Multiapplication, Intertile Synchronization on Ultra-High-Resolution Display Walls

Sungwon Nam,[1] Sachin Deshpande,[2] Venkatram Vishwanath,[3] Byungil Jeong,[4]
Luc Renambot,[1] Jason Leigh[1]

[1] Electronic Visualization Laboratory
842 W. Taylor St.
Chicago, IL 60607

snam5, luc, spiff@evl.uic.edu

[2] Sharp Laboratories of America
5750 NW Pacific Rim Blvd.
Camas, WA 98607

sdeshpande@sharplabs.com

[3] Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439

venkatv@mcs.anl.gov

[4] Texas Advanced Computing Center
10100 Burnet Rd., Bldg. 196
Austin,TX 78758

bijeong@tacc.utexas.edu

ABSTRACT

Ultra-high-resolution tiled-display walls are typically driven by a cluster of computers. Each computer may drive one or more displays. Synchronization between the computers is necessary to ensure that animated imagery displayed on the wall appears seamless. Most tiled-display middleware systems are designed around the assumption that only a single application instance is running in the tiled display at a time. Therefore synchronization can be achieved with a simple solution such as a networked barrier. When a tiled display has to support multiple applications at the same time, however, the simple networked barrier approach does not scale. In this paper we propose and experimentally validate two synchronization algorithms to achieve low-latency, intertile synchronization for multiple applications with independently varying frame rates. The two-phase algorithm is more generally applicable to various high-resolution tiled display systems. The one-phase algorithm provides superior results but requires support for the Network Time Protocol and is more CPU-intensive.

Categories and Subject Descriptor

I.3.2 [**Computer Graphics**]: Graphics Systems – *Distributed/network graphics*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Client/server*; D.4.1 [**Operating Systems**]: Process Management – *Synchronization*

General Terms

Algorithms, Performance

Keywords

Frame synchronization, Tiled display, Cluster computing

MMSys'10, February 22-23, 2010, Phoenix, Arizona, USA.

1. Introduction

Ultra-high-resolution display walls are fast becoming a standard tool for scientific research. These types of displays are the only means by which scientists can see the massive data generated from their instruments and supercomputer simulations. With the advent of low-cost LCDs, researchers are now using tiled-display walls as "mash up" environments where they can juxtapose a variety of data so that they can look at them as a whole [22]. While similar to the notion of Project Rooms or War Rooms, a key difference is that for large-scale and collaborative scientific research, there is no other way to look at this magnitude of data. These projects routinely deal with time-varying data on the order of terabytes to petabytes. It is impossible to manage this information by printing out static diagrams on sheets of paper and pinning them to a wall, as has been the traditional approach. The microscopes and telescopes used by scientists today t are integrated with complex computing systems that perform noise filtering, tiling, and feature detection. Ultra-high-resolution displays are becoming the new lenses that bring the data from these instruments into focus.

To meet this challenge, the Electronic Visualization Laboratory at University of Illinois at Chicago has been conducting research with Sharp Laboratories of America on scalable display-wall hardware and software. The culmination of this work is LambdaVision, a 100-megapixel LCD wall, and the Scalable Adaptive Graphics Environment (SAGE) [22], a middleware system for driving such walls. Figure 1 shows the LambdaVision driven by SAGE that is used for weekly meetings in the Electronic Visualization Laboratory.

LambdaVision is an array of 55 LCD panels driven by a cluster of 28 computers. The computers cooperate to give users the

illusion of a seamless display environment. Therefore, precise coordination and synchronization between the computers are necessary to ensure that the animated images displayed on the wall appear seamless. The Scalable Adaptive Graphics Environment (SAGE) has been developed for this purpose [35]. Unlike other tiled-display middleware such as Chromium [20] or Equalizer [16], SAGE was designed at the outset to manage multiple images or animations from different applications at the same time, enabling users to simultaneously access, stream, and juxtapose them on ultra-high-resolution tiled-display walls.

The need to support multiple applications at the same time poses a significant challenge for image synchronization. Traditional frame synchronization mechanisms used in systems such as Chromium or Equalizer do not scale because of the increased message complexity when supporting multiple applications simultaneously. In this paper, we propose and validate two new algorithms: a two-phase and a one-phase frame synchronization algorithm to achieve low-latency, intertile synchronization for multiple applications with varying frame rates. The two algorithms achieve the same goal but differ in resource utilization and complexity. A key contribution of this paper is proposing a scalable way to achieve frame synchronization among display nodes in a tiled display wall that can support multiple applications simultaneously.

Figure 1. The SAGE in action. A student presenting at a weekly meeting in the Electronic Visualization Laboratory.

In the following two sections, we discuss related work and describe in greater detail the synchronization requirements of ultra-high-resolution environments. We then present background information about the SAGE. We also detail the limitations of applying traditional synchronization approaches to SAGE, and we describe how our new algorithms provide significant improvements.

2. Related Work

The traditional model for driving tiled display walls was to use the entire surface to display a single visualization in exquisite resolution. However, as display walls began to grow in size and resolution, users found it more useful to be able to use the expansive screen estate and resolution for displaying not only a single visualization but multiple visualizations simultaneously so that they can be compared side by side [6, 12, 34, 36]. Middleware systems in the former category include WireGL [19], Chromium [20], DMX [2], Equalizer [16], and CGLX [1]. In WireGL and its successor Chromium, one or more servers convert data from unmodified applications into OpenGL graphics primitives, which are then streamed to clients driving the tiled-display wall. CGLX does not distribute graphics primitives but runs the same copies of the OpenGL-based application on all clients and replicates the data on all the clients. Equalizer offers a hybrid approach where the user can combine various rendering techniques. DMX (Distributed Multihead X Project) provides an X Window system compatible environment where multiple displays connected to multiple machines are presented as a single unified screen. In DMX, a master node distributes X Window primitives on a tiled display.

All of these approaches assume that a single application occupies the entire wall at any given instant. Rendering of the content is often conducted directly on the graphics cards that are connected to the displays. This approach has the advantage that it enables low-latency generation and manipulation of the images. Moreover, the frame synchronization among display nodes can be achieved easily by implementing a networked barrier at the point that needs to be synchronized. A networked barrier works by having all display nodes send a message to a barrier server node and wait while the barrier server counts the number of messages it has received. When all the messages have been received the barrier server broadcasts acknowledgements to every display node, which upon receipt unblocks the display nodes. The synchronization barrier can be implemented in several ways. In cluster computing, the Message Passing Interface (MPI) [3] is the de facto communication mechanism among the nodes. MPI supports a "barrier" among all its communication nodes, which ensures that all progress is blocked until all the processes running on the nodes enter that barrier. This approach is sufficient to ensure frame synchronization for single applications occupying an entire tiled display.

Total image synchronization across displays in a tiled-display wall is best achieved through a combination of hardware and software solutions and is required for the display of stereoscopic images [28]. In terms of hardware, the synchronization of vertical refresh across multiple graphics cards can be achieved by using specialized genlock hardware built into advanced graphics cards such as those found in Nvidia's Quadro series [25-27]. Alternatively, Allard et al. [4] presented a cost-effective approach using custom hardware and parallel port for distributing vertical refresh synchronization signals. This approach can be used with any graphics hardware. Combined with software synchronization methods presented in many papers, these can provide a cost-effective solution for total frame synchronization between display nodes.

Chen et al. [8] discussed three communication methods for a cluster-based, high-resolution display system: a synchronized execution model where all render nodes have the same copy of the application instance (e.g., CGLX), a primitive distribution model where a client distributes graphics primitives to render servers (e.g., Chromium), and a pixel distribution model where a client renders and transmits only pixels to display servers. A synchronization barrier at a certain program execution point (such as before graphics buffer swap) can be directly used to ensure frame synchronization in the synchronized execution model, such as in [5, 8, 9, 17, 37]. In the primitive distribution model, which can also be considered as a centralized model in that only a single node that has application instance distributes

graphics primitives or pixels to server nodes that render and display, the frame synchronization can be achieved implicitly, leaving small asynchronies between display nodes [29], or a synchronization barrier can be used explicitly such as in [16, 19, 21, 24] for tighter frame synchronization. SAGE can be categorized as a *parallel* pixel distribution model, since multiple clients (applications) send pixels to multiple display servers (display nodes). In the earlier generations of SAGE, we implemented a frame synchronization scheme to support multiple applications by using multiple synchronization barriers, one barrier per application. However, this per-application-based synchronization scheme was unable to provide the tight synchronization tolerances expected by display manufacturers such as Sharp. Our new approaches provide significant improvement.

3. Frame Synchronization Requirements of a Tiled-Display Wall

In order to display a continuous image on a tiled display, all the tiles that constitute an application window need to be synchronously updated. This synchronization is especially important for interactive visualizations and animations.

There are three requirements for seamless frame synchronization on tiled displays:

1. Data synchronization: The application data to be displayed must be coherent. That is, the various display nodes must display parts of the same frame. For multiple applications, data synchronization must be achieved for each application being displayed.
2. Swap buffer synchronization: The display thread on each node should swap the contents of the graphics buffer synchronously in order for the various application windows to appear consistent on the display.
3. Synchronization of the vertical refresh cycles of the various displays (gen-lock): the physical refresh of monitors on each node should occur synchronously.

Perfect frame synchronization on tiled displays is achieved by satisfying all three requirements. In this paper we focus on all but the third requirement, which can normally be achieved through the use of specialized hardware.

In the case of dedicated tiled-display walls that are limited to running only a single application at a time, data and swap buffer synchronization can be ensured easily with a single synchronization barrier [5, 7, 9, 10, 17-19, 21, 24, 28, 30, 32, 33]. However, the problem we are attempting to solve is more challenging because tiled-display walls can have an arbitrary number of different application windows in which frame updates occur at different rates. If a frame synchronization method for tiled-display system that runs single application is applied, it becomes per-application-based synchronization, which is not scalable becausse of excessive synchronization messages over the network generated for each frame from each application. Also, with per-application-based synchronization, it is difficult to obtain swap buffer synchronization across display nodes because each application sends frames at different rates. Thus, the events (data and swap buffer synchronizations) of the same application can become partially ordered on a cluster, thereby leading to unsynchronized display of frames. For multiple applications to be displayed seamlessly on a tiled-display wall, total ordering of data and swap buffer synchronization across all applications is required. The total ordering of events in a distributed system is described in detail in [23].

In Section 5, we propose two scalable frame algorithms that

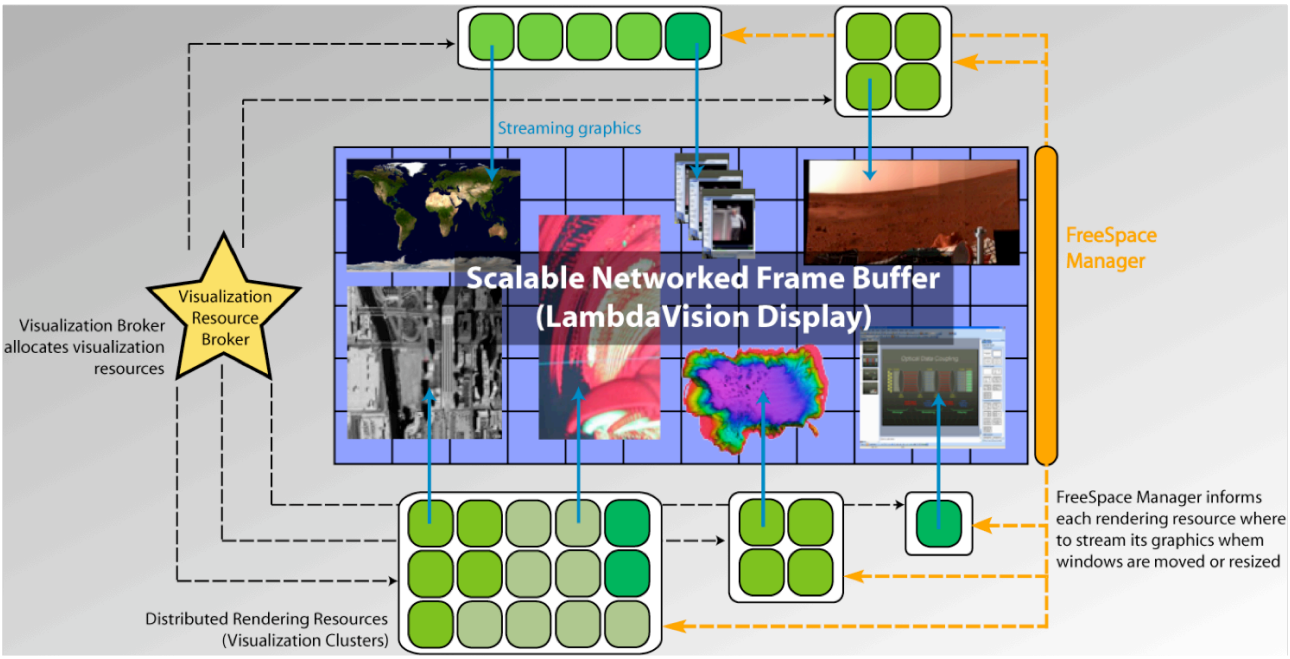

Figure 2. In SAGE, while a compute cluster drives the individual displays, it merely acts as a lightweight client that receives pixels from remote rendering resources such as visualization clusters or supercomputers.

ensure total ordering of data synchronization of all applications and swap buffer synchronization between display nodes, with minimal impact on applications' frame rate and latency.

4. Scalable Adaptive Graphics Environment

SAGE is a cross-platform middleware system for driving ultra-resolution tiled displays. Unlike other 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 used to render datasets that may be too large to fit on an individual graphics card [13]. In SAGE, a single window displayed on a wall may be driven by any number of display nodes, and multiple windows can be displayed on the wall simultaneously. As windows are moved from one portion of the wall to another, SAGE seamlessly reroutes the pixels to a different set of computers driving the display tiles so that handling of the windows on the display is totally transparent from the application. The SAGE model is shown in Figure 2.

4.1 Architecture

In SAGE, the application pixel streams are received by the SAGE Application Receiver threads (APP). Each application gives its rendered pixels to the SAGE Application Interface Library (SAIL), which streams them to the appropriate display nodes depending on the current position and size of the window on the tiled display. Each node has a Node Display Manager (NDM) responsible for displaying the contents of all applications on the display; multiple pixel streams can be displayed independently to allow multiple applications to be shown concurrently on the tiled display. The Free Space Manager (FSManager) keeps track of the current display parameters and the arrangement of the application pixels on the tiled display. Based on the requested arrangement, the FSManager directs SAIL to distribute an application's pixels to the appropriate display nodes. The applications can be dynamically moved and resized with the help of the UI client. An example of SAGE session that runs on four display nodes and displays two applications is depicted in Figure 3.

4.2 SAGE's Frame Synchronization Algorithm

In this section, we discuss the frame synchronization method we initially used for SAGE: a dynamic networked barrier per application for data synchronization.

In this method a synchronization group (SyncGroup), which consists of a set of display nodes that shows an application's image, is maintained for each application. Members (display nodes) in the group can be dynamically changed as a user moves or resizes an application window. And a data synchronization manager thread, which ensures synchronized frame update of display nodes in the SyncGroup, is created for each synchronization group. A dynamic SyncGroup and data synchronization manager thread pair implements a dynamic barrier for each application. An example of the data synchronization manager and synchronization group pairs in SAGE is shown in Figure 4.

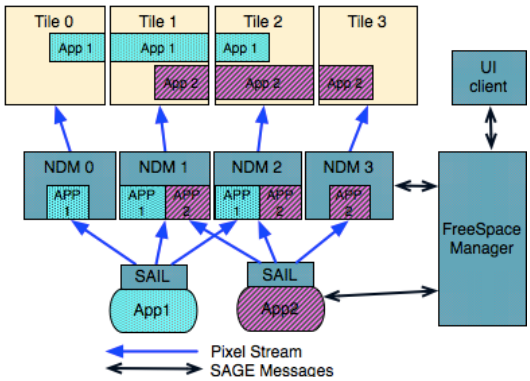

Figure 3. The SAGE components. Shown is an example of four display nodes running two applications, App1 and App2 each distributed on tile 0, 1, and 2 and tile 1, 2, and 3 respectively.

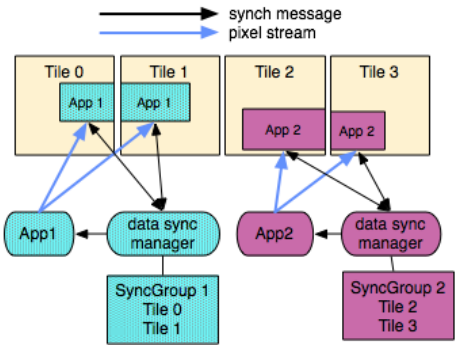

Figure 4. Example of four display nodes displaying two applications. There is a data sync manager and SyncGroup pair for each application.

Table 1. Number of messages that need to be exchanged in SAGE in a single round with its former synchronization algorithm to display application data on the tiled display.

	Number of Messages
Application Frame Updates	***M * N*** (Each application on a node sends a message to its sync master.) For M applications running on a display driven by N nodes, we have the worst case of ***M*N*** messages for frame updates.
Data Sync Messages	The worst case of ***M*N messages*** from data synch manager
Total messages per round	***(M*N) + (M*N) = 2*M*N***

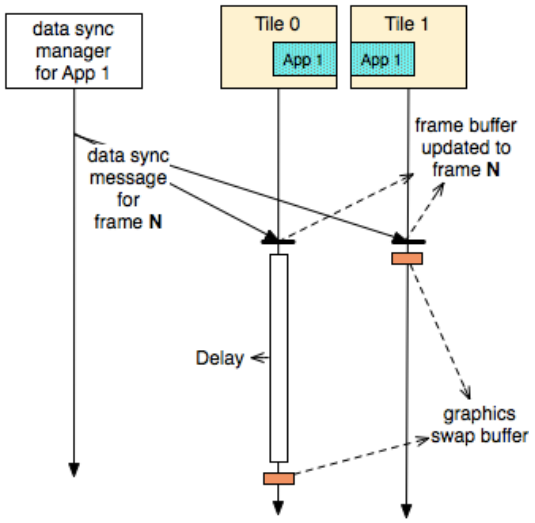

Figure 5. Effect of lacking swap buffer synchronization. The tile node 1 executed swap buffer right after updating to frame N, but the tile node 0 did not, because of its CPU load and scheduling. This uncertain delay may increase as the number of applications on the tile 0 increases. This incurs frame synchronization jitter.

Although this method was able to achieve the data synchronization of multiple applications [22], it requires excessive messages because it has a separate data synchronization manager for each application. Table 1 shows the worst-case message complexity when a tiled display consists of N display nodes displaying M applications. We also did not implement swap buffer synchronization explicitly. Since graphics swap buffer at each node can be performed as soon as data synchronization is finished, swap buffer synchronization can be achieved implicitly. Although this implicit swap buffer synchronization can be enough for supporting one application at a time [29], it may incur nontrivial synchronization jitter when it handles multiple applications because of the difference in CPU load and scheduling across display nodes. As the number of applications increases, the variance of the swap buffer completion time across all display nodes increases. Figure 5 illustrates implicit the swap buffer synchronization that may result in incoherent frame display.

5. Two New Algorithms

In our new approach, data synchronization is achieved by a single global synchronization master instead of a separate data synchronization manager for each application. We present two approaches in this paper—a two-phase algorithm and a one-phase algorithm.

In both algorithms, the global synchronization manager provides data synchronization. Whereas the two-phase algorithm achieves swap buffer synchronization with a network barrier after the data synchronization phase, the one-phase algorithm uses Network Time Protocol (NTP) synchronized clocks on each node. Consequently, the two-phase algorithm is more generally applicable to diverse high-resolution tiled-display systems, whereas the one-phase approach yields higher synchronization accuracy but has a limited range of target systems since it requires support for NTP and is more CPU-intensive.

5.1 Two-Phase Algorithm

The two-phase algorithm consists of two distinct phases: data synchronization for all applications and synchronization of the swap buffer events of all display nodes.

Figure 6 depicts the two-phase synchronization algorithm. In this case, we have a single global synchronization master (*SYNC MASTER*). Upon receiving a new frame at a display node, a corresponding application receiver on a display node sends a message with the new frame number and the node ID of the application receiver to the *SYNC MASTER*. The *SYNC MASTER* has an interval timer that runs at a periodic rate called **Sync Refresh Rate (SRR)**. The SRR must be a rate greater than the highest frame rate of all the applications in order to refresh all the applications at their desired rate. When the timer expires, the *SYNC MASTER* computes the highest common frame number for each application on all the nodes. After computing the highest common frame for each application, the *SYNC MASTER* sends a broadcast message to the NDM on each node. This message contains a list of the highest common frame number for each application. The NDM on each node uses this list in order to display the appropriate frame for each application.

Upon finishing the first phase, the NDM on each node enters the second phase in order to synchronize the swap buffer events. This synchronization is achieved by placing a networked barrier right after the frame buffer drawing and just before the frame buffer swap in the NDM. This enables swap buffer synchronization with each NDM displaying multiple applications on the tiled-display wall. Table 2 depicts the number of messages needed per round with the two-phase algorithm, where ***M*** is the number of applications and ***N*** is the number of nodes driving the display wall. The two-phase method uses ***(M-3) * N*** messages less than the former SAGE frame synchronization algorithm per round.

The key differences between the two-phase and the former SAGE frame synchronization algorithm are as follows:

1. The former version of SAGE has a data synchronization manager for each application, whereas the two-phase algorithm has a single data synchronization manager responsible for all the applications. This dramatically reduces the number of messages needed to reach data synchronization per round.
2. The former approach does not achieve swap buffer synchronization explicitly, whereas the two-phase algorithm uses a networked barrier to ensure swap buffer synchronization.

5.2 One-Phase Algorithm

In the one-phase approach, we achieve both data and swap buffer synchronization in a single phase. We avoid the second phase in the two-phase approach by synchronizing the clocks of the nodes driving the tiled display. The NTP, a common component in most major operating systems including Linux, helps synchronize the clocks on the cluster nodes. The data synchronization procedure is identical to the first phase of the two-phase approach.

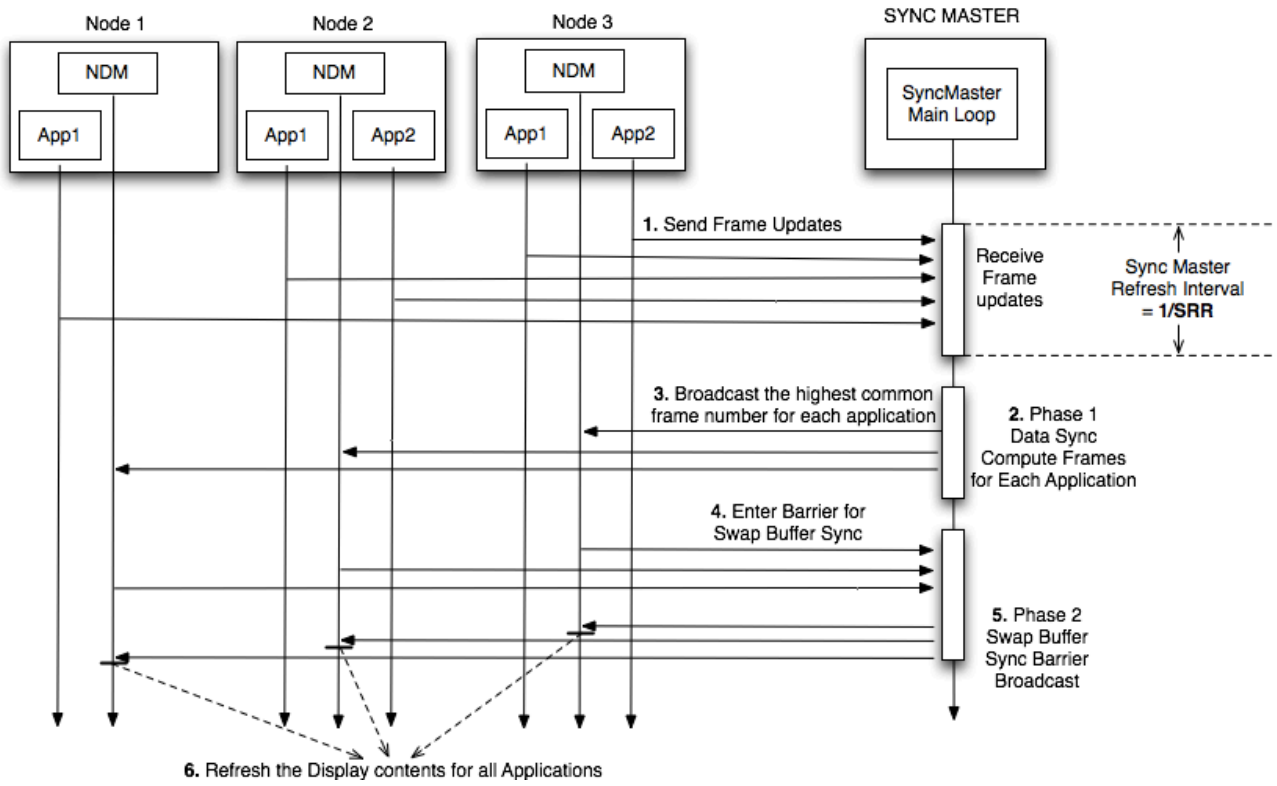

Figure 6. The two-phase synchronization algorithm to achieve seamless display of multiple applications on a tiled-display wall. The first phase ensures data synchronization, and the second phase ensures swap buffer synchronization using a networked barrier.

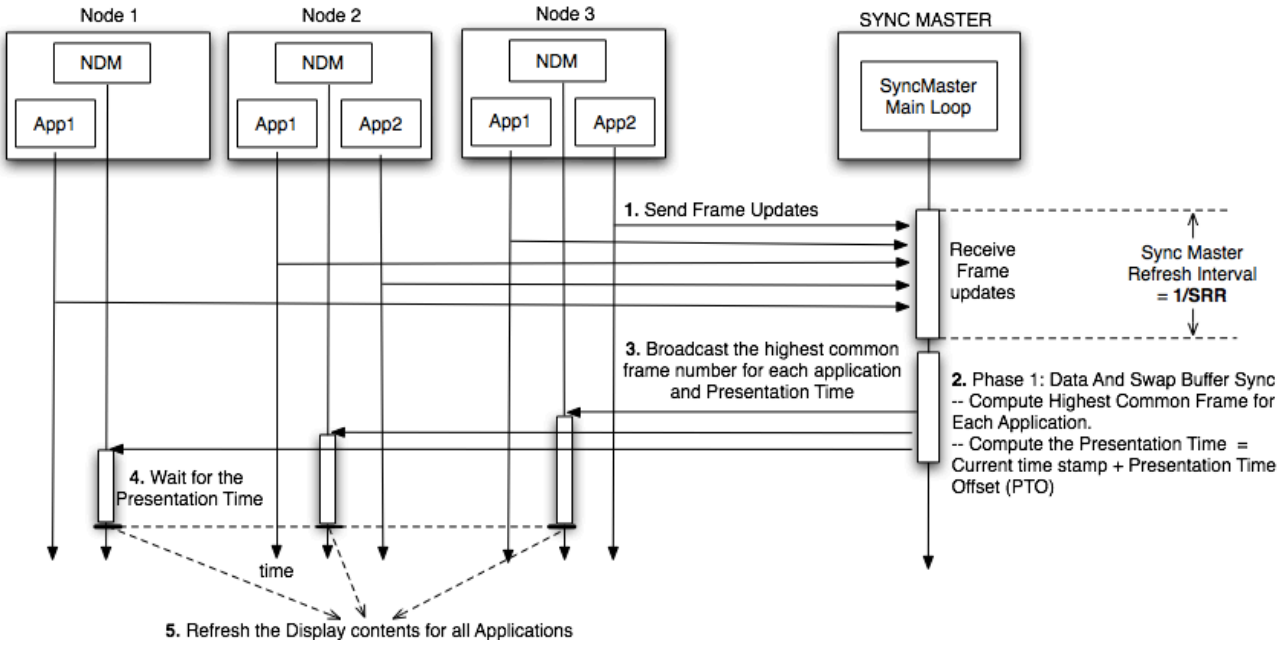

Figure 7. The one-phase synchronization algorithm to achieve seamless display of multiple applications on a tiled-display wall. The first phase ensures data synchronization, and the synchronized swap buffer is ensured by making each node wait until presentation time instead of using a centralized networked barrier at the SYNC MASTER.

A new term introduced in the algorithm is the **Presentation Time (PT)** that informs each NDM of the time when they should swap their buffer contents. After computing the highest common frame for each application, the *SYNC MASTER* computes the PT by adding a **Presentation Time Offset (PTO)** to the current time. The *SYNC MASTER* sends a broadcast message to each NDM. This message contains the PT and a list of the highest common frame number for each application. Each NDM waits till the PT and then displays the appropriate frame for each application according to the highest common frame number list. This procedure achieves both data and swap buffer synchronization.

The PTO depends on a number of factors, including the computational load on each node, the message delivery time, and the maximum frame rate of all applications. The PTO can be either fixed to a constant large value or computed dynamically by the *SYNC MASTER* based on periodic feedback from the various clients. In our prototype, we chose a fixed value empirically.

The trade-off of the one-phase algorithm against the two-phase algorithm is lower synchronization jitter in return for higher CPU usage caused by the implementation limit. However, because of the unpredictable nature of user interaction, network status, and computational load at each node, it is hard to determine the proper PTO for each frame (for each round) adaptively at the *SYNC MASTER*.

Table 2. Number of messages that need to be exchanged in SAGE in a single round with the two-phase algorithm to display application frames on the tiled display.

	Number of Messages
Application Frame Updates	***M * N*** (Each application on a node sends a message to the *SYNC MASTER*.) For M applications running on a display driven by N nodes, we have ***M*N*** messages for frame updates.
Phase 1: Data Sync	***N*** (The *SYNC MASTER* sends a message to an NDM on all the ***N*** nodes.)
Phase 2: Swap Buffer Sync	***2N***
2a. Barrier msg from each NDM to the Barrier Master	N
2b. Broadcast msg from the Master to all NDM's	N
Total Messages per round	***(M*N) + N + 2N = (M+3)* N***

Table 3. Number of messages that need to be exchanged in SAGE in a single round with the one-phase algorithm to display application data on the tiled display.

	Number of Messages
Application Frame Updates	***M * N*** (Each application on a node sends a message to the *SYNC MASTER*.) For M applications running on a display driven by N nodes, we have ***M*N*** messages for frame updates.
Phase 1: Data Sync	***N*** (The *SYNC MASTER* sends a message to an NDM on all the ***N*** nodes. The Presentation Time is embedded in the message.)
Total Messages per round	***(M*N) + N = (M+1) * N***

6. Experiments

In this section, we evaluate the efficacy of the two-phase and one-phase approach and compare them with the prior SAGE synchronization algorithm. We also evaluate how the two new algorithms scale with respect to the number of applications, number of nodes, and the frame rate of applications. In Figures 9 through 13, "old sage" refers to the prior version of SAGE that does not use the enhanced synchronization methods. In the case of the one-phase approach, the wait period till the PT is implemented by using high-resolution hardware counters.

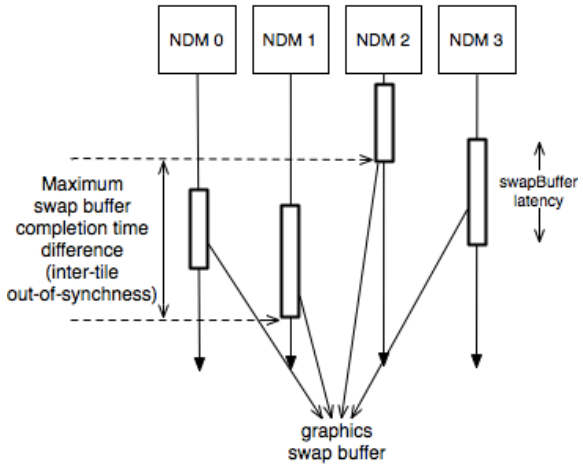

Figure 8. Method for computing the difference in the swap buffer completion time at each frame among four display nodes.

The testbed consists of a 28-node cluster driving an 11x5 tiled-display wall. The cluster nodes are each equipped with 64-bit dual processor 2.4 GHz AMD Opterons with 4 GB RAM, Nvidia Quadro 3000 Graphics Card in an AGP slot, and a dedicated 1 GigE network interface card. The cluster nodes run Linux kernel 2.6. The nodes are interconnected via a CISCO 3750 switch with 96 Gbps bisection bandwidth. A software-

based NTP server is run on the master node and a NTP client is run on each cluster node to synchronize the clocks. The NTP protocol has an accuracy of 100 microseconds. For the test application, we use several 1K (approximately 1000x1000 pixel) animations stored on a high-end dual-processor dual-core AMD Opteron node, which is equipped with 8 GB RAM and is connected to the 28-node cluster over a 10-gigabit network via a 10G Neterion NIC.

We use the difference in the swap buffer completion time among the various nodes as a metric to evaluate intertile frame synchronization. Figure 8 depicts the swap buffer completion time, which is defined as the time difference between the earliest and the latest swap buffer completion times among all nodes for a particular refresh cycle. A large difference indicates the tiles are out of synchronization. In the case of video playback, which is our main concern in this paper, the simultaneous frame transition across display nodes should occur in several milliseconds [15, 29]. Since the clocks are synchronized by using NTP, the swap buffer completion time is measured by time-stamping the swap buffer completion events on each node.

We show our experiments on the intertile swap buffer completion time differences for a single application in Section 6.1 and for multiple applications in Section 6.2. In Sections 6.3 and 6.4, we show the effect of the synchronization method on the frame rates of a single and multiple applications. The scalability with respect to the number of display nodes is shown in Section 6.5. In Section 6.6, we compare the average CPU usage of the NDM of the two-phase and the one-phase algorithms.

6.1 Intertile Swap Buffer Completion Time Difference (Out-of-Sync Time) for a Single Application

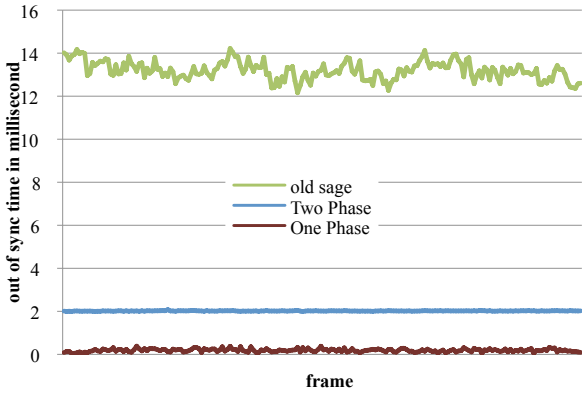

Figure 9. Comparison of the swap buffer completion time differences. The graph shows that the proposed synchronization algorithms achieve extremely low swap buffer variance in comparison to the prior frame synchronization method in SAGE, which in turn results in better intertile frame synchronization. The high variance in the prior method is mainly because it lacks swap buffer synchronization.

A single test application was run on the entire display at 30 frames per second. The *SYNC MASTER* Refresh Rate (SRR) was set to 60 Hz and the PTO for the one-phase method was configured to 9 ms. Figure 9 depicts the swap buffer completion time variance observed for 1,000 consecutive frames. As seen in the figure, the difference in the completion time of swap buffer among the various nodes in the prior approach is around 12–14 ms, which caused the viewers to visually perceive tearing in the application image on the display wall.

The two-phase algorithm achieves around 2 ms—a *6-fold* improvement over the prior approach. The one-phase algorithm achieves around 0.15 ms—a *10-fold* improvement over the two-phase and up to *90-fold* improvement over the prior approach. The relatively high variance in "old sage" is mainly because it lacks swap buffer synchronization (i.e., the second phase of the two-phase algorithm) in NDMs. This showed the need for swap buffer synchronization even for the case of single application. The swap buffer completion time variance in the two-phase algorithm was higher than that of the one-phase algorithm because of the barrier latency in the second phase of the two-phase algorithm. However, both the two-phase and one-phase algorithms achieve intertile frame synchronization and exhibit a visually seamless display across the wall.

6.2 Intertile Swap Buffer Completion Time Difference (Out-of-Sync Time) with Increasing Number of Applications

In our next experiment, we increased the number of applications streaming to the display wall and evaluated the impact of the synchronization mechanisms on the intertile swap buffer completion time. The SRR was set to 60 Hz, and the PTO for the one-phase algorithm was fixed at 9 ms. The test application was a 1K animation remotely streamed at 30 fps. We arranged multiple instances of this animation window so that the entire display wall was covered with nonoverlapping windows.

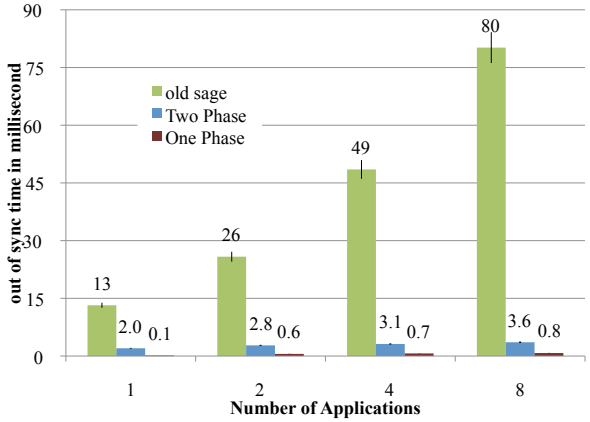

Figure 10. Average of maximum swap buffer completion time difference of multiple applications. The graph shows the algorithm scales as the number of applications is increased.

The results shown in Figure 10 indicated that the prior approach, per application synchronization, failed to sustain acceptable synchronization jitter with increasing number of applications, whereas both the two-phase and the one-phase algorithms ensured tight frame synchronization despite the increasing number of applications. This was due to the single global synchronization master for all applications. Though minor increments of the swap buffer completion time difference were

incurred by the increased system overhead on the display nodes as we increase the number of applications, this result still satisfied very tight frame synchronization tolerance.

6.3 Evaluation on an Application's Frame Rate

Three-dimensional stereoscopic tiled-display walls including the Varrier [31] and StarCAVE [14] are used for interactive and immersive stereo visualizations. Such applications require support for a high frame rate, up to 120 fps to achieve interactivity [11]. A frame synchronization scheme for these applications must be able to achieve tight synchronization with minimal impact on the application's frame rate. Though the current prototypes of these systems are designed for a single application, we expect the future extension for multiple applications. Thus, we evaluated the performance of the synchronization algorithms as we scaled the frame rate of a 1K animation displayed across the entire 28-node tiled display wall. In each case, the *SYNC MASTER* Refresh Interval was set 10 Hz higher than the application rate, and the PTO for the one-phase algorithm was set to 6 ms. Figure 11 depicts the effect of the synchronization algorithms on the frame rate of the animation as we increased the target frame rate of the animation from 30 fps to 120 fps. From the figure, we observed that the two new algorithms were able to sustain the target frame rate with minimal deviation. The prior algorithm sustained the target frame rate till 60 fps but failed to sustain it at 120 fps.

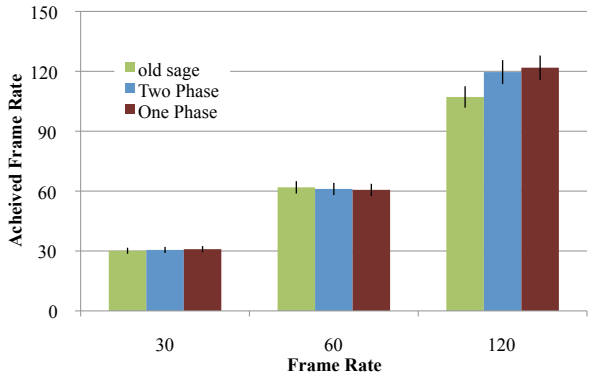

Figure 11. Comparison of the synchronization algorithms on the application frame rate as we scale the frame rate of a single application. Both the two-phase and the one-phase algorithms scale with an application's frame rate.

6.4 Aggregate Frame Rate with Increasing Number of Applications

In addition to being able to sustain high frame rates, a good synchronization mechanism should be able to sustain the frame rates as more applications are launched on a tiled-display wall. To test this capability, we increased the number of applications streaming to the display wall and evaluated the impact of the synchronization mechanisms on the aggregate achievable frame rate. Again, each application streamed a 1K animation at 30 fps. From Figure 12, we observed that the synchronization mechanisms of the previous version SAGE ("old sage") was able to sustain the frame rate only up to four applications and showed a *25%* drop for eight applications. As indicated in Section 4, the reason is that the per-application-based data synchronization mechanism requires excessive synchronization messaging. In contrast, both the two-phase and the one-phase algorithms scale as the number of applications increases.

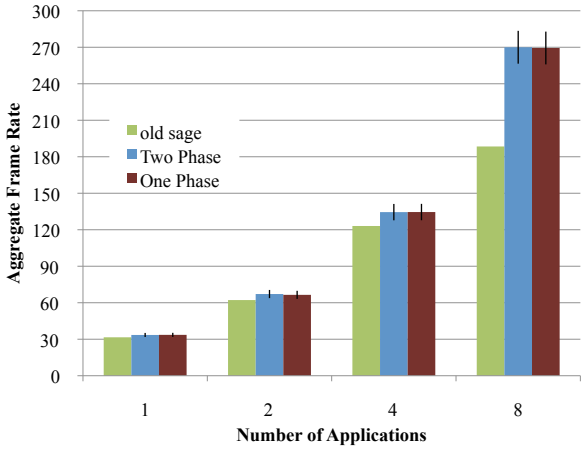

Figure 12. Comparison of the synchronization algorithms on the aggregate application frame rate as we increased the number of applications each runs at 30 frames per second. The two-phase and one-phase methods scaled with the number of applications, whereas the old SAGE synchronization algorithm showed its limited scalability.

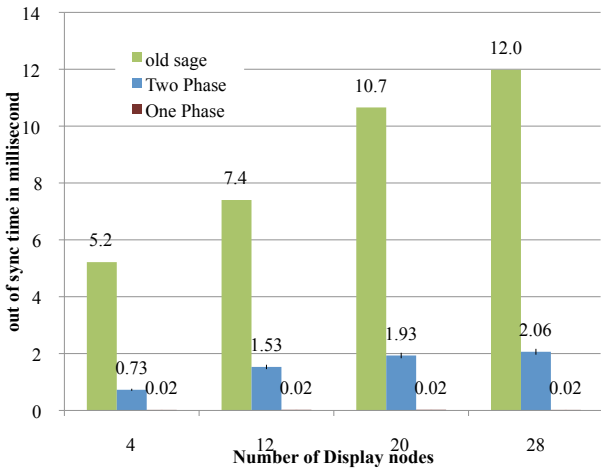

Figure 13. Scalability of the algorithms as the number of display nodes increases. The swap buffer completion time difference in the two-phase algorithm slightly increases because of the swap buffer synchronization phase using a networked barrier, whereas the effect of the display node increase is minimal in the one-phase algorithm.

6.5 Scalability Analysis with Increasing Number of Nodes

Another key requirement of a good synchronization mechanism is the ability to scale with the number of display nodes. Figure 13 depicts the intertile swap buffer completion time differences as we increased the number of display nodes and the associated tiled-display size. In the case of the two-phase algorithm, we

observed minor increments in the intertile swap buffer completion time difference as we scaled the number of nodes.

These werre primarily an effect of the networked swap buffer barrier employed in the second phase: as the number of nodes increased, the time to broadcast all the messages increased and incurred additional lag. However, the 2 ms difference was too short for a user to notice any asynchrony between tiles. The one-phase algorithm achieved the tightest synchronization (~0.02 milliseconds), primarily because of the synchronized clocks and use of high-resolution hardware timers for displaying the frame as close as possible to the presentation time. Again, the two new algorithms showed much more improved scalability over the prior approach in this test.

6.6 Comparison of Average CPU Utilizations of the Two-Phase and the One-Phase Synchronization Algorithms

Resource utilization of the algorithms is a key concern. In the one-phase algorithm, each display node enters into busy waiting loop, waits until the presentation time, and then executes graphics swap buffer. This approach achieves a synchronized swap buffer across display nodes and eliminates 2***N*** network messages, where ***N*** is the size of the cluster (total ***N*** number of display nodes). Thus, the one-phase algorithm reduces message complexity at a cost of more CPU cycles than the two-phase algorithm because of the wait-loop. In this experiment, for example, we evaluated CPU usages of the NDM when a 1K animation was displayed on the display wall. The one-phase algorithm used 20% of CPU time, whereas the two-phase algorithm used only 4%. Therefore, when a tiled display is driven by a cluster of thin-client computers that do not have enough CPU resources for the wait-loop, or when computationally intensive processes need to run on the tiled display cluster, the one-phase algorithm should be avoided.

7. Discussion

The PTO of the one-phase algorithm has been set manually in the current implementation. The PTO has to be carefully chosen. If the PTO is set too long, then extra busy waiting can occur at a display node resulting in wasted CPU cycles. A large PTO also can affect the application's frame rate because it increases the frame transition interval. If the PTO is set too small, a synchronization message can arrive at a display node behind the PT, or a node can complete the frame buffer drawing behind PT. The swap buffer synchronization fails in these cases. An example of a small PTO is shown in Figure 14. The NDM3's graphics swap buffer is not synchronized because the synchronization message arrives at the NDM3 behind PT. These problems can be resolved by adaptively determining the PTO for each frame at the *SYNC MASTER*. However, to make PTO adaptive is challenging because of the uncertainty in the system load on the cluster.

The two algorithms we have presented assume reliable delivery of frames, namely, that the image frame data not be lost or dropped in any of the display nodes. We are investigating synchronization techniques to handle unreliable streaming of frames.

8. Conclusion

We presented the two-phase and the one-phase algorithms to achieve a seamless display of multiple applications on a high-resolution tiled display wall driven by a cluster of computers. Whether one would choose to utilize the one-phase versus two-phase algorithm depends on the desired synchronization accuracy and the availability of system resources. The two-phase algorithm has the advantage that it is more generic and can therefore be easily applied to most high–resolution tiled display systems including the ones driven by networked thin clients. It provides high synchronization accuracy generally acceptable for interactive high-resolution visualization. The one-phase algorithm provides superior synchronization characteristics because of its low degree of messaging complexity. However, the one-phase algorithm requires support for NTP and sophisticated display thread scheduling (which is currently implemented via a busy-wait loop). Hence, if very tight synchronization is required and if NTP and additional unused processing cores are available, one should use the one-phase algorithm. Both methods, however, will scale with respect to the number of applications, the frame rates of the applications, and the number of cluster nodes.

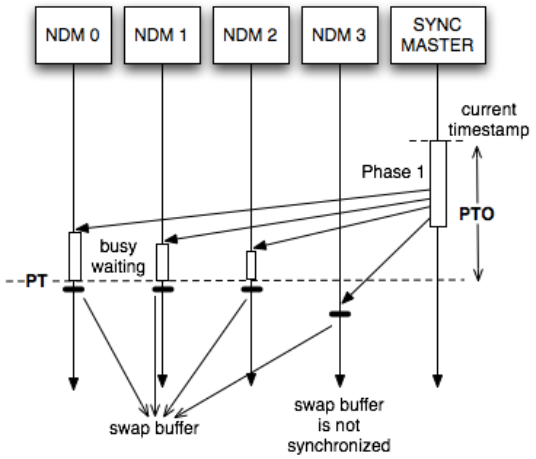

Figure 14. Effect of small PTO. The Presentation Time (PT) calculated from the Presentation Time Offset (PTO) is passed when the synchronization message is received at NDM3. Graphics swap buffer at NDM3 cannot be synchronized with other display nodes.

Acknowledgments

The publication was based on work supported in part by Sharp Laboratories of America, the King Abdullah University of Science and Technology (KAUST) (Award US-2008-107/SA-C0064), the National Science Foundation (Award OCI 0943559), and the Office of Advanced Scientific Computing Research, Office of Science U.S. Department of Energy, under Contract No. DE-AC02-06CH11357.

We would like to thank Lance Long, Alan Verlo, Patrick Hallihan, Andrew Johnson, and Maxine Brown at the Electronic Visualization Laboratory, and Gail Pieper at the Argonne National Laboratory for reviewing the paper.

References

[1] Cross-Platform Cluster Graphics Library (CGLX), http://vis.ucsd.edu/~cglx/

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

[3] Message Passing Interface, http://www.mpi-forum.org/

[4] Allard, J., Gouranton, V., Lamarque, G., Melin, E., and Raffin, B. "SoftGenLock: active stereo and genlock for PC cluster," *Proceedings of the workshop on Virtual environments 2003*, Zurich, Switzerland, pp.255-260, 2003.

[5] Allard, J., Gouranton, V., Lecointre, L., Melin, E., and Raffin, B. "Net Juggler: running VR Juggler with multiple displays on a commodity component cluster," *Virtual Reality, 2002. Proceedings. IEEE*, pp.273-274, 2002.

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

[7] Bues, M., Blach, R., Stegmaier, S., Häfner, U., Hoffmann, H., and Haselberger, F. "Towards a Scalable High Performance Application Platform for Immersive Virtual Environements," *Proceedings of Immersive Projection Technology and Virtual Environments*, Stuttgart, pp.165-174, 2001.

[8] Chen, H., Chen, Y., Finkelstein, A., Funkhouser, T., Li, K., Liu, Z., Samanta, R., and Wallace, G. "Data distribution strategies for high-resolution displays," *Computers & Graphics*, vol.25, no.5, pp.811-818, 2001.

[9] Chen, H., Clark, D. W., Liu, Z., Wallace, G., Li, K., and Chen, Y. "Software environments for cluster-based display systems," *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, Washington, DC, pp.202-210, 2001.

[10] Choi, J.-D., Byun, K.-J., Jang, B.-T., and Hwang, C.-J. "A synchronization method for real time surround display using clustered systems," *Proceedings of the tenth ACM international conference on Multimedia*, Juan-les-Pins, France, pp.259-262, 2002.

[11] Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. "Surround-screen projection-based virtual reality: the design and implementation of the CAVE," *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, 1993.

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

[13] De Winter, D., Simoens, P., L., D., F., D. T., J., M., B., D., and P., D. "A hybrid thin-client protocol for multimedia streaming and interactive gaming applications," *Proceedings of the 2006 International Workshop on Network and Operating System Support for Digital Audio and Video*, Newport, Rhode Island, 2006.

[14] DeFanti, T. A., Dawe, G., Sandin, D. J., Schulze, J. P., Otto, P., Girado, J., Kuester, F., Smarr, L., and Rao, R. "The StarCAVE, a third-generation CAVE and virtual reality OptIPortal," *Future Generation Computer Systems*, vol.25, no.2, pp.169-178, 2009.

[15] Deshpande, S. "A method for synchronization mismatch perception evaluation for large ultra high resolution tiled displays," *Quality of Multimedia Experience, 2009. QoMEx 2009. International Workshop on*, pp.238-243, 2009.

[16] Eilemann, S., Makhinya, M., and Pajarola, R. "Equalizer: A Scalable Parallel Rendering Framework," *Visualization and Computer Graphics, IEEE Transactions on*, vol.15, no.3, pp.436-452, 2009.

[17] Huang, J.-y., Wang, K. M., and Hsu, K.-W. "The frame synchronization mechanism for the multi-rendering surrounding display environment," *Displays*, vol.25, no.2-3, pp.89-98, 2004.

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

[19] Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., and Hanrahan, P. "WireGL: a scalable graphics system for clusters," *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 2001.

[20] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. "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.

[21] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. "Chromium: a stream-processing framework for interactive rendering on clusters," *ACM SIGGRAPH ASIA 2008 courses*, Singapore, 2008.

[22] 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," *Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference*, pp.24-24, 2006.

[23] Lamport, L. "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol.21, no.7, pp.558-565, 1978.

[24] Nirnimesh, Harish, P., and Narayanan, P. J. "Garuda: A Scalable Tiled Display Wall Using Commodity PCs," *Visualization and Computer Graphics, IEEE Transactions on*, vol.13, no.5, pp.864-877, 2007.

[25] Genlock, http://www.nvidia.com/object/IO_10793.html

[26] Nvidia Quadro G-Sync, http://www.nvidia.com/page/quadrofx_gsync.html

[27] Nvidia, Quadro FX 3000G Solutions for Advanced Visualization. Technical Report. NVIDIA Corporation, 2003.

[28] Raffin, B., Soares, L., Tao, N., Ball, R., Schmidt, G. S., Livingston, M. A., Staadt, O. G., and May, R. "PC Clusters for Virtual Reality," *Virtual Reality Conference*, pp.215-222, 2006.

[29] Rrustemi, A. "Computing Surface - a platform for scalable interactive displays," Doctoral Thesis, University of Cambridge, 2008.

[30] Samanta, R., Zheng, J., Funkhouser, T., Li, K., and Singh, J. P. "Load balancing for multi-projector rendering systems," *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, Los Angeles, California, United States, 1999.

[31] Sandin, D. J., Margolis, T., Ge, J., Girado, J., Peterka, T., and DeFanti, T. A. "The Varrier™ autostereoscopic virtual reality display," *ACM SIGGRAPH 2005 Papers*, Los Angeles, California, 2005.

[32] Schaeffer, B. "Networking and Management Frameworks for Cluster-Based Graphics," *Virtual Environment on a PC Cluster Workshop*, Protvino, Russia, 2002.

[33] Schikore, D. R., Fischer, R. A., Frank, R., Gaunt, R., Hobson, J., and Whitlock, B. "High-resolution multiprojector display walls," *Computer Graphics and Applications, IEEE*, vol.20, no.4, pp.38-44, 2000.

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

[35] Smarr, L., Brown, M., and de Laat, C. "Special section: OptIPlanet -- The OptIPuter global collaboratory," *Future Generation Computer Systems*, vol.25, no.2, pp.109-113, 2009.

[36] Tan, D. S., Gergle, D., Scupelli, P., and Pausch, R. "Physically large displays improve performance on spatial tasks," *ACM Transactions on Computer-Human Interaction*, vol.13, no.1, pp.71-99, 2006.

[37] Wallace, G., Anshus, O. J., Bi, P., Chen, H., Chen, Y., Clark, D., Cook, P., Finkelstein, A., Funkhouser, T., Anoop, G., Hibbs, M., Li, K., Liu, Z., Rudrajit, S., Rahul, S., and Troyanskaya, O. "Tools and applications for large-scale display walls," *Computer Graphics and Applications, IEEE*, vol.25, no.4, pp.24-33, 2005.