Experiences with Scalability of Display Walls

Han Chen, Grant Wallace, Anoop Gupta[§], Kai Li, Tom Funkhouser, Perry Cook
Computer Science Department, Princeton University
{chenhan, gwallace, li, funk, prc}@cs.princeton.edu, [§]anoopg@microsoft.com

Abstract

Immersive virtual environments require large-scale high resolution displays to match the human visual acuity. This drives the need to study scalability issues related to display wall systems. This paper reports our experiences in scaling up the display wall system at Princeton University from 8 to 24 projectors. We discuss scalability techniques for automatic projector calibration, cluster management, data distribution for still images, motion video, and multi-channel audio.

1. Introduction

An immersive virtual environment calls for scalable large-scale display surface with high resolution to match the large field of view and high acuity of the human visual system. In recent years, there have been many multi-projector display wall projects. These include the Power Wall at University of Minnesota [29], the Office of the Future project at University of North Carolina [22], the Interactive Workspaces Project at Stanford University [14], and display wall projects in various national laboratories such as Argonne [12], Lawrence Livermore [24], Sandia [10], and National Center for Supercomputing Applications [20], etc. However none of these systems address scalability issues in the following areas: automatic projector calibration, cluster management, still image viewer, motion video player, and multi-channel audio system. In this paper, we present our experiences in these aspects of scaling our display wall system.

In March 1998, Princeton built its first display wall with an $18' \times 8'$ rear projection screen and 8 Proxima LCD commodity projectors, see Figure 1 left. This system had a resolution of 4096×1536 pixels and was driven by a network of 8 Pentium II PCs running Windows NT [17].

In November 2000, we scaled the display up with 24 Compaq MP1800 DLP projectors and a network of 24 Pentium III PCs running Windows 2000, see Figure 1 right. There are also computers for various inputs, including mouse, camera, HDTV, etc. as shown in Figure 2. The resolution of our new system is 6144×3072.

Figure 1: Projector Setups of the Princeton Display Wall.
Left: First Generation System with 8 Projectors
Right: Second Generation System with 24 Projectors

In our scaling efforts, we found that the techniques that were sufficient for 8 projectors became excessively time consuming and labor intensive for 24 projectors. Projector geometric calibration was the first stumbling block we encountered. It took six hours to do an initial manual alignment of the projectors to a one to two pixel accuracy. Afterwards time intensive weekly adjustments were still needed. Our automatic alignment system designed for the first generation display wall was also too time consuming to be practical. Next we found our previously developed methods of displaying content stretched to their limit - it took ten seconds to show an high quality 18-million-pixel image (5MB JPEG). And throughout there was the need to maintain the machines, update drivers, manipulate dialog boxes and present an intuitive and simple interface to the daily user.

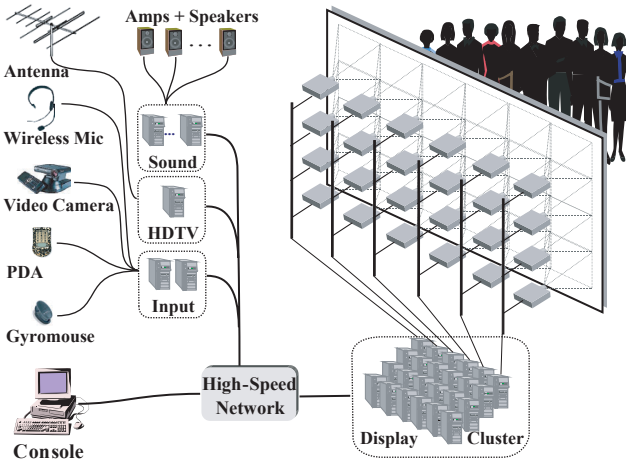

Figure 2: Architecture of Princeton Display Wall

The rest of the paper is organized as follows. The first two sections deal with hardware related issues: first we discuss scalable automatic projector calibration methods, and then scalable control and administration interfaces for display walls. The next three sections deal with scalable content distribution. These include methods for distributing still images, motion videos and multi-channel audios. Finally we conclude with a summary of what we learned in the process of scaling our display wall and directions for future work.

2. Multi-Projector Calibration

Projector calibration is the key to turning individual projectors into one cohesive display. The most obvious and time consuming part of it is geometric alignment. Vibration, heat expansion, and projector power-on-and-off can cause several pixels of drift per week even with clamped projectors. With a 6-degree-of-freedom projector mount, we can manually align the projectors. This process is laborious and yet does not yield precise alignments (1 or 2 pixels average discontinuity).

Several automatic projector geometric alignment systems [23][27] have been described; for instance, Chen et al. developed an automatic alignment system [6] for our first generation display wall. It uses a pan-tilt camera mounted in front of the screen to acquire point and line correspondences between adjacent projectors. This information is then used to form a global optimization problem and *Simulated Annealing* (SA) is used to solve for the projective transformation matrix for each projector. The transformation matrices are used to pre-warp images so that the projected results look seamless. This approach achieves better results than manual alignment and it takes about one hour to complete for the 8-projector system.

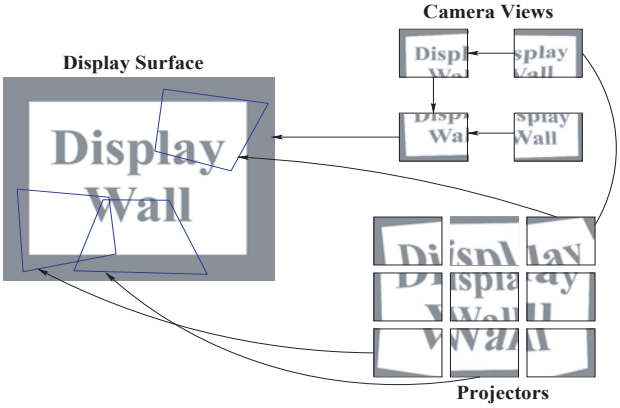

Figure 3: Homographies between Various Elements: Screen, Camera Views, and Projectors

When we scaled our wall to 24 projectors, we found that neither the manual nor the simulated annealing method was practical; it took six hours to manually align the projectors or about three hours for the automatic alignment system using simulated annealing. Thus, we developed a new method called *Homography Tree Optimization* (HTO) [5] based on the *Smarter Presentation* system [26]. It extends the idea of *PixelFlex* [31] to scales beyond the scope of a single camera. In our system, we use a pan-tilt camera to detect feature points on each of the projector screen by displaying geometric patterns. The camera's field of view covers a 2x2 subset of the projectors. We allow for a 50% overlap between adjacent camera views and use 15 camera views to cover our 4x6 projector array. Based on the common features points between adjacent camera views, a homography can be calculated. A tree based homography optimization algorithm is then used to fine tune the homographies to make them consistent. This is conceptually equivalent to stitching the 15 camera images to form a 2D mosaic; however, our algorithm is able to achieve much better accuracy. The feature points in this virtual camera frame are then warped to make them rectilinear. Finally, a projective transformation matrix can be extracted for each projector from the locations of its feature points. Figure 3 shows the homographies between the various elements in the system.

Table 1: Comparison of Different Alignment Methods

Method	2×4 Display Wall		4×6 Display Wall	
	Time(min)	Error(pixel)	Time(min)	Error(pixel)
Manual	120	1-2	360	2-3
SA[1]	69[2]	1.4[2]	181[3]	2[3]
HTO	4[4]	0.8[4]	12	0.8

1. SA may produce better results with signficantly longer run time.
2. Data collection time: 33 min, SA runtime: 36 min with 20K steps [6].
3. Data collection time: 90 min, SA runtime: 91 min with 50K steps.
4. Results were obtained by using a 2x4 subset of the new display wall.

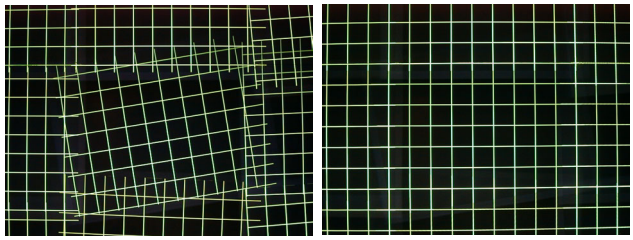

Figure 4: Grid Patterns Shown on Unaligned (left) and Aligned (right) Display Wall

The new system can calibrate our 24-projector display wall in 12 minutes, while achieving satisfactory alignments (0.8 pixel average discontinuity), see Figure 4. We also evaluated the new system on different size display walls, and found that the data collection and computation

time increases linearly with display size, while maintaining a constant average local discontinuity. Table 1 shows a comparison of different alignment methods on our two generations of display wall. It is clear that our new method is fast, accurate and scalable.

3. Display Wall Management

Using a display wall incorporates the management of multiple resources including computers, projectors, software and user input. Coordinating the use of all these resources can be confusing to an inexperienced user, and time consuming to an administrator unless adequate management facilities exist.

The two main properties we wanted in a management facility were ease of use, and speed of common operations. Ease of use is important for inexperienced users. We have about 20 new students every semester who use the display wall regularly as part of their class work. The time consumed in common operations is important to administrators. Upgrading a device driver on a Windows computer involves interacting with several dialog boxes and hitting many OK buttons.

When the number of resources is relatively small, the difficulties are less acute. When we had an 8 node display wall running Windows NT, we used a remote control to control the projectors, a *Keyboard-Video-Mouse* (KVM) switch to access the computers, and a set of command line scripts to start and stop processes. When we upgraded to 24 projectors and 24 Windows 2000 workstations, the old systems became more difficult to wield. Everyone becomes tired quickly of trying to turn on 24 projectors with a remote, installing a device driver 24 times and typing out long parameter lists on command line scripts.

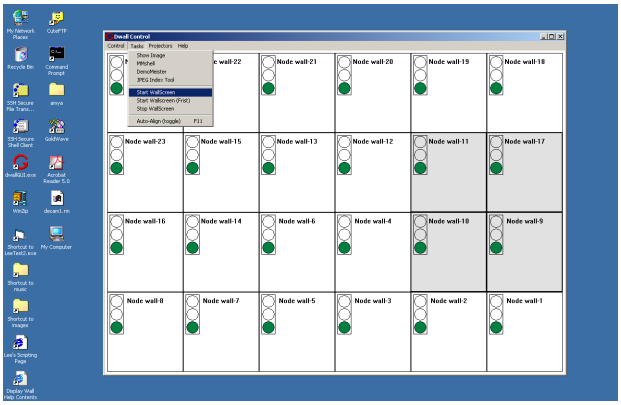

Figure 5: Display Wall Control GUI

Our solution to these issues was to incorporate all control functionality in one central place. We developed a *Graphical User Interface* (GUI) called DwallGUI to do this, see Figure 5. The DwallGUI brings together the control of three main components: projectors, computers and software; with ease of use as a goal.

The first component is the projectors. Our projectors do not provide an interface for computer control because they are portable units (but they have other good characteristics such as small, inexpensive and lightweight). They are also intended to be unplugged after every use. To get around these negatives we built an analog multiplexer which can be controlled from the parallel port of a computer. We connected a computer controlled IR sender to the multiplexer and strung IR emitters to each projector. This gives us complete control of the projectors individually or in groups. We added to this a set of Ethernet based power controllers so that we could remotely toggle the power to projectors.

The next component of DwallGUI is cluster management, which typically involves process control, monitoring and rebooting computers [3][25]. Because in a display cluster each computer is connected to a projector, we can avoid using KVM switches by sending keyboard and mouse commands in the DwallGUI. It is more powerful than using KVM switches in that we can see all displays and work on all computers or any subset simultaneously. This makes installing a Windows device driver on the cluster as easy as installing it on one computer.

This cluster management functionality is implemented on the display nodes through a Windows Service we developed called DwallRunner. This is a server that runs on each display node and is part of a multicast group. All control messages are multicast to the display cluster and selectively processed or ignored. DwallRunner is controlled by the DwallGUI and incorporates functionality for process control, monitoring, rebooting, and sending mouse/keyboard events.

Finally, the DwallGUI allows easy access to some of the most frequently used software on the cluster. This is especially important to new and casual users. They can select "show image" from the menu and browse for the image to show on the wall. They can run multimedia presentations, demos and other tools just as easily. In our experience, having a centralized control GUI has increased the user base of the display wall system.

4. Image Viewer

One of the most commonly used functionalities of a large scale display wall is to show high resolution images on it. Regardless of the actual system setup, in order to display an image on a cluster based display wall, one must first distribute the image data from one location, (e.g. file server, satellite receiver, etc.) to a cluster of computers, each of which will in turn decode/decompress the image

and display a proper portion of the image on its attached projector. The image might also be pre-warped with a projective transformation to create a seamless image across the display surface.

Because of the sheer number of pixels on a large scale display wall, e.g. 18 million pixels on the Princeton setup, storing or transferring images in uncompressed format is not practical. We use JPEG as our compression method. With JPEG compression, a typical image can be reduced to less than 10% of its original size even with the highest quality setting.

In a naïve implementation of image viewer, such as those using texture mapping with WireGL [7][13], a client program reads an image file from disk upon user request, decodes it in local memory, and then sends the decoded pixels to the display servers. This is easy to implement but doesn't scale well: the client becomes a bottleneck in terms of both computation and network bandwidth when the display wall size increases. For example, just sending 18 million pixels in RGB color (54MB of data) over a fully utilized fast Ethernet will take over five seconds.

In contrast to this client centric system, a distributed system ameliorates these problems by deferring the decoding process to the server side and transferring only the compressed images over the network. In such a system, the client simply sends a show-this-image command to decode/display servers, which then read the image file from file server, decode and show it. To avoid decoding the whole image while only a small portion of it is needed as in the case of the decode/display server, a technique of *Region-of-Interest* (ROI) decoding can be used to save computation requirement, such as that provided in the *Intel JPEG Library* (IJL) [15].

In our first generation of the display wall system, we used the aforementioned system and found the results to be satisfactory. However, as we scaled the system up to 24 projectors and PCs, we found that this technique became inadequate. The file server becomes a bottleneck because it has to send a copy of the whole image file to each decode/display server. IJL's ROI decoding routine has to parse the whole JPEG file even though it only needs to decode a portion of it. Because the *Minimum Coded Units* (MCU) in a JPEG image are dependent on the previous ones [21], one cannot decode an MCU without first decoding the immediately preceding one, see Figure 6 left.

To overcome this limit, we augmented the JPEG format with an index file, which saves the MCUs' DC predictors and position in file at regular intervals, thus breaking the dependency chain. We created a custom decoder that can recognize and utilize this index. When a decoder needs to skip a number of blocks, it can consult the index file to find out where the next closest indexed block starts in the bit stream, and what the DC coefficient predictors are, thus completely avoiding touching unused bits, as illustrated in Figure 6 right. We choose the indexing interval to match the operating system block caching size for network mounted file system, for example, 4KB is used for Windows 2000. Five short integers (10 bytes) are needed for every indexed block; therefore, the index file is only about 1/400 of the original JPEG file size.

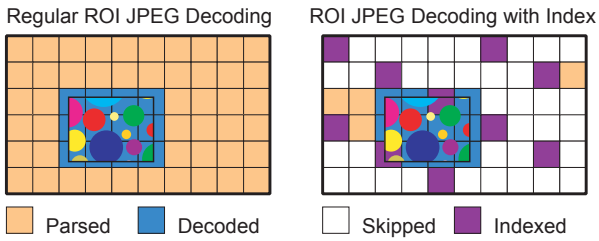

Figure 6: ROI JPEG Decoding with and without Index

We developed a multimedia presentation system based on the image viewer. It preloads all images before starting the sequence. We measured the load time of several presentations with and without the index to evaluate its performance, see Table 2. The speed-up is much less than the ideal number of 24, because the load time also includes JPEG decoding time and the block caching of operating system increases the amount of data transfer, as illustrated in the right side of Figure 6.

Table 2: Load Time of Multimedia Presentations

No.	# Images (Size)	w/o Index	w/ Index	Speed Up
1	294 (167MB)	358	155	2.3
2	113 (110MB)	248	90	2.8
3	73 (77.4MB)	165	61	2.7

5. Parallel MPEG-2 Decoder

Video is a powerful way of visualizing complex data. It is highly desirable to have the capability to show real time video at or close to the native resolution of the display wall. Because of the vast amount the data involved, storing and transferring video data uncompressed is not feasible. MPEG-2 is one of the most widely used video compression standards [8]. There are quick and naïve ways to bring MPEG-2 video to a cluster based display wall. One method is to decode a video stream in one node and send the decoded pixels to the display servers. Clearly, the performance of the system is limited to the computation power and network bandwidth of the decoding node. It will work, but only for video with relatively low resolution. Another method is to decode a video stream, split it into several sub-streams, re-encode them, distribute the sub-streams to the decode/display servers' local storage, and then decode and play these sub-streams in synchroni-

zation. This method works for higher resolution videos but it requires tremendous amount of offline computation, which can not be done in real-time. Also, the re-encoding process introduces additional quantization error and limits the ranges of motion vectors, thus reducing the video quality. There are also parallel MPEG-2 video decoders [1][2] for tightly coupled SMP machines. They don't scale on a cluster. To address these problems, we designed a cluster-based parallel MPEG-2 decoder system [4].

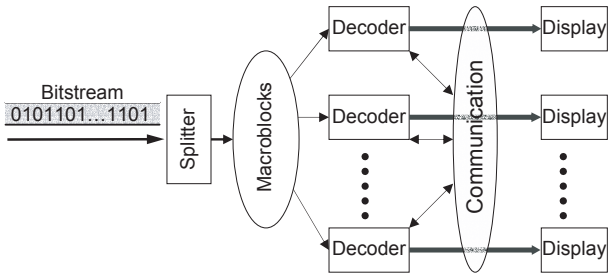

Figure 7: Macroblock-Level Parallel MPEG-2 Decoder

In the 8-node display wall system, we used a macroblock level parallel MPEG-2 decoder. As shown in Figure 7, there are three major components working together to decode a video stream: a splitter divides the input stream into macroblocks and sends them to the appropriate decoders. The decoders need to communicate with each other when the motion vector of a macroblocks references data not present in the local node. Finally, the decoded pixels are sent to the displays and might be pre-warped to correct for perspective distortion. In this system, the communication requirement is very low - the splitter sends out compressed bit streams; the decoders send and receive a small amount of pixel data, which are distributed in nature; no pixels need to be reshuffled before being displayed. The system can play 1080i HDTV at more than 30 frames per second.

When we scaled the display wall system up to 24 nodes and tried to play higher resolution video, e.g. near-IMAX quality (3840×2800), we noticed that the splitter became a bottleneck. This is because splitting an MPEG-2 video stream at macroblock level requires parsing the entire bit stream. A highly optimized MPEG-2 parser can only process about 40Mbps with our hardware setup, whereas the video itself is about 120Mbps.

To achieve scalable high resolution decoding, we introduced a hierarchical decoding system, see Figure 8. It consists of two levels of splitters and a set of decode/display servers. A root splitter (P-Splitter) reads in an input bit stream, scans it to find out where a picture starts and ends, copies the picture data to an output buffer, and then sends it to one of the k second-level splitters (M-Splitters) in a round-robin fashion to balance the workload. The second-level splitter parses the picture into macroblocks, and sorts them into one or more output buffers to make $m \times n$ sub-pictures, which are not necessarily MPEG-2 conforming. Because there is no inter-picture dependency in splitting a picture at macroblock level, each second-level splitter can process the picture it receives independently. The splitter then sends the sub-picture data to the corresponding decoding servers. The decode/display servers work in the same way as in the original system. We call this decoding method *1-k-(m,n)* system. The original macroblock level parallel decoder can be viewed as a special case of this hierarchical system where *k=1*.

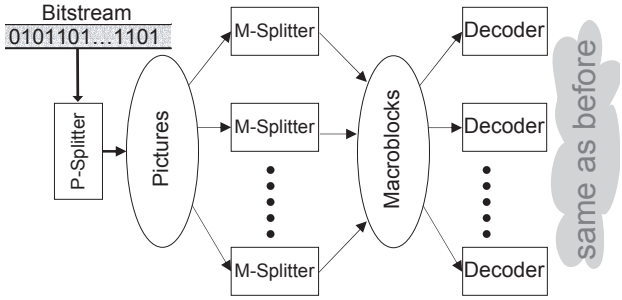

Figure 8: Hierarchical Parallel MPEG-2 Decoder

We used MPEG-2 video streams with resolution ranging from 720×480 to 3840×2800 to test the performance and scalability of the system. Figure 9 shows the frame rate of one-level and two-level decoders playing DVD and HDTV contents with varying size of display wall. We notice that the performance of a one-level system flattens out after a certain screen size indicating that the splitter becomes a bottleneck. However, a two-level system is able to scale. Figure 10 shows the number of pixels decoded per second versus the number of nodes ($1 + k + m \times n$) in a two-level decoder. Video size is chosen to match the screen size, where the highest resolution is 3840×2800 shown on a 4×4 subset of the screen. It is clear that a two-level decoder achieves near linear acceleration.

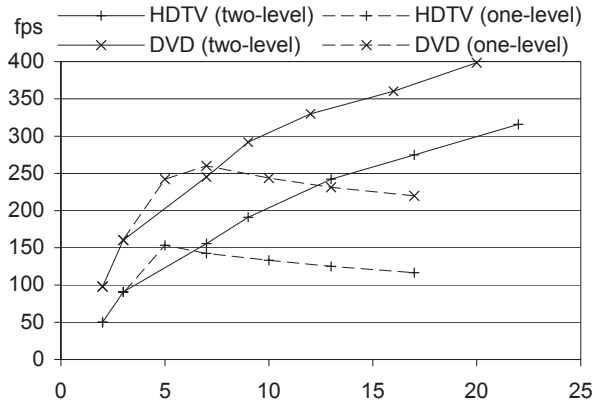

Figure 9: Frame Rate of Parallel MPEG-2 Decoders

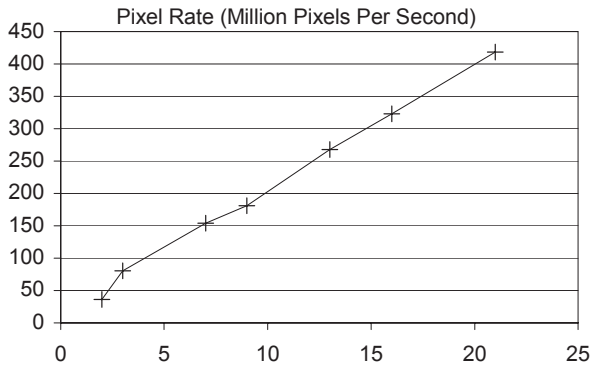

Figure 10: Pixel Decoding Rate of Two-Level Decoder

6. Parallel Sound Server

To provide a truely immersive experience, we need both a high resolution large scale visual display and a multi-channel spatialized audio system. In our first generation display wall system, a PC with multiple sounds cards supported up to 16 speakers; 10 surrounding the display, 4 in the back of the room, and 2 subwoofers. Besides having a limited number of sound channels, this single PC sound system was prone to failure due either to hardware or driver dependencies. So we designed and built a scalable distributed multi-channel parallel sound server [11].

Our display cluster workstations each came with an integrated sound card and this hardware gave us the base for a distributed parallel multi-channel sound system. The goal of our distributed sound system was that it should scale with the size of the cluster and networking bandwidth, and that it should be flexible.

The system we built satisfies these goals by streaming sound sources from a distributor computer to the cluster nodes (each running a program called soundlet) and synchronizing the sound cards on the cluster. The main obstacle in this approach is the synchronization of the sound cards. The ear is sensitive to timing variations of less than 1 ms. This is on the order of the latency of a network packet, and some operating systems have timing granularity as high as 10ms. The other problem is that the sound card crystals all have a slightly different frequency (which is also different from the computer timing crystal). As a result the sound cards tend to drift apart over time.

To solve the synchronization problem, we employed two techniques. First we require the distributor computer to have a sound card; this then serves as the master clock. Second we provide a jitter buffer on each of the soundlets. Each data packet sent out doubles as a network synch message. If at a network synch point the jitter buffer on a soundlet doesn't match, then we slow down or speed up the playback timing by locally re-sampling the sound in real-time. This results in each of the soundlets tracking the distributor's notion of time. Figure 11 depicts the architecture of such a distributed sound system.

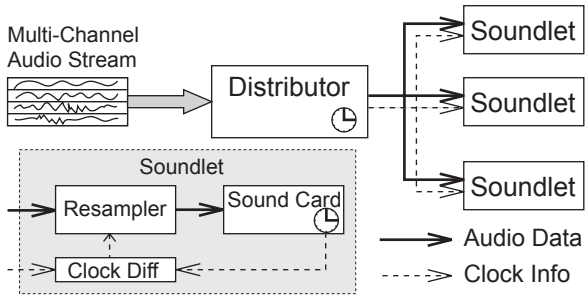

Figure 11: Distributed Parallel Sound Server Architecture

The final system is highly flexible. It allows us to change the number of channels of sound, location of a streamed channel, delay time, playback rate, synchronization rate, compression, and the distribution method of the streamed audio, all in a matter of a few keystrokes. In our current system all of this works together to give us a 48 channel spatialized sound system.

7. Conclusion and Future Work

Building larger scale display walls is important as we push to achieve resolutions approaching the visual acuity of human eyes. This brings new challenges in the areas of scalable projector calibration, data distribution/synchronization and cluster management. To summarize our conclusion:

We found that it is possible and desirable to use roughly aligned inexpensive portable projectors and perform automatic on-the-fly projective corrections to produce a seamless image. As long as the projectors do not exhibit any major pincushion or barrel distortion this correction can be done on commodity graphics cards (such as the GeForce2) at very little cost. Portable projectors have the benefit of being lightweight and inexpensive however may have to be augmented for computerized control.

We also found that, in order to distribute data efficiently across the cluster, we need to transmit the data in compressed form, and avoid unnecessary transfer of data. To achieve this, we designed parallel decoding systems for image and video. Sometimes we needed to augment existing file formats for data indexing.

Synchronization is a critical issue in any distributed systems. For image, video or 3D rendering, it is adequate to synchronize the frame buffer swapping of all graphics cards by exchanging short messages. For distributed audio, we must employ more sophisticated sychronization methods to compensate for the fact that different sound

cards have slightly different clock rates. We solve this problem by doing real-time audio stream re-sampling at each soundlet.

Finally, scaling a Windows cluster requires the capabilities to manipulate user interface elements in parallel. This is accomplished by multicasting events including mouse and keyboard, and allows for simultaneous cluster management.

While our 4×6 display wall has been operational for over 12 months now, there are still several issues remained to be addressed in future work:

• **Distributed Intelligent Storage**: As a cluster grows in scale, a centralized storage system becomes a bottleneck. Distributed file system, such as xFS [30] and Frangipani [28], and disk-directed I/O [16] were proposed to address this issue. Ideally, we would want to have a distributed storage system that utilizes the idle capacity and bandwidth of hard disks in all display nodes. This storage system should also be intelligent in that it can performan content-specific transformations on the data. For example, MCUs in an JPEG image will be stored in the nodes where it is most likely to be displayed, and the storage system can provide application pixels instead of just compressed bits of an MCU.

• **Scalable 2D content creation tools**: So far there is no tool specifically designed for creating and manipulating large format images or video content. It is extremely hard to do this with current desktop applications because a single PC's computing power is very limited and the screen size is too small to visualize the results. To solve this problem will call for parallelized versions of content creation tools that run directly on the display wall system.

• **Scalable virtual desktop environments**: It's desirable to run desktop applications direct on a display wall system as if it is one large virtual desktop. VNC wall [19] and Xplit [9] are two examples of attempts to provide such a solution. We plan to implement a Virtual Display Driver for Windows system and a distributed X server with built-in warping support for arbitrarily overlapped projectors.

As displays become ubiquitous through OLED and other new technologies on the horizon, continuing to push the envelop of using and building large displays will provide valuable insight into the future infrastructure and usage paradigm of high resolution displays.

8. Acknowledgements

This project is supported in part by Department of Energy under grant ANI-9906704, grant DE-FC02-01ER25456 and grant DE-FC02-99ER25387, by Intel Research Council and Intel Technology 2000 equipment grant, and by National Science Foundation under grant CDA-9624099 and grant EIA-9975011. Han Chen is supported in part by a Wu Fellowship.

9. References

[1] A. Bala, D. Shah, U. Feng, and D.K. Panda. Experience with software MPEG-2 video decompression on an smp pc. In *Proceedings of the 1998 ICPP Workshop on Architectural and OS Support for Multimedia Applications/Flexible Communication Systems/Wireless Networks and Mobile Computing*, pp. 29–36, 1998.

[2] A. Bilas, J. Fritts, and J.P. Singh. Real-time parallel MPEG-2 decoding in software. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.

[3] M. Brim, R. Flanery, A. Geist, B. Luethke, and S. Scott. Cluster Command & Control (C3) Tool Suite. Computer Science & Mathematics Division, Oak Ridge National Laboratory, http://www.epm.ornl.gov/torc/C3/Papers/pdcp-v2.0.pdf

[4] H. Chen, K. Li, and B. Wei. A Parallel Ultra-High Resolution MPEG-2 Video Decoder For PC Cluster Based Tiled Display System. To Appear In *Proceedings of the International Parallel and Distributed Processing Symposium*, April 2002

[5] H. Chen, R. Sukthankar, G. Wallace, and T.-J. Cham. Accurate calculation of camera homography trees for calibration of scalable multi-projector displays. Technical Report TR-639-01, Princeton University, September 2001.

[6] Y. Chen, D.W. Clark, A. Finkelstein, T. Housel, and K. Li. Automatic Alignment Of High-Resolution Multi-Projector Displays Using An Un-Calibrated Camera. *Proceedings of IEEE Visualization 2000*, Salt Lake City, Utah, October 2000.

[7] Chromium Project, http://sourceforge.net/projects/chromium/

[8] S. Eckart and C. E. Fogg. ISO/IEC MPEG-2 software video codec. *Proc. Digital Video Compression: Algorithms and Technologies 1995*, 100-109, SPIE, 1995.

[9] S. Feng. Xplit Research Report. Argonne National Laboratory, July, 2001. http://people.cs.uchicago.edu/~songyanf/code/papers/Xplit%20Research%20Report%201.doc

[10] J.A. Friesen and T.D. Tarman, Remote High-Performance Visualization and Collaboration. *IEEE Computer Graphics and Applications*, Vol. 20, No. 4, pp. 45-49, July/August 2000.

[11] A. Gupta, P. Cook. Parallel Distributed Multi-Channel Sound Server. Princeton University, Spring 2001. http://www.cs.princeton.edu/ugprojects/anoopg/senior/Fianl%20Write-UP.doc

[12] M. Hereld, I.R. Judson, J. Paris, and R.L. Stevens. Developing Tiled Projection Display Systems. *Proceedings of Fourth Immersive Projection Technology Workshop*, 2000

[13] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. WireGL: A Scalable Graphics System for Clusters. *Proceedings of SIGGRAPH 2001*, 2001.

[14] G. Humphreys and P. Hanrahan. A Distributed Graphics System for Large Tiled Displays, *Proceedings of IEEE Visualization*, 1999.

[15] Intel Corp. Intel JPEG Library, http://www.intel.com/software/products/perflib/ijl/

[16] D. Kotz. Disk-directed I/O for MIMD multiprocessors. *Proceedings of First Symposium on Operating System Design and Implementation*, pp. 61-74, 1994.

[17] K. Li, H. Chen, Y. Chen, D.W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J.P. Singh, G. Tzanetakis and J. Zheng. Building and Using A Scalable Display Wall System. *IEEE Computer Graphics and Applications*, 20(4): 671-680, July/August 2000.

[18] A. Majumder, Z. He, H. Towles, G. Welch. Color Matching of Projectors for Multi-Projector Displays. In *Proceedings of EUROGRAPHICS '2000*, Volume 19, 2000.

[19] National Center for Supercomputing Applications. VNC-Wall. http://www.ncsa.uiuc.edu/TechFocus/Deployment/DBox/Doc/vnc.html

[20] NCSA's Visualization and Virtual Environments group. http://www.ncsa.uiuc.edu/Divisions/DMV/Vis/Projects/Tiled-Wall/

[21] W.B. Pennebaker, J.L. Mitchell. Jpeg: Still Image Data Compression Standard. Kluwer Academic Publishers, ISBN: 044201272

[22] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, H. Fuchs. The Office of the Future: A Unified Approach to Image-Based Modeling and Spatially Immersive Displays. *Proceedings of ACM SIGGRAPH*, 1998.

[23] R. Raskar, M. Brown, R. Yang, W. Chen, G. Welch, H. Towles, B. Seales, and H. Fuchs. Multi-projector displays using camera-based registration. In *Proceedings of IEEE Visualization*, 1999.

[24] D.R. Schikore, R.A. Fischer, R. Frank, R. Gaunt, J. Hobson, and B. Whitlock. High-resolution multi-projector display walls and applications. *IEEE Computer Graphics Applications*, July/August 2000.

[25] SCMS - Smile Cluster Management System 1.2.2. http://smile.cpe.ku.ac.th/research/scms1.2.2/

[26] R. Sukthankar, R. Stockton, and M. Mullin. Smarter presentations: Exploiting homography in camera-projector systems. In *Proceedings of International Conference on Computer Vision*, 2001.

[27] R. Surati. *A Scalable Self-Calibrating Technology for Seamless Large-Scale Displays*. PhD thesis, Department of Electrical Engineering and Computer Sceince, Massachussetts Institute of Technology, 1999.

[28] C.A. Thekkath, T. Mann, and E.K. Lee. Frangipani: A Scalable Distributed File System. *Proceedings of the ACM Symposium on Operating Systems Principles*, pp. 224-237, Dec. 1997.

[29] The University of Minnesota. Power Wall. http://www.lcse.umn.edu/research/powerwall/powerwall.html

[30] R. Y. Wang, T. E. Anderson. xFS: A Wide Area Mass Storage File System. *Proceedings of Fourth Workshop on Workstation Operating Systems*, pp. 71-78, October 1993.

[31] R. Yang, D. Gotz, J. Hensley, H. Towles, and M.S. Brown. PixelFlex: A Reconfigurable Multi-Projector Display System. In *Proceedings of IEEE Visualization*, 2001.