Tools and Applications for Large-Scale Display Walls

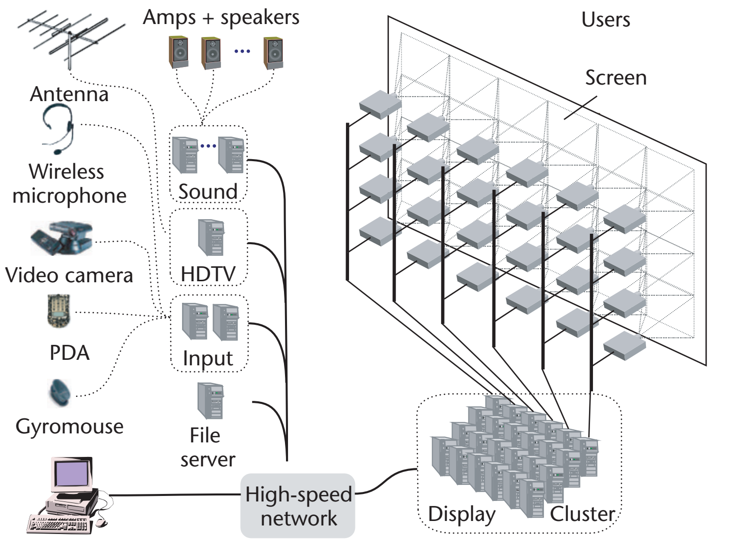
Grant Wallace, Otto J. Anshus, Peng Bi, Han Chen, Yuqun Chen, Douglas Clark, Perry Cook, Adam Finkelstein, Thomas Funkhouser, Anoop Gupta, Matthew Hibbs, Kai Li, Zhiyan Liu, Rudrajit Samanta, Rahul Sukthankar, and Olga Troyanskaya
Princeton University

To create a scalable and easy-to-use large-format display system for collaborative visualization, the authors have developed various techniques, software tools, and applications.

In the last two decades, compute power, storage density, and network bandwidth have improved by more than three orders of magnitude. In this same time period, however, display resolution has merely doubled. The increased processor and storage capacities have supported the computational sciences, but have simultaneously unleashed a data avalanche on the scientific community. As a result, scientific research is limited by data analysis and visualization capabilities. These new bottlenecks have been the driving motivation behind the Princeton scalable display wall project. The project started in 1998 with the goal of building a large-format, high-resolution display system with inexpensive commodity components.

Our first-generation display wall, built in March 1998, used an 18 × 8 foot rear projection screen and eight Proxima LCD commodity projectors.[1] This system had a resolution of 4,096 × 1,536 pixels and was driven by a network of eight Pentium II PCs running Windows NT.

In November 2000, we scaled the display up with 24 Compaq MP1800 digital light-processing (DLP) projectors and a network of 24 Pentium III PCs running Windows 2000. The system resolution is 6,144 × 3,072 pixels. The system also connects to I/O devices including mouse, PDA wireless inputs, video cameras, high-definition television (HDTV), and a distributed sound server, as Figure 1 shows. In our scaling efforts, we found that some previous techniques (such as geometric alignment) that were sufficient for eight projectors were excessively time consuming for 24 projectors and that some tools (such as an MPEG decoder) could no longer handle the system resolution. As a result of these findings, we've focused our research efforts on making display walls more scalable, usable, and useful for collaborative research.



1 Architecture of Princeton's second-generation display wall.

Managing tiled displays

To make the 24-projector display system easy to use, we've developed tools to align, color balance, and manage tiled display systems. The goal is to provide a large-format tiled display system with the look and feel of a large single display with accurate geometric alignment and good color balance.

Scalable automatic geometric alignment

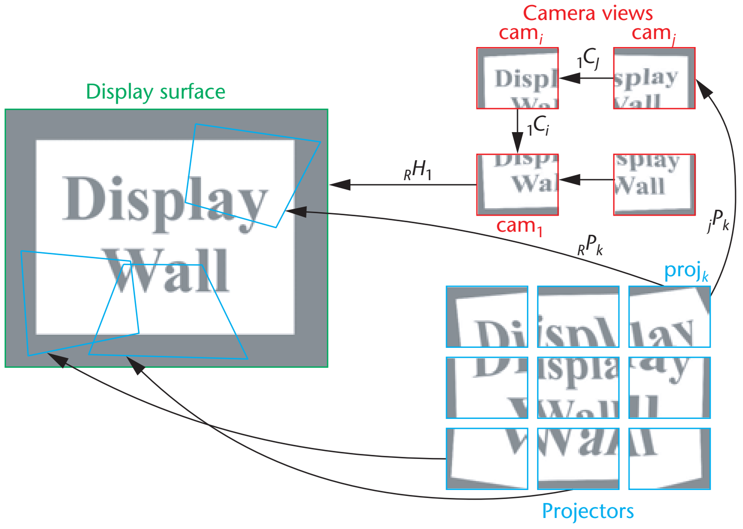
Manual geometric alignment is impractical for large-scale tiled display systems: The process is time consuming and the alignment error tends to propagate and increase as the system grows. The research challenge is to develop a scalable, automatic alignment method that lets computers warp images to achieve subpixel accuracy for a large array of commodity projectors.

Our approach uses an inexpensive, uncalibrated pan-tilt-zoom camera, which we can program to zoom into small sections of the display wall to detect feature points. This lets us align displays with orders of magnitude higher resolution than the camera's resolution, while still achieving subpixel accuracy. Our previous automatic alignment method achieved subpixel accuracy for an eight-projector system but didn't work well for our 24-projector system, taking over two hours to align a 24-projector system while achieving an accuracy of about two pixels.

Our new alignment research focuses on efficiency, accuracy, and scalability.[2] To achieve these requirements, we collect multiple zoomed-in camera views of feature points projected on the display wall. These detailed views let us determine with great precision the feature point positions in the camera's coordinate system. We significantly overlap the camera views to calculate transformations that map one view's coordinate system to its neighboring views. These camera views and transformations form the vertices and edges of a *homography graph*. We use a spanning tree of the homography graph to warp and merge all camera views into a mosaic view represented by the tree's root, as Figure 2 illustrates. However, we lose valuable feature correspondence information in forming the spanning tree, resulting in increased error in the feature point estimation and ultimately lower alignment accuracy of the display wall. To address this problem, we've developed an optimization algorithm that recaptures all of a homography graph's correspondence information in a tree.

First, we select a spanning tree that minimizes the path length between any pair of adjacent camera views. Next, we run a minimization algorithm that iteratively refines the homography associated with each edge in the tree by traversing all paths between the two sets of camera views separated by the edge. The process continues until the feature point location variance in the mosaic view is below a set threshold.

Using the refined homography tree, we can accurately create a mosaic view, in which we can locate all feature points in a global coordinate system. We then extrapolate and derive each projector's corner positions in the global frame. From the relative positions of projector corners, we can find a set of projective transformations, one per projector, that when preapplied to imagery will minimize the alignment discontinuities seen on the display wall.

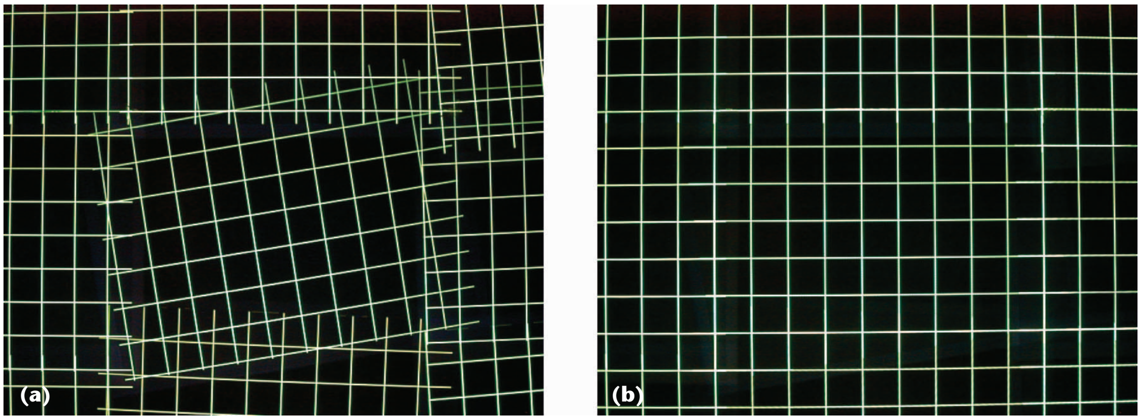


2 **Diagram of the alignment process showing relations among camera views, projectors, and display screen.**

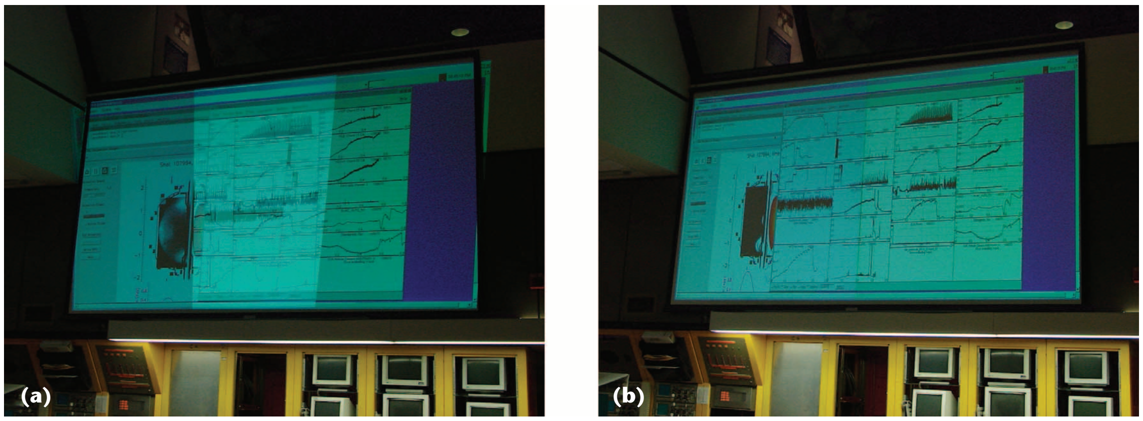
Using this algorithm, we align the 24-projector display system in under 10 minutes. The system reliably achieves an average alignment error of 0.55 pixels. We calculate this error by displaying a grid pattern and measuring the projected discrepancies between grid points in overlapping projector regions. Figure 3 shows an example of an arbitrarily misaligned projector image before and after the alignment.

We've also scaled the system down to a few projectors driven by a single PC. This lets us create inexpensive installations, eliminates the compute cluster, and lets us use unmodified desktop applications.

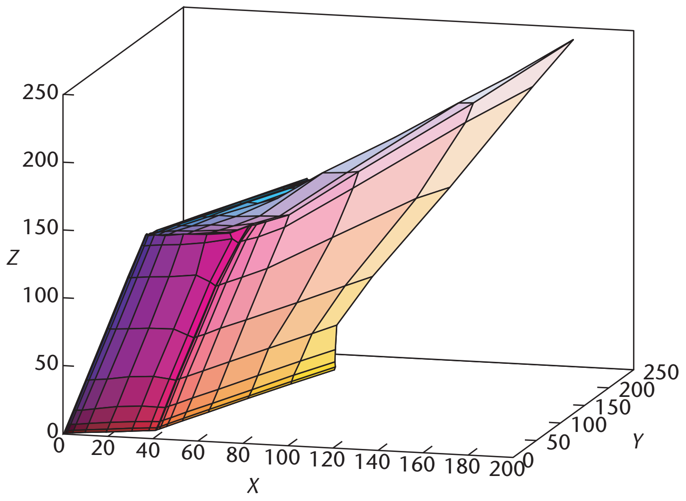
The challenge when automatically aligning a single-PC tiled display is intercepting the pixels and warping them before they're displayed. Typically, the operating system handles the rendering from the desktop through to the graphics card. Some researchers are investigating smart projectors[3] to achieve pixel warping; however, few commodity projectors currently support this.



3 **Grid pattern shown on the display wall (a) before and (b) after automatic alignment.**



4 Automatic alignment result of a two-projector display system running from a single PC.



5 The color gamut of a typical single-chip DLP projector plotted in CIE XYZ space.

To access the pixels, we pair graphics pipelines on multiheaded graphics cards, letting the Windows OS render desktop content normally to each pair's first frame buffer. We then copy the pixels to texture memory and warp them to the second frame buffer. Projectors are connected to the second set of frame buffers only, so they display just the transformed imagery.

Using this technique, we run an automatically aligned tiled desktop from a single PC. The alignment has subpixel accuracy. Refreshing the aligned screen at 30 Hz consumes about 10 percent of an 866-MHz processor's time.

Figure 4 shows before and after photos of the automatic alignment of a two-projector display system at the Princeton Plasma Physics Laboratory control room.

Color balance for tiled DLP projectors

Balancing a large projector array's color and luminance is a challenging problem in the calibration process. RGB channel balancing is a technique that has been employed successfully for LCD projectors,[4] but the emergence of DLP projectors has made this process more difficult. DLP projectors use a white enhancement technique, which produces a higher contrast ratio in single-chip DLP projectors by using a four-component color wheel with red, green, blue, and clear filters. The clear filter increases the luminance for less saturated colors. Unfortunately, this produces a nonadditive color space that's difficult to model, as Figure 5 demonstrates.

Our color-mapping algorithm operates across the full color gamut[5] as opposed to individual color channels. We first subsample each projector's color space using an inexpensive colorimeter. Next, we calculate the intersection of all of the projector gamuts, which gives us a common gamut. The algorithm then produces a color mapping for each projector that translates its native gamut into this common gamut. We apply this mapping in real time on commodity graphics cards by loading the map as a volume texture. Our results show that we can achieve good color balance with a 1.47-percent variance between projectors.

Tiled display-management tools

Managing a large-scale display wall built with commodity components can be time consuming because it involves many resources, including numerous projectors, computers, and software tools. The challenge is to make the large-scale display system have the feel of a single computer so an inexperienced user, such as a visual arts student, can use it easily and without extensive training.

Our tiled-display management tools, which we call DwallGUI,[6] coordinate all resources in a display wall system. DwallGUI consists of a server running on each of the display cluster nodes and a client running on the console PC.

DwallGUI lets users manage a selected set of projectors and their driving PCs. Users can power on or off sets of projectors or access the projectors' internal menus. Using the same method, users can issue commands to show images or movies or run certain applications. In addition, DwallGUI helps with process control, monitoring, and installations. Instead of using a keyboard-video-mouse switch to manage individual computers in a cluster, DwallGUI lets users broadcast the keyboard and mouse commands and simultaneously see the results on the display wall. For example, if a user needs to install a device driver on certain PCs in the cluster, he or she can use DwallGUI to select the desired PCs and perform the installation procedure while monitoring the progress on the display wall screen. The user need only issue the installation commands once for all selected PCs.

Rendering information

The main challenge to rendering information such as pixels and sound in a large-scale multiprojector system is delivering scalable performance with minimal communication requirements. We've developed an IMAX-quality parallel MPEG decoder, a parallel 3D rendering system, and a spatialized sound server.

Parallel MPEG decoder

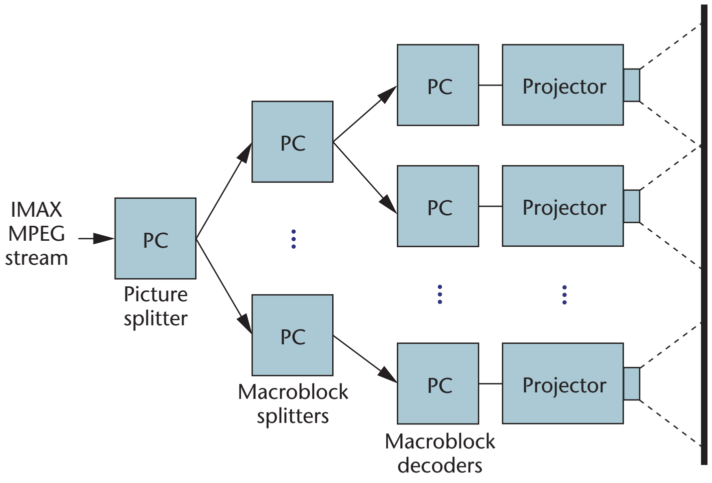
A large-scale multiprojector display system creates the possibility of showing high-resolution and high-quality video. This is especially desirable for the playback of detailed scientific visualization and simulation results, immersive telepresence, VR, digital cinema, and other entertainment applications. Although for some applications, a multistream, multiresolution approach might be adequate or even desirable, others, such as simulation playback, demand efficient decoding of high resolution compressed video streams.

Today's PC hardware can decode HDTV-resolution video streams in real time. However, tackling an IMAX-quality MPEG stream (for example, 4,000 × 3,000 pixels) is still an extremely challenging task requiring an order of magnitude more compute power. One possible, but undesirable, way to achieve this is through preprocessing. Preprocessing could be used to split a high-resolution video into multiple lower-resolution streams, re-encode and distribute them to the PC cluster nodes for playback in real time. But this process is time consuming, not flexible, and it reduces the video quality during re-encoding.

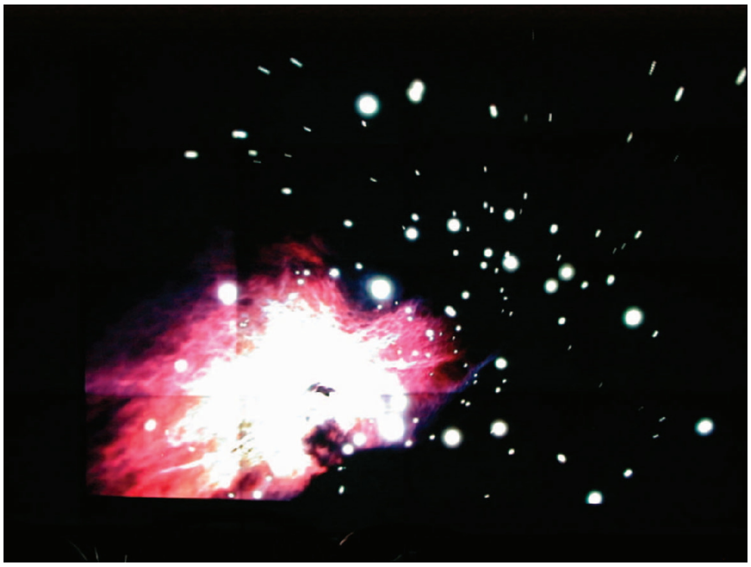
Data-driven parallelization techniques seek to avoid the decode-encode cycle by working directly on the compressed bitstream domain. Picture or slice-level parallelization works on shared memory symmetric multiprocessing machines, but not on PC clusters because of the high network bandwidth requirement for redistributing pixel data. Our previously reported approach parallelizes an MPEG decoder at macroblock level with minimal network traffic, making it suitable for cluster systems.[1] Unfortunately, this method doesn't scale well because of the high computational requirement of parsing an MPEG stream into macroblocks.

To address the computation bottleneck and achieve scalable high-resolution decoding, we use a hierarchical parallel decoding system, shown in Figure 6.[7] It consists of two levels of splitters and a set of decode and display servers. A root-level picture splitter scans the input bitstream and turns it into a sequence of pictures, which is a simple and fast operation. It sends the pictures to several second-level macroblock splitters in a round-robin fashion. The macroblock splitters parse the pictures into macroblocks, which they direct to appropriate macroblock decoders. The macroblock splitters also provide macroblock reference information so the decoders know how to exchange data with each other when a macroblock's motion vector references data not on the local node. The macroblock decoders then decode the macroblocks and display results on the projectors.

Because this approach only transfers encoded macroblocks and reference information between PCs, the network bandwidth requirement is much lower than for previous approaches.



6 Architecture of a two-level parallel MPEG decoder. A PC splits a high-resolution MPEG stream at picture level and then several PCs split the result into macroblocks.

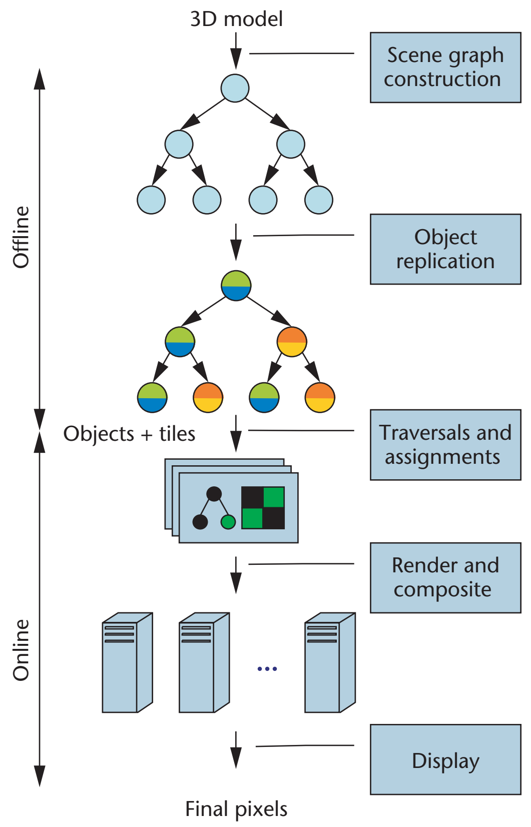


7 Playback of Orion flythrough video stream (3,840 × 2,800 resolution) at 38.9 frames per second.

Figure 7 shows the result of running such a decoder system with one picture splitter, four macroblock splitters, and a 4 × 4 tiled portion of our display wall as decoder and display servers. The implementation achieves 38.9 frames per second on the Orion flythrough MPEG video stream with 3,840 × 2,800 resolution. The maximum communication bandwidth requirement for any node in the PC cluster is less than 50 Mbits per second.

Parallel rendering with k*-way replication*

A high-resolution display wall introduces new challenges in parallel rendering of images using a PC cluster. The goal is to develop parallel rendering algorithms that achieve high utilization of hardware resources such as



8 Parallel rendering with *k*-way replication.

CPUs, graphics cards, and memories while requiring minimal network bandwidth.

A parallel rendering algorithm's performance depends mainly on how it partitions and distributes graphics primitives among rendering PCs. Chromium, for example, supports a sort-first partitioning algorithm that distributes OpenGL graphics primitives among the rendering servers in real time as the application generates them.[8] This approach is flexible and requires minimal memory on each PC, but requires a high-bandwidth network to keep up with the graphics hardware on the rendering PCs. To overcome this problem, it's usually possible to replicate static graphics primitives in a scene graph on the rendering PCs and transmit only a set of object IDs to inform a rendering PC to render part of the scene graph. Although this approach requires a lower network bandwidth, it usually requires every PC to have enough memory space to hold the entire scene graph.

We propose a k-way replication strategy to overcome the limitations of both of these approaches.[9] The strategy replicates the graphics primitives of each scene graph object on k of the n rendering PCs ($k << n$), which avoids full replication of the scene graph, but still allows efficient, view-dependent partitioning of the objects during interactive parallel rendering. With dynamic load-balancing algorithms, a system employing k-way replication can achieve rendering performance and network bandwidths close to full replication, while incurring storage costs closer to 1-way replication.

Our investigation has focused on k-way partitioning strategies and dynamic load-balancing algorithms. During the offline phase (top of Figure 8), we organize the input 3D model into a multiresolution hierarchy of objects and choose k rendering PCs to store each object. This phase aims to distribute copies of the objects such that a later load-balancing algorithm can perform an effective view-dependent partition, while avoiding starvation of any server as the user zooms in to view a subset of the scene. During a later online phase (bottom of Figure 8), we perform a dynamic, view-dependent partition of the objects, selecting exactly one out of k servers to render each object so as to balance the load among the rendering servers and minimize network transmission costs. We use a peer-to-peer, sort-last communication strategy to composite the rendered images into the final image.

We've investigated the tradeoffs of various choices at each stage of the process using an 800-Mbyte scanned-in 3D model of Michelangelo's *David*. We rendered the model on a 24-node PC cluster in which each node has 256 Mbytes of memory. For object replication, we've experimented with various granularities of k. For dynamic object assignments, we've tried several algorithms, such as assigning objects to the least loaded server first, or optimizing assignments to minimize pixel redistribution and composition overheads.

We've learned from our experiments that with a small k (for example, $k = 4$), efficiency improves quickly, approaching n-way (full) replication. Choosing k is a tradeoff between communication and memory usage. A simple method for dynamic screen partitioning based on proximity to dots representing servers performs quite well. Our rendering pipeline from client to display can render Michelangelo's *David* at about 40 million polygons per second with 65-percent efficiency. This shows that we can indeed support 3D models larger than any single PC's memory capacity while retaining the reduced communication overheads of dynamic view-dependent partitioning.

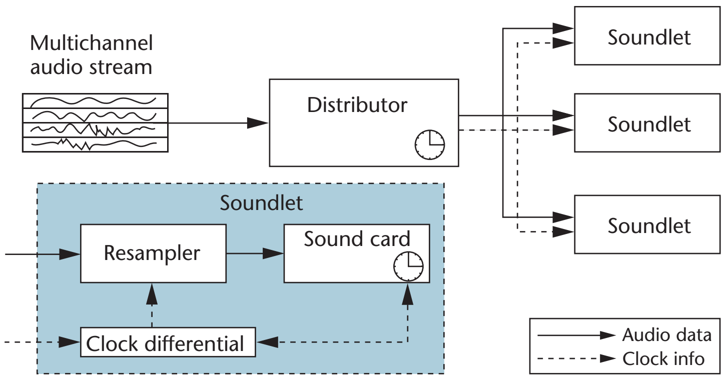
Distributed sound server

Providing a truly immersive and informative experience requires both a high-resolution, large-scale visual display and a multichannel spatialized audio system. Our distributed sound server aims to scale with the cluster's size and networking bandwidth, thus providing flexibility in audio rendering. The old adage of "sound guides the eyes" is certainly true in our system, as we can use spatially targeted auditory events to draw attention to important aspects of visualizations that might lie outside the visual field.[10] Also, combined with multichannel audio, the large visual display can be useful for browsing large collections of sound effects and music.

Our system streams sound sources from a distributor computer to the display-cluster computers, each of which runs a soundlet program. The main obstacle is that because the soundlets play on different PCs, they must be synchronized. The human ear is sensitive to timing variations of less than 1 ms. Network packet latency is less than this threshold, letting our data packets double as synchronization markers. This is important because the sound card crystals all have slightly different frequencies, causing them to drift apart over time.

To synchronize the sound cards we first require the distributor computer to have a sound card; it then serves as the master clock. Second, we provide a jitter buffer for each soundlet. If, at a network-synchronization point, the jitter buffer on a soundlet doesn't match, we slow down or speed up the playback timing by locally resampling the sound in real time. This results in each of the soundlets tracking the distributor's notion of time. Figure 9 depicts the architecture of such a distributed sound system.

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



9 **Architecture of a distributed sound server that renders multichannel audio.**

Applications using tiled displays

Scientific visualization and collaboration have driven the development of display walls. We've been working with scientists to develop software tools for genomic data visualization, isosurface extraction, and collaborative control rooms. We've also developed a synchronized programming environment to help scientists bring legacy applications to display walls.

Genomics data visualization

We've been working with genomic scientists on visualizing large-scale microarray data to facilitate analysis and discovery of biological information. Biologists create microarray data sets by measuring the activity level of genes in an organism under various conditions. This results in matrices containing hundreds of measurements for each gene in the genome. Simple organisms such as yeast have about 6,000 genes, whereas humans have about 30,000. Biologists analyze this data to identify genes' biological functions, searching for patterns that imply that a group of genes behave together to perform a task. Although biologists use many purely numerical methods for this analysis, the high level of noise in the data and the lack of a gold standard for verifying results cause data visualization to be a key component of analysis. However, normal desktop resolutions let researchers view detailed data for only about 100 genes simultaneously. This seriously hampers efforts to find correlations across disparate genes, as well as the ability to analyze multiple aspects of the data concurrently.

To address this problem, we've developed applications that let researchers view genomic data on large high-resolution displays. The 24-node display wall provides 24 times the information of a normal desktop, or raw data for about 2,400 genes. Using the display wall's large scale and high resolution lets us look at multiple views of each microarray data set, enabling more detailed analysis of relationships between genes and groups of genes. For example, using principal component analysis (PCA), we define a 3D space that we project the data onto. This projection groups genes with similar measurements near each other. These groupings, which can be dense and hard to distinguish on a desktop display, become easier to distinguish when viewed on a large display.

We've deployed a display wall at the Lewis-Sigler Institute for Integrative Genomics. Figure 10 shows our

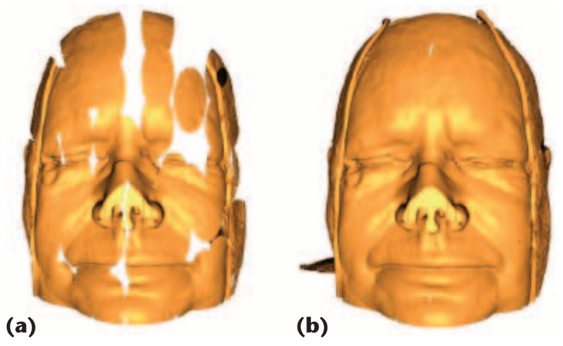
10 **Visualizing genomics data sets on a tiled display system.**

system displaying a yeast cell cycle data set on a tiled display. On the left is a display of raw data measurements and on the right is our PCA projection. The display's large format creates an environment in which multiple researchers can interact collaboratively. Such collaborations are especially critical in fields like genomics, where computational and biological researchers work side by side, analyzing and interpreting the data.

Scalable isosurface extraction

Applications such as large-scale simulations typically generate scalar fields and store them as volumetric data sets. A known method for visualizing the scalar field is to display isosurfaces where $F(x, y, z) = v$ for a given threshold v. Visualizing the isosurfaces of massive data sets on a scalable display system requires an algorithm that can be efficiently parallelized using limited network bandwidth.

We've developed a hybrid isosurface extraction algorithm that shares several features with existing acceleration methods. The algorithm casts rays into an octree to identify visible seed cells and then uses propagation to extend the isosurface from the seed cells. Unlike previous propagation methods that propagate to the whole



11 We achieve fast isosurface extraction by casting rays into the data set and propagating. (a) After 1 second, 85 percent of the surface is extracted, and (b) after 2 seconds it's 99.5-percent extracted. Extracting the remaining 0.5 percent takes 16 seconds.

isosurface, our method uses distance and viewing criteria to decide where to stop propagation, and thus generates only a small piece of the isosurface connected to each seed cell.[11] We patch these pieces together to form a view-dependent region of the isosurface, which includes all visible triangles as well as a small number of occluded triangles near the visible surface (see Figure 11).

Because the isosurface extraction from each seed is independent of the others, this method is natural as the base method for building a parallel isosurface visualization system. We've built such a system for our tiled display. The user selects viewing parameters from the console computer. Based on these parameters, the system calculates an individual viewport for each display tile. The display computers load the data set and perform the isosurface extraction based on their viewport setting. The user can choose between local and remote display options. When the system displays the data locally, the display wall computers can provide high-resolution visualization. When it displays the data remotely, the display wall computers send extracted triangles to another computer console for progressive display. This enables visualization on a desktop computer of limited computation power. In both scenarios, changing viewing parameters serves as a synchronization signal to interrupt computation for the previous frame, clear the screen, and start the new frame.

Synchronized programming environment

There are many existing closed- and open-source applications that users want to bring to a tiled display. Running such applications from a PC cluster can be challenging without proper tools.

One way to run an application on a tiled display wall is to use a synchronized program execution model. In this model, each server runs a duplicate instance of the application, similar to techniques used in the fault-tolerant community. However, we change the environment information for each instance, such as which tile in the display they represent. This model aims to minimize communication over the network. The system need send only control messages such as user input and synchronization, which tend to require little bandwidth.

In the synchronization model, we establish a synchronization boundary such that within this boundary all instances assume identical behavior. We've experimented with having this boundary at both the system level and the application level. With the synchronization boundary at the system level, each server produces identical graphics primitives and the graphics accelerator performs tile-specific culling such that each display-tile renders only the primitives falling within its server's screen area (see Figure 12). This technique is especially useful if the application's source code isn't available. Moving the synchronization boundary to the application level enables more optimizations. A view-dependent software layer can restrict itself to generating tile-specific primitives (rather than generating all of the primitives for the scene). An example would be a scene graph render program that organizes the scene data in a hierarchy of objects. Given a tile-specific view frustum, the program can remove objects falling completely outside the frustum. Such an approach can improve processing performance as well as data transfer performance.

We've implemented a synchronization framework for both system and application levels.[12] The framework consists of barrier synchronizations set up on certain function calls. One server acts as a coordinator and broadcasts the result of the function call to all other servers. At the system level, we use dynamic linked library replacement to intercept and synchronize on frame buffer swaps, timer calls, and I/O operations. At the application level, we've implemented a simple API including calls such as `SynchronizeResult()`, which synchronizes a barrier and distributes results.

Our results show that the synchronization communication overhead is small, less than 500 bytes per frame. For immediate-mode 3D graphics applications, we've achieved speedups of 1.2 to 4.2 times that of a client-server approach (such as distributing primitives). In general, the synchronization model performs better when there are many primitives and the per-primitive computation time is low.

Multiuser shared display system

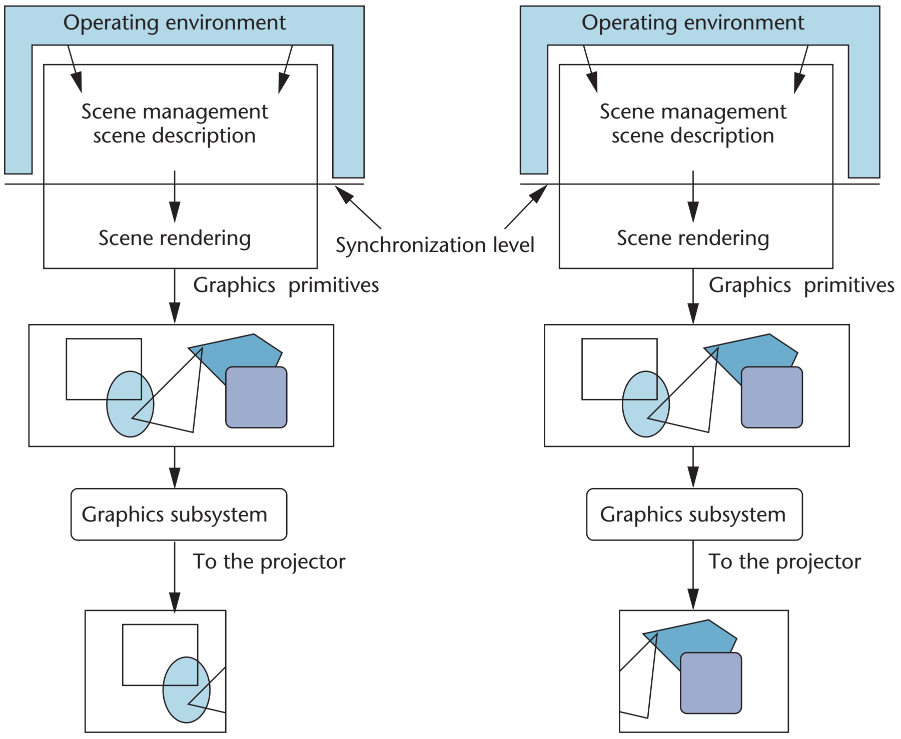
A large-format display system for a collaborative environment should virtually connect displays such that users can move display information seamlessly from one display to another. Ideally this will provide support for collaboration independent of the platforms, operating systems, and applications used. Such a system should also let multiple users work within the shared display space simultaneously.

Existing tools that support display sharing are either platform dependent (HP SharedX, for example) or too coarse grained (such as virtual network computer, or VNC, which shares the entire desktop).

To accommodate cross-platform display sharing with finer-grained control, we extended the VNC protocol. We added a ShareWindows message, which only sends the pixels of selected windows. The modified VNC server runs on a collaborator's PC and lets collaborators share or unshare selected windows to a group display.

The system should also let multiple users simultaneously interact with the shared display. Existing windowing systems assume a single-user model with a single cursor, keyboard, and input focus. Our multicursor X window manager[13] provides for multiple simultaneous cursors at the desktop level and lets multiple users concurrently interact with all components on a shared display, including applications, system menus, and window positioning.

We've deployed an initial version of the multiuser shared display system in the control room at the Princeton Plasma Physics Lab, as Figure 13 shows. The shared display system incorporates the multicursor window manager with application sharing. Feedback from control room users has been positive, and the system has quickly become a necessary collaborative mural for the lab's experimental fusion research.



12 **Synchronized application execution with tile-specific clipping.**

13 **Shared display using multicursor X and application sharing in the control room of Princeton Plasma Physics Lab.**

Future work

During the development of our second-generation display wall system, we focused on usability, scalability, and application impact. Large-scale, high-resolution display systems can significantly impact digital design, scientific data visualization, collaboration, and other disciplines. The key to achieving such impact, however, is to work closely with researchers in other disciplines to identify key problems.

Display walls are still difficult to use. To convince real users to use them, we must continue making display systems more usable. Managing component failures is a significant challenge in scaling display walls. When we scaled from eight to 24 projectors, we purchased additional components to prepare for component failures. However, the failure rates exceeded our anticipation. Since November 2000, our average annual component failure rate has been six projectors, three graphics cards, two fans, and one PC power supply. This averages to about one component failure a month for a 24-projector tiled display wall driven by a 24-PC cluster. We found that automatic calibration and cluster-management tools are extremely helpful for quick recalibration and reconfiguration after component failures, especially when digital design classes and student labs are using the display wall system. However, challenges remain in failure detection, correction, and tolerance for display walls. Tools providing reduced resolution or performance modes to tolerate component failures would reduce ownership costs and increase the reliability of heavily used display wall systems.

As we've shown, we can achieve high parallel rendering performance with relatively low network bandwidth requirements by carefully designing algorithms with replicated data on a cluster. However, we've only considered cases in which data structures fit in main memory. The next challenge is to develop out-of-core algorithms to deal with massive amounts of data.

We were pleasantly surprised by the positive feedback from the shared display system at Princeton Plasma Physics Lab and suspect that such a system is an important use case for the display wall. Our work on a multicursor X window manager is a first step toward providing systems software support for a shared display system. Ultimately, we want to let users with heterogeneous computing devices communicate with each other and visualize each other's information effortlessly and seamlessly. ■

Acknowledgments

US Department of Energy grants DE-FC02-01ER25456 and B543440 and US National Science Foundation grants CNS-0406415 and EIA-0101247 supported this work in part.

References

1. K. Li et al., "Building and Using A Scalable Display Wall System," *IEEE CG&A*, vol. 20, no. 4, 2000, pp. 671-680.
2. H. Chen et al., "Scalable Alignment of Large-Format Multiprojector Displays Using Camera Homography Trees," *Proc. IEEE Visualization*, IEEE CS Press, 2002, pp. 339-346.
3. R. Raskar et al., "iLamps: Geometrically Aware and Self-Configuring Projectors," *Proc. ACM Siggraph*, ACM Press, 2003, pp. 809-818.
4. A. Majumder and R. Stevens, "LAM: Luminance Attenuation Map for Photometric Uniformity across a Projection Based Display," *Proc. ACM Virtual Reality and Software Technology*, ACM Press, 2002, pp. 147-154.
5. G. Wallace, H. Chen, and K. Li, "Color Gamut Matching for Tiled Display Walls," *Proc. Immersive Projection Technology Symp.*, ACM Press, 2003, pp. 293-302.
6. G. Wallace, "Display Wall Cluster Management," *Workshop on Commodity-Based Visualization Clusters*, *IEEE Visualization*, 2002; http:/cs.princeton.edu/omnimedia/papers/Display%20Cluster%20Management.doc.
7. H. Chen, K. Li, and B. Wei, "A Parallel Ultra-High Resolution MPEG-2 Video Decoder for PC Cluster Based Tiled Display System," *Proc. Int'l Parallel and Distributed Processing Symp.* (IPDPS), IEEE CS Press, 2002, p. 30.
8. G. Humphreys et al., "Chromium: A Stream Processing Framework for Interactive Graphics on Clusters," *Proc. Siggraph*, ACM Press, 2002, pp. 693-702.
9. R. Samanta et al., "Parallel Rendering with K-Way Replication," *IEEE 2001 Symp. Parallel and Large-Data Visualization and Graphics*, IEEE Press, 2001, pp. 75-84.
10. P. Cook et al., "N>>2: Multi-Speaker Display Systems for Virtual Reality and Spatial Audio Projection," *Proc. Int'l Conf. on Auditory Display* (ICAD), 1998; http://www.icad.org/websiteV2.0/Conferences/ICAD98/papers/Cook/Cook.pdf.
11. Z. Liu, A. Finkelstein, and K. Li, "Progressive View-Dependent Isosurface Propagation," *Computers & Graphics*, vol. 26, no. 2, 2002, pp. 209-218.
12. Y. Chen et al., "Software Environments for Cluster-Based Display Systems," *Proc. Symp. Cluster Computing and the Grid*, IEEE CS Press, 2001, p. 202.
13. G. Wallace et al., *A Multicursor X Window Manager Supporting Control Room Collaboration*, tech. report TR-707-04, Dept. of Computer Science, Princeton Univ., 2004.

Grant Wallace *is a research staff member in the Department of Computer Science at Princeton University. His research interests include collaborative software environments and scalable display systems. Wallace has an MS in computer science from Rutgers University. Contact him at gwallace@cs.princeton.edu.*

Otto J. Anshus *is a professor in the Department of Computer Science at the University of Tromsø, Norway. His research interests include operating systems and cluster and Grid computing. Anshus has a PhD in computer science from the University of Tromsø, Norway. He is a member of the IEEE Computer Society, the ACM, and the Norwegian Computer Society. Contact him at otto@cs.uit.no.*

Peng Bi *has an MS in computer science from Princeton University. Her research interests include security, operating systems, and large displays. Contact her at pbi@cs.princeton.edu.*

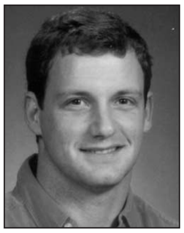
Han Chen *is a research staff member in the IBM T.J. Watson Research Center. His research interests include distributed computing systems, scalable display systems, and multimedia. Chen has a PhD in computer science from Princeton University. He is a member of the IEEE Computer Society. Contact him at chenhan@us.ibm.com.*

Yuqun Chen *is a researcher at Microsoft Research. His research interests include software protection, computer security, operating systems, and large displays. Chen has a PhD in computer science from Princeton University. Contact him at yuqunc@microsoft.com.*

Douglas Clark *is a professor of computer science at Princeton University. His research interests include computer architecture, low-power techniques, and clocking and timing in digital systems. Clark has a PhD in computer science from Carnegie Mellon University. Contact him at doug@cs.princeton.edu.*

Perry Cook *is a professor of computer science, with a joint appointment in music, at Princeton University. His research interests include music synthesis and modeling, animation, and music perception and cognition. Cook has a PhD in electrical engineering from Stanford University. He is a member of the IEEE, the ACM, and president of the International Computer Music Association. Contact him at prc@cs.princeton.edu.*

Adam Finkelstein *is an associate professor in the Department of Computer Science at Princeton University. His research interests include computer graphics, with a recent emphasis on nonphotorealistic rendering and animation. Finkelstein has a PhD in computer science from the University of Washington. Contact him at af@cs.princeton.edu.*


Thomas Funkhouser *is an associate professor in the Department of Computer Science at Princeton University. His research interests include computer graphics, geometric modeling, and shape analysis. Funkhouser has PhD in computer science from the University of California, Berkeley. Contact him at funk@cs.princeton.edu.*

Anoop Gupta *has a BSE in computer science from Princeton University.*

Matthew Hibbs *is a graduate student at Princeton University. His research interests include visualizing and analyzing genomic data to identify gene and protein functions. Contact him at mhibbs@cs.princeton.edu.*

Kai Li *is a Charles Fitzmorris Professor in the Department of Computer Science at Princeton University. His research interests include operating systems, parallel architecture, scalable display systems, and data exploration. Li has a PhD in computer science from Yale University. He is a senior member of the IEEE Computer Society and a Fellow of the ACM. Contact him at li@cs.princeton.edu.*

Zhiyan Liu *is a PhD candidate in the Department of Computer Science at Princeton University and a software engineer at Google. Her research interests include large scale data. Contact her at zhiyan@google.com.*

Rudrajit Samanta *is a Studio Tools software engineer at Pixar Animation Studios. His research interests include computer graphics, large-scale displays, and parallel computing. Samanta has a PhD in computer science from Princeton University. Contact him at rudro@pixar.com.*

Rahul Sukthankar *is a principal research scientist at Intel Research and adjunct research professor at Carnegie Mellon University. His research interests include computer vision and machine learning. Sukthankar received a PhD in robotics from Carnegie Mellon. Contact him at rahuls@cs.cmu.edu.*

Olga Troyanskaya *is an assistant professor in the Department of Computer Science and the Lewis-Sigler Institute for Integrative Genomics at Princeton University, where she leads a bioinformatics and functional genomics group. Her research interests include biological data integration, visualization of genomic data, microarray analysis, and prediction of protein function and biological pathways. Troyanskaya received a PhD in biomedical informatics from Stanford University. Contact her at ogt@cs.princeton.edu.*