, consists of 23 Dell XPS 720 render nodes and a Dell Precision 690 head node. Each render node has an Intel quad-core Q6600 CPU, two NVIDIA GeForce 8800 GTX GPUs and 4 GB of memory. The head node

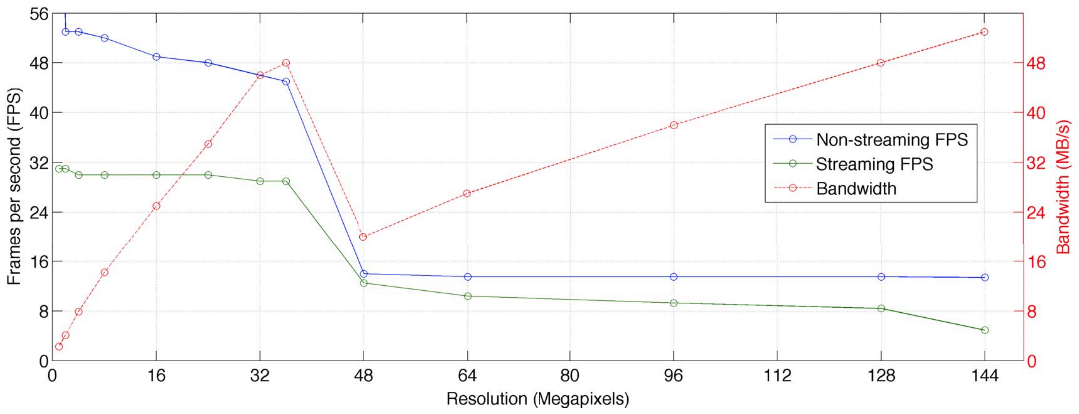

Figure 8. Performance of application streaming from the remote visualization cluster *Longhorn* to the tiled display *Stallion*. The non-streaming frame rate shows rendering performance independent of DisplayCluster, and the streaming frame rate shows the displayed frame rate on the tiled display.

has two Intel Xeon X5355 quad-core CPUs, two NVIDIA GeForce 8800 GTX GPUs and 16 GB of memory. The cluster has both an InfiniBand SDR interconnect and a gigabit Ethernet network. Stallion drives a 15 x 5 tiled display of 30-inch Dell LCD monitors, each at 2560 x 1600 resolution. In total, Stallion provides 307 megapixels of resolution, and is currently the highest-resolution tiled display in the world (Figure 1).

Longhorn is TACC's Dell XD visualization cluster. It has 256 compute nodes and 2 login nodes, with 240 nodes containing 48GB of RAM, 8 Intel Nehalem cores (@ 2.5 GHz), and 2 NVIDIA Quadro FX 5800 GPUs. Longhorn also has an additional 16 large-memory nodes containing 144GB of RAM, 8 Intel Nehalem cores (@ 2.5 GHz), and 2 NVIDIA Quadro FX 5800 GPUs. Longhorn provides 2048 compute cores, 13.5 TB aggregate memory and 512 GPUs.

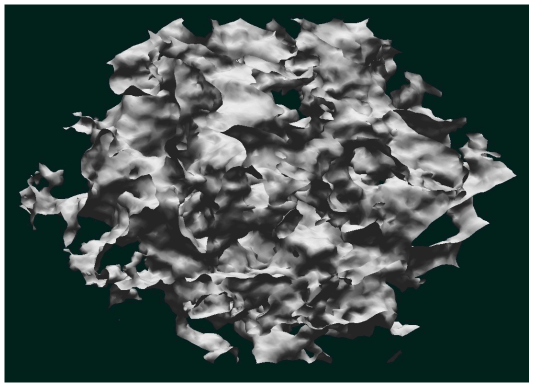

Figure 9. The example IceT application renders static geometry representing isosurfaces of a noise dataset in a rotating view. The application is run at varying resolutions on the *Longhorn* remote visualization cluster and streamed to DisplayCluster running on the *Stallion* tiled display system.

B. Experiments

We evaluate the performance of streaming high-resolution parallel rendering using DisplayCluster's modified version of the IceT library. The example IceT application renders static geometry representing isosurfaces of a noise data set in a rotating view (Figure 9). Although IceT can be used for sort-last data parallel applications, in this experiment we provide each rendering process with the entire dataset, thereby eliminating the costly compositing required for the data-parallel algorithm. This approach prevents compositing from becoming a bottleneck, and enables us to evaluate the performance impact of rendering and streaming high-resolution results to a remote DisplayCluster system. Our results thus demonstrate an upper-bound on performance; the actual performance of real-world applications will vary from this depending on the level of data parallelism (and thus compositing) and application-specific processing and rendering performance.

We conduct tests running the DisplayCluster-enabled IceT application on Longhorn streaming to Stallion at varying process counts and resolutions. In every case, eight rendering processes run on each allocated Longhorn node (one for each core) and share two graphics cards. In Figure 8 we show the speed of display, in frames per second, of rendered images of increasing size, as well as the application's frame rate without streaming to DisplayCluster. As we increase the size of the displayed image, we have altered the number of rendering processes and the sizes of the segments each must render. The number of rendering processes determines the number of streaming network connections to DisplayCluster, while the segment size determines the rendering load on Longhorn's GPUs. These were varied in an effort to maximize the streamed frame rate for each resolution; future work will provide a firmer basis for optimally specifying these parameters. In Table I we show the number of rendering processes and the sizes of tiles each renders for each full output resolution. We use a maximum of 144 processes and a segment size of 512^2 from 2 to 36 megapixels; at 48 megapixels we increase the segment size to 1024^2 for results up to 144 megapixels. A dip in frame rate occurs at 48 megapixels which corresponds to the increased rendering and image compression / decompression load of the 1024^2 segment size. Note that DisplayCluster's stream decoding on the rendering cluster is currently throttled to approximately 30 frames per second.

DisplayCluster supports nearly 30 frames per second up to 36 megapixel resolutions, and scales up to 144 megapixels at 4.9 frames per second, which represents a $3\times$ improvement in rendering speed over SAGE and a $4\times$ increase in size at comparable rendering rate. Further, we achieve over a gigapixel per second of aggregate bandwidth streaming between a remote visualization cluster and our tiled display system. In all test cases, the bandwidth used is under 53 megabytes per second. This allows DisplayCluster to be used for high-resolution application streaming on existing shared networks between supercomputing sites and tiled display locations. Further, on slower networks DisplayCluster will still be able to provide interactive performance at high resolutions. SAGE, the other commonly used tiled display streaming framework, has only been shown to scale up to 38 megapixel resolutions at 9.3 frames per second, with most experimental results at 16 megapixels or less [25]. Those results required a high-speed 10 gigabit dedicated network connection.

V. Conclusion

In this paper, we have described DisplayCluster, a novel environment for driving large tiled displays. DisplayCluster combines the features of previous display environments, for collaboration, application integration and image and video display, and offers better scalability and the ability to stream hundred megapixel images and native applications to remote tiled displays, and DisplayCluster directly supports visualization applications such as ParaView [6] through a commonly used parallel rendering framework. DisplayCluster is deployed on a variety of hardware configurations, and it has demonstrated cross-platform capabilities on Linux, MacOS and Windows. We have shown how it can combine the capabilities of large-scale remote visualization clusters and high-resolution tiled display systems.

We have shown that DisplayCluster's streaming capability scales to resolutions approaching that of the highest resolution tiled displays in the world. This is accomplished at frame rates supporting interactive visualization, without requiring high-speed dedicated network connectivity. We have shown that DisplayCluster handles up to 36 megapixel resolutions at near 30 frames per second, and scales up to 144 megapixels at 4.9 frames per second. Further, we achieve over a gigapixel per second of aggregate bandwidth streaming between a remote visualization cluster and our tiled display system. These results show both higher resolution and a higher degree of interactivity than previously published results.

In the future we plan to further enhance DisplayCluster's remote streaming capabilities, and provide better guidance for optimally determining streaming configurations. We also plan to continue exploration and integration of new interaction technologies and methods to enhance collaborative usage of large display environments. Further, we plan to examine DisplayCluster's performance on more recent and capable tiled display cluster hardware.

Acknowledgments

This work was funded in part by U.S. National Science Foundation grant OCI-0906379 and through the generous support of Dell.

References

[1] Boost C++ Libraries, http://www.boost.org/.

[2] FFMPEG Multimedia Framework, http://ffmpeg.org/.

[3] libjpeg-turbo JPEG Library, http://www.libjpeg-turbo.org/.

[4] OpenNI Natural Interaction Framework, http://www.openni.org/.

[5] Qt Framework, http://qt.nokia.com/products/.

[6] James Ahrens, Berk Geveci, and Charles Law. *The Visualization Handbook*, chapter ParaView: An End-User Tool for Large Data Visualization. Elsevier, 2005.

[7] R. Ball and C. North. Realizing Embodied Interaction for Visual Analytics through Large Displays. *Computers & Graphics*, pages 380–400, 2007.

[8] R. Ball and C. North. The Effects of Peripheral Vision and Physical Navigation in Large Scale Visualization. In *Graphics Interface*, pages 9–16, 2008.

[9] R. Ball, M. Varghese, B. Carstensen, E.D. Cox, C. Fierer, M. Peterson, and C. North. Evaluating the Benefits of Tiled Displays for Navigating Maps. In *International Conference on Human-Computer Interaction*, pages 66–71, 2005.

[10] Robert Ball and Chris North. Effects of Tiled High-Resolution Display on Basic Visualization and Navigation Tasks. *CHI 05 Extended Abstracts on Human Factors in Computing Systems*, 05:1196, 2005.

[11] A. Bierbaum, P. Hartling, P. Morillo, and C. Cruz-Neira. Implementing Immersive Clustering with VR Juggler. In *Computational Science and Its Applications*, pages 1119–1128, 2005.

[12] Andrew Bragdon and Hsu-sheng Ko. *Gesture Select : Acquiring Remote Targets on Large Displays without Pointing*, pages 187–196. ACM Press, 2011.

[13] M. Czerwinski, G. Smith, T. Regan, B. Meyers, G. Robertson, and G. Starkweather. Toward Characterizing the Productivity Benefits of Very Large Displays. In *Eighth IFIP International Conference on Human-Computer Interaction*, 2003.

[14] M. Czerwinski, D.S. Tan, and G.G. Robertson. Women Take a Wider View. In *ACM Conference on Human Factors in Computing Systems*, pages 195–201, 2002.

[15] Mary Czerwinski, Greg Smith, Tim Regan, Brian Meyers, George Robertson, and Gary Starkweather. *Toward Characterizing the Productivity Benefits of Very Large Displays*, pages 9–16. Number c. IOS Press, 2003.

[16] J Davis and X Chen. Lumipoint: Multi-User Laser-Based Interaction on Large Tiled Displays. *Displays*, 23(5):205–211, 2002.

[17] T.A. DeFanti, J. Leigh, L. Renambot, and et. al. The OptiPortal, a Scalable Visualziation, Storage and Computing Interface Device for the OptiPuter. *Future Generation Computer Systems*, 25:114–123, 2009.

[18] Kai-Uwe Doerr and Falko Kuester. CGLX: A Scalable, High-Performance Visualization Framework for Networked Display Environments. *IEEE Transactions on Visualization and Computer Graphics*, 17(3):320–332, 2011.

[19] S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: A Scalable Parallel Rendering Framework. *IEEE Transactions on Visualization and Computer Graphics*, 15:350–364, 2009.

[20] R.E. Faith and Kevin E. Martin. DMX: Distributed Multi-headed X, 2004.

[21] Otmar Hilliges and Lucia Terrenghi. Overcoming Mode-Changes on Multi-User Large Displays with Bi-Manual Interaction. *Proceedings of International Workshop on MultiUser and Ubiquitous User Interfaces*, 10(1):1–3, 2006.

[22] Greg Humphreys, Ian Buck, Matthew Eldridge, and Pat Hanrahan. Distributed Rendering for Scalable Displays. *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, 2000.

[23] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. In *Proceedings of SIGGRAPH Asia*, 2008.

[24] Ratko Jagodic, Luc Renambot, Andrew Johnson, Jason Leigh, and Sachin Deshpande. Enabling Multi-User Interaction in Large High-Resolution Distributed Environments. *Future Generation Computer Systems*, 27(7):914–923, 2010.

[25] Byungil Jeong, Luc Renambot, Ratko Jagodic, Rajvikram Singh, Julieta Aguilera, Andrew Johnson, and Jason Leigh. High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment. *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, (November):24–24, 2006.

[26] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. TUIO : A Protocol for Table-Top Tangible User Interfaces. *Neuroinformatics*, 2005.

[27] K. Moreland, B. Wylie, and C. Pavlakos. Sort-Last Parallel Rendering for Viewing Extremely Large Data Sets on Tile Displays. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 85–92, 2001.

[28] Mathieu Nancel, Julie Wagner, Emmanuel Pietriga, Olivier Chapuis, and Wendy Mackay. Mid-Air Pan-and-Zoom on Wall-Sized Displays. *Design*, (May):177–186, 2011.

[29] Paul A. Navrátil, Brandt Westing, Gregory P. Johnson, Ashwini Athyle, Jose Carreno, and Freddy Rojas. A Practical Guide to Large Tiled Displays. In *International Symposium on Visual Computing*, 2009.

[30] Nirnimesh, P Harish, and P J Narayanan. Garuda: A Scalable Tiled Display Wall Using Commodity PCs. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):864–877, 2007.

[31] Peter Peltonen, Esko Kurvinen, Antti Salovaara, Giulio Jacucci, Tommi Ilmonen, John Evans, Antti Oulasvirta, and Petri Saarikko. *Its Mine, Don't Touch!: Interactions at a Large Multi-Touch Display in a City Centre*, volume 16, pages 1285–1294. ACM, 2008.

[32] Kevin Ponto, Kai Doerr, Tom Wypych, John Kooker, and Falko Kuester. CGLXTouch: A Multi-User Multi-Touch Approach for Ultra-High-Resolution Collaborative Workspaces. *Future Generation Computer Systems*, 27(6):649–656, 2010.

[33] Luc Renambot, Byungil Jeong, and Jason Leigh. Real-Time Compression For High-Resolution Content. *Proceedings of the Access Grid Retreat*, 2007.

[34] Luc Renambot, Arun Rao, Rajvikram Singh, Byungil Jeong, Naveen Krishnaprasad, Venkatram Vishwanath, Vaidya Chandrasekhar, Nicholas Schwarz, Allan Spale, Charles Zhang, Gideon Goldman, Jason Leigh, and Andrew Johnson. SAGE: the Scalable Adaptive Graphics Environment. In *Fourth Annual Workshop on Advanced Collaborative Environments (WACE)*, 2004.

[35] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. *IEEE Conference on Computer Vision and Pattern Recognition (2008)*, 2(3):1297–1304, 2011.

[36] L. Shupp, C. Andrews, M. Kurdziolek, B. Yost, and C. North. Shaping the Display of the Future: The Effects of Display Size and Curvature on User Performance and Insights. *Human-Computer Interaction*, 24(1), 2009.

[37] Daniel Stødle, Tor-magne Stien Hagen, John Markus Bjørndalen, and Otto J Anshus. Gesture-Based, Touch-Free Multi-User Gaming on Wall-Sized, High-Resolution Tiled Displays. *Journal of Virtual Reality and Broadcasting*, 5:0009–6, 2008.

[38] Daniel Vogel and Ravin Balakrishnan. Distant Freehand Pointing and Clicking on Very Large, High Resolution Displays. *Proceedings of the 18th Annual ACM Symposium on User interface Software and Technology*, 10(3):33, 2005.

[39] Brandt Westing, Benjamin Urick, Maria Esteva, Freddy Rojas, and Weijia Xu. Integrating Multi-Touch in High-Resolution Display Environments. In *2011 International Conference for High Performance Computing Networking Storage and Analysis SC*, pages 1–9. IEEE, 2011.