MassivePixelEnvironment: A Tool for Rapid Development with Distributed Displays

Brandt Westing
TACC Visualization Lab
201 E 24th St
Austin, TX 78712 USA
bwesting@tacc.utexas.edu

Heriberto Nieto
TACC Visualization Lab
201 E 24th St
Austin, TX 78712 USA
hnieto@tacc.utexas.edu

Robert Turknett
TACC Visualization Lab
201 E 24th St
Austin, TX 78712 USA
turknett@tacc.utexas.edu

Kelly Gaither
TACC Visualization Lab
201 E 24th St
Austin, TX 78712 USA
kelly@tacc.utexas.edu

Abstract
We describe a software library called the MassivePixelEnvironment (MPE) for use in visualization, arts, humanities, and interface prototyping in distributed display environments. We discuss the implementation of the software and its unique qualities when contrasted with other distributed graphics libraries and environments in the areas of interaction, rapid development, and rich library support. We provide concrete examples of its usage at multiple sites, lessons learned, and a discussion on the future of tiled display environments.

Author Keywords
distributed graphics; tiled displays; visualization; interface prototyping; arts; humanities

ACM Classification Keywords
I.3.2 [Graphics Systems]: Distributed/network graphics

General Terms
Design; Human Factors

Introduction
Software development with tiled displays can become cumbersome when the number of compute resources required to drive the display(s) is greater than one. It is necessary for a cluster, or multiple compute resources, to
drive a display system when a single computer lacks the necessary graphics or computation capabilities to drive the display(s) alone. These systems are difficult to use because of the lack of general purpose software developed for distributed systems. Several libraries and algorithms have been developed to ease the burden of graphical application programming for such systems. However, we have found these frameworks cumbersome, invasive, or unsatisfactory for novice users. Past developments have proved to be particularly difficult for non-computer scientists to use because of the knowledge necessary to navigate low-level programming interfaces.

MPE was developed to solve problems that are seen often with distributed display systems: How can users be enabled to quickly create visualizations, researchers to easily test interface prototypes for interaction studies, and non-computer scientists the functional literacy to utilize distributed displays effectively?

**Background**

Distributed display systems come in many shapes and sizes. Seamless displays can be created from multiple projectors driven by multiple compute resources. Majumder[10] describes the practical design of multi-projector systems. Tiled Liquid Crystal Displays (LCD) can be used to create high resolution and high aggregate pixel count displays suitable for high resolution visualization. Countless tiled LCD displays exist, the largest of which include the RealityDeck[12], Stallion[11], and the HIPERWall OptiPortal[3].

The development of software for these display systems has been as varied as the systems themselves. Three common architectures have been employed to simplify the usage of distributed displays: pixel streaming, sort first non-invasive, and sort first invasive rendering. Pixel streaming architectures perform computation and rendering at one or more compute resources and stream the rendered pixels to rendering resources that then display the pixel stream images. Pixel streaming architectures require high computational ability at the application node (which performs computation and rasterization), and large available bandwidth for the pixel streams - especially for dynamic content. The Scalable Adaptive Graphics Environment (SAGE)[7] is an example of a pixel streaming architecture. Chromium[6], in its tile-sort mode, is a sort first non-invasive architecture for distributed rendering. Chromium intercepts the OpenGL stream from the application node(s) and streams the commands to the render nodes, which render a section of the final image shown on the display. Sort first invasive architectures are those that require calls within the application code (invasive) for distributed rendering to function. These architectures have a complete graphics pipeline at each render resource. Representative sort first invasive architectures include cglX[4], DisplayCluster[8]1, Equalizer[5], and MPE (described here). MPE, however, is the only sort first invasive architecture based around a higher-level programming language with ease-of-use and a visualization focus as primary considerations.

Importantly, MPE is inspired in function and namesake by an earlier work by Daniel Shiffman[13] intended for distributed displays with Processing. Unlike Shiffman’s previous work, the work presented here provides support for the latest Processing versions, high numbers of distributed hosts, optimized synchronization, and easier configuration through the use of a single configuration file.

---

1In the parallel streaming mode. Normal operation displays only static content such as images and video.
The Processing Language
Processing is a programming environment that was developed in 2001 to promote software literacy in the visual arts[1]. It is a free and open-source programming language and development environment that was originally developed to teach fundamentals of computer programming, but quickly developed into a tool for creating professional work. Processing abstracts complicated programming concepts and simplifies the application work flow by: hiding compilation, linking, and running the program executable; by implementing a scripting layer on Java; and by providing a simplified development environment. Furthermore, Processing provides a programming construct broken into two functional components that abstract the control loop found in most applications.

Figure 1: The MPE rendering process uses a centralized barrier synchronization method over TCP.

Implementation
MPE provides an abstraction to the distributed environment and aims to make the jump from serial to parallel programming as simple as possible for the developer. As such, development with MPE differs from developing with a serial Processing environment by only two ways: the configuration description file and a small number of MPE setup calls.

The Configuration File
The configuration file is a single XML file that describes the distributed display system. The file specifies the host machines involved in the distributed system, the displays associated with the hosts, and the resolution and orientation of the displays. The file serves to enable the software to properly set the view frustum and configure synchronization among the rendering clients.

The MPE Setup and Rendering Process
When running a distributed program using MPE and Processing, the user makes an MPE function call within the Sketch (main program) setup code that starts a communication thread that runs in parallel to the main thread. The communication thread handles synchronization between the hosts in the distributed system. Specifically, a centralized barrier synchronization method is used over TCP sockets between a leader process and the following or rendering processes. Communication messages are simple 8-bit messages that represent a frame event or done event. The frame events are sent from the central process and allow the rendering client(s) to unlock their frame lock (whereupon the scene is rendered), while the done events (sent from the followers) allow the central process to provide a barrier until all rendering clients have rendered. When the leader process receives all the done events, it broadcasts a frame event, allowing the rendering clients to resume rendering.
event. This looping process can be seen in Figure 1.

Before the frame is rendered by a follower, the view frustum is culled such that the rendered scene reflects only the portion of the display area the display is responsible for. For instance, a machine driving a quadrant of a distributed display (1 screen out of 4), will render only the part of the scene that will be viewable in the quadrant.

Results
We provide three usage examples of MPE in tiled display settings. Furthermore, we contrast the usage of MPE against other distributed display libraries and toolkits.

One: Interactive Visualization of HPC Queues
Using Paul Bourke’s visualization of HPC queue statistics as an inspiration[2], the Texas Advanced Computing Center’s Ranger Supercomputer queue is visualized in real-time on the Stallion 328 mega-pixel display (shown in Figure 2, top left). MPE is used to facilitate this visualization on the Stallion system, which is composed of 21 distributed hosts. The visualization shows usage of Ranger’s 4000 compute nodes, showing 100’s of active jobs and features control via multi-touch gestures from a tablet. Multi-touch is enabled through the application of the TUIO[9] protocol to Processing. The resolution of Stallion allows the complete Ranger queue to be visualized and spatially explored.

Two: Scientific Visualization of Antarctic Ice Sheets
Light detection and ranging (LIDAR) data from Antarctica’s coastline ice sheets is an important source of information on climate change. The time varying data can reflect changing temperatures on earth. The Systems Ecology Laboratory at the University of Texas at El Paso, along with researchers at TACC, used MPE to visualize LIDAR datasets on the 45-node Amethyst tiled display (Figure 2, top right) at the University of Texas at El Paso. The interactive visualization served to allow researchers to collaboratively and spatially explore the data collected and envision the possibility of comparing large sets of LIDAR data on Amethyst to better understand climate change on earth.

Figure 2: Visualizations using MPE. Top Left: Interactive Visualization of HPC Queues. Top Right: Antarctic Ice Sheet Visualization. Bottom: Structural Visualization of Shakespeare’s The Tempest.
Three: Structural Visualization of The Tempest

Processing is a powerful tool for information visualization and has proven to be useful to researchers who need to quickly visualize the organization of data and/or its structure. This visualization illustrates the structure of text, and is called the Text Universe. The Text Universe shown here is an excerpt from Shakespeare's The Tempest, revealing the structure of Shakespeare's prose using a node graph (Figure 2, bottom). Inspired by a prototype visualization by Tiemen Rapati, this visualization is intended to motivate humanities and arts researchers or students to take advantage of distributed displays as a new medium for exploration, exposition, and insight. The visualization is shown running on TACC's Stallion tiled display.

Discussion

MPE directly compares to the Cross Platform Cluster Graphics Library (cglX): they are both libraries used to develop applications for distributed displays with a sort first invasive architecture. cglX is implemented in C++ and provides an API at the OpenGL level. cglX provides the same type of synchronization as MPE, and operates over TCP sockets; therefore their execution requirements and messaging overhead are similar. cglX may be preferred by a developer who works at a lower level, while MPE will clearly be preferred by those with less experience, less time, or desire the rich libraries that Processing has to offer for visualization, input, and data processing. When compared to pixel streaming architectures, MPE uses far less bandwidth due to its small synchronization messages (compared to pixel content). It is latency sensitive, as are the pixel streaming architectures in SAGE and DisplayCluster. Compared to the sort first non-invasive architecture of Chromium, MPE requires less bandwidth and has similar latency constraints. MPE programs must be modified in the setup phase of a Sketch, unlike Chromium which intercepts OpenGL calls from the application and can run application unmodified.

The display architectures mentioned above all have positive and negative attributes. While these libraries and middle-ware have an important place in distributed display systems, there is a current trend in the capability of graphics hardware to drive a greater number of pixels. The adoption of display standards such as DisplayPort allow graphics pipelines to driver higher amounts of pixels across a single output, and hardware vendors are supporting an increasing number of ports per graphics card. It is possible now to see display systems driven by a single computer that drive a large number of displays. In these systems, there is no need for distributed display middle-ware and native applications may run unimpeded (clearly a winning scenario for developers who do not wish to modify code-bases). It remains to be seen how these systems will evolve and whether distributed display architectures (as mentioned here) will be needed in the future to drive tiled displays. It is certain, however, that distributed display software will be an important part of the future of display environments: wearable displays and ubiquitous display environments both require the notion of distributed display architectures if they are to function as a community of devices.

Conclusion

The MassivePixelEnvironment (MPE) is a tool for rapid development with distributed display systems. MPE abstracts the parallelism of such an environment from the user and allows for the creation of visualizations, deployment of interface prototypes, and non-expert use of systems that are inherently challenging to develop for. The architecture of MPE is scalable: MPE has been
tested on clusters of up to 50 nodes and on a distributed display with an aggregate resolution of 328 megapixels. MPE is an open-source, free library for the Processing programming language. The toolkit requires only 3-5 lines of code to be added to a developer’s visualization, and is therefore relatively easy for a developer to extend the visualization from a laptop or workstation to a wall-sized display.

Acknowledgments
This work was made possible by funding from the National Endowment for the Humanities (NEH) Grant: HD-51475-11, A Thousand Words: Advanced Visualization for the Humanities. We would like to thank Greg Abram of the Texas Advanced Computing Center for creating the Antarctic ice sheet visualization in Processing and Craig Tweedie of the Systems Ecology Laboratory at The University of Texas at El Paso for the LIDAR data used in the visualization.

References