High-Resolution Multiprojector Display Walls

Daniel R. Schikore, Richard A. Fischer, Randall Frank, Ross Gaunt, John Hobson, and Brad Whitlock
Lawrence Livermore National Laboratory

Visualization hardware and software now being built will display up to 15 times the number of pixels in a typical desktop display.

Recent advances in scientific simulation have rapidly led to the generation of multi-terabyte-size data sets of unprecedented spatial and temporal resolution. The interactive visualization resources these data sets demand overwhelm the display capabilities of the typical desktop system, both in terms of raw graphics processing power and display resolution. The Accelerated Strategic Computing Initiative (ASCI) Program of the United States Department of Energy (DOE) is building a comprehensive environment for performing scientific simulation, visualization, and analysis of such data sets.[1] As part of this environment, we at Lawrence Livermore National Laboratory (LLNL) are building visualization hardware and software systems that can display up to 15 times the number of pixels in a typical desktop display.

The scientific simulations of interest at LLNL have many different mesh-type and computational-zone characteristics, but share the same basic visualization problem—the impossibility of visualizing the entire data set at full resolution on typical desktop displays. The largest simulation performed to date is a turbulence calculation with a regular grid resolution of $2{,}048 \times 2{,}048 \times 1{,}920$, for a total of more than 8 billion computational zones per time step.[2] For visualization purposes, a single variable was saved and quantized to 8-bit resolution at 274 time steps, for a total of 2 Tbytes of time-varying visualization data.

In the context of postprocessing and analysis for terascale scientific simulations, display wall capabilities look desirable for several reasons. First and foremost, data set resolution now exceeds that of any individual display device. Additional considerations include increased performance from multiple graphics cards and the physical size of the display—useful for presentation and collaboration in addition to enhancing the sense of immersion. For these reasons, we have pursued and deployed high-resolution tiled displays in conference rooms and offices at LLNL.

The current visualization environment at LLNL centers on a pair of SGI Onyx2 visualization servers with several Tbytes of fiber-channel disk storage. The larger system is a 64-processor (R12000) machine with 16 Infinite Reality (IR) pipes. We currently use this system to drive a 15-projector display wall (see Figure 1), two 2×2 tiled flat-panel displays deployed in user offices, and classical desktop displays in several offices.

Multipipe display systems

A variety of multipipe, multiscreen display systems have been introduced in recent years, initially in academic institutions and eventually as commercial products. The typical display includes multiple projectors situated for front or rear projection onto a screen. Screen geometry can be flat, curved, or surround, while the materials have included solid single-piece walls, flexible screens, and tiled panels, both opaque and transparent. Individual decisions typically must satisfy floor space, graphics hardware, interaction, and expense constraints.

A number of high-level frameworks have been designed to facilitate the development of applications for display walls and similar display devices. Some of these frameworks focus on virtual reality applications,[3-5] while others focus on the high-resolution display aspects of multipipe systems.[6-8] Toolkits such as Iris Performer[9] deliver an entire infrastructure for scene description and rendering for multipipe environments, while lower level libraries such as SGI's Multi-Pipe Utilities (MPU, http://www.devprg.sgi.de/devtools/tools/MPU) are designed as thin layers of abstraction that give more control to the programmer.

In this article we describe our experience in building tools for large display walls for scientific analysis and visualization. Our goal was to achieve high-performance rendering on large multipipe display systems, both for new applications and existing single-pipe applications. Our approach resembles other multipipe systems, with a bias against placing restrictions on data formats and locality—a key feature for applications that handle terascale data. We will describe our hardware system design, low-level software library, and experiences with several applications that use the library for multipipe rendering.

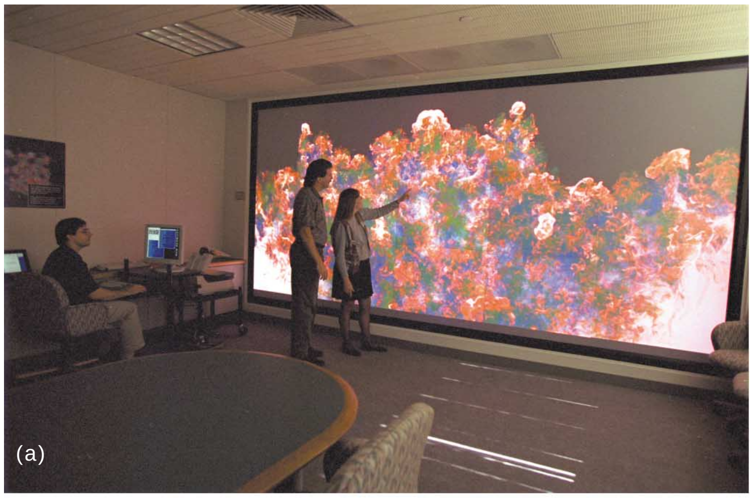
System design

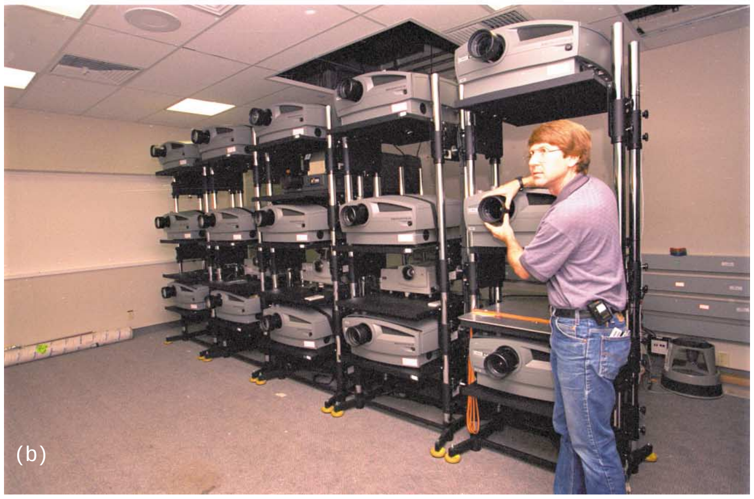
The 15 projectors of the display wall are arranged in a 5 × 3 array, producing a display of 6,400 × 3,072 pixels across a 16 × 8-foot rear-projection screen. A key goal of the display was to edge-match each projected image to its neighbor without overlap or separation lines. We wanted to maximize the number of pixels displayed, but image blending across the display was deemed undesirable because blending would sacrifice a large percent of pixels, especially around the four edges of each center row image. Achieving edge-matched alignment of all 15 images isn't practical using traditional 3-gun CRT projectors. CRT projector convergence drifts over time, and important areas such as the corners are the most troublesome to converge.

We opted for single-lens LCD-based projectors that maintain convergence for the life of the projector. We selected Electrohome DLV-1280 projectors for their high resolution, high lumen output, and color-balancing features, and for the key ability to finely adjust the lens position horizontally and vertically without moving the projector body. The lens adjustment controls physically move the lens, providing a more exact pixel alignment than electronic whole-pixel image positioning (although the DLV-1280 supports electronic positioning as well).

Color-balancing all 15 projectors proved a difficult task. The DLV-1280 allows for adjusting the red, green, and blue levels at six points along the gamma curve, resulting in 270 color settings for the entire wall display. Each projector varies slightly in color output, but the color output of Xenon lamps varies greatly between bulbs and drifts over time. A well color-balanced display will drift out of balance in just a few weeks of use. We are developing a computer-assisted color-balancing system in which the output from a color analyzer is fed into a portable computer, compared to a standard setting, and used to adjust each projector's color through a serial input.

We also built and deployed tiled, high-resolution flat-panel displays in offices. The displays consist of four LCD panels, arranged 2 × 2, closely spaced together in a single enclosure. The glass in the panels butts together, bringing the distance between viewable pixels of neighboring panels to approximately 3/8 inch. Our current tiled office display brings more than 5 million pixels to the user's office in a 29 × 33-inch unit that mounts on the wall. Users can take advantage of this extra viewing space as extended console space or as a 2,560 × 2,048 mini display wall. From a software standpoint there's no fundamental difference between display walls of projectors or tiled flat panels.



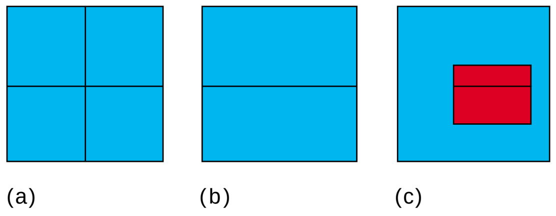


1 **This conference room display wall (a) consists of 15 rear-projected (b) 1,280 × 1,024 panels.**

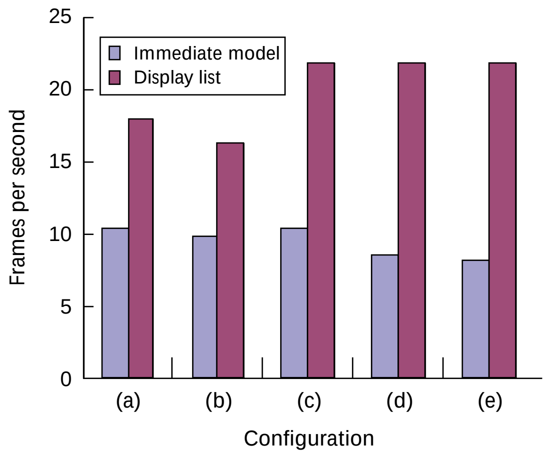
Low-level library: VDL

To make it easy for application developers to support the wall and possible to achieve scalable rendering performance, we developed a low-level interface library, referred to as the Virtual Display Library (VDL). VDL provides a simple application interface for threaded, multipipe rendering that provides hooks for realizing application-specific data and geometry culling techniques. Our initial goals for this library include

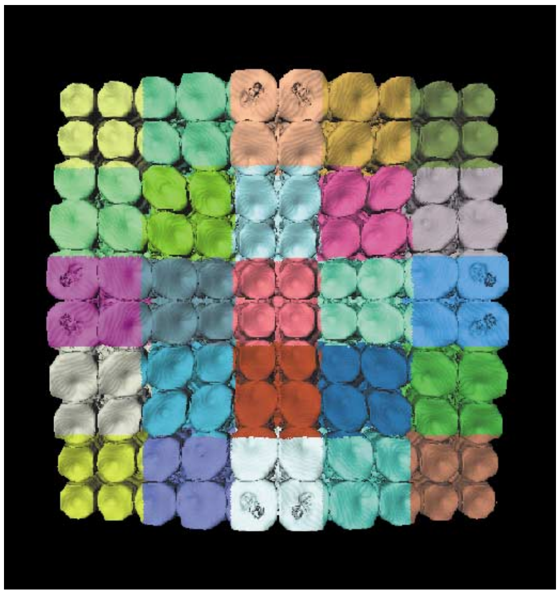
- providing a high level of abstraction so that little effort is required to modify an existing application to use a display wall;
- allowing access to lower levels of abstraction so that programmers may optimize primitive submission for performance;
- allowing flexible and easy configuration for multiple, varied hardware configurations;
- avoiding imposing restrictions on application event handling;



2 **Layers of abstraction in describing high-resolution display walls. (a) A display wall made up of 4 projectors (physical displays). (b) Two logical displays correspond to two graphics pipes. (c) On the full virtual display, a single canvas has been created, realized through the creation of two tiles.**



3 **Performance tests for the high-level VDL abstract layers. (a) Original benchmark application at 984 × 984. (b) One-pipe VDL application. (c) Two-pipe VDL with same window size. (d) Four-pipe VDL with same window size. (e) Four-pipe VDL at 1,968 × 1,968.**



4 **Decomposition of an isosurface into 25 bins.**

- avoiding imposing restrictions on application data models and access patterns;
- providing tiled offscreen rendering capabilities for generation of high-resolution movies or printing; and
- performing in a scalable manner.

The library's layered architecture is driven by the hardware used in constructing the display wall. At the lowest level are the projectors, or the physical displays. At the next level are the IR pipes, referred to as logical displays, which drive one or more physical displays. At the highest level of abstraction, there can be one or more virtual displays that represent a single high-resolution display system. A rectangular display region on the virtual display is referred to as a canvas, which is made up of multiple tiles, one for each graphics pipe. An example display wall configuration appears in Figure 2.

The library provides basic display management services for applications. VDL handles the creation and management of windows on multiple graphics pipes, creates and manages threads that render in parallel to the windows, performs view matrix transformations to provide a high-level abstraction to the application, and synchronizes the double buffering of multiple windows. VDL can also perform these operations for abstract devices, simplifying the creation of high-resolution movies through offscreen rendering and tiling.

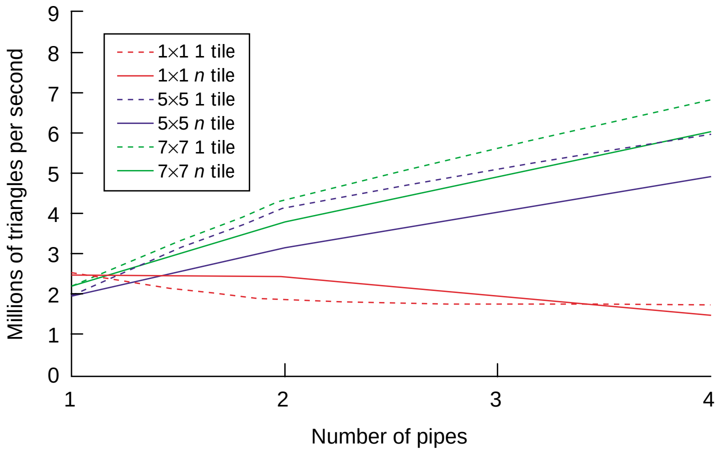
Because VDL is a low-level library, it's particularly important that the library not impose undue overhead on the application, limiting scalability. We took one measure of this overhead from an in-house OpenGL performance analysis tool. The benchmark code displays a mesh consisting of more than 318,000 triangles with an average strip length of 20. Each triangle is assigned a different color. Figure 3 shows measurements of the frame rate of the original OpenGL benchmark for comparison with a version modified for use with VDL. The modified version uses the high-level abstractions of VDL, performing no optimizations such as frustum culling for the individual tiles.

We performed tests using both immediate mode rendering and display lists. To isolate performance of the library from the fill rate of the graphics pipes, we performed the first three tests with an increasing number of pipes managing a constant-size VDL canvas of 984 × 984 pixels. As shown in Figure 3b, a small performance drop occurred for a single pipe due to the added function call overhead of the VDL interface. When scaled to two or four pipes (Figure 3c and 3d), performance with display lists was better than for the single-pipe application, while the immediate mode tests showed signs of scalability limitations. Figure 3e shows the results of a four-pipe test, drawing to four times the number of pixels with negligible performance drop, indicating only that this benchmark was not limited by the fill rate.

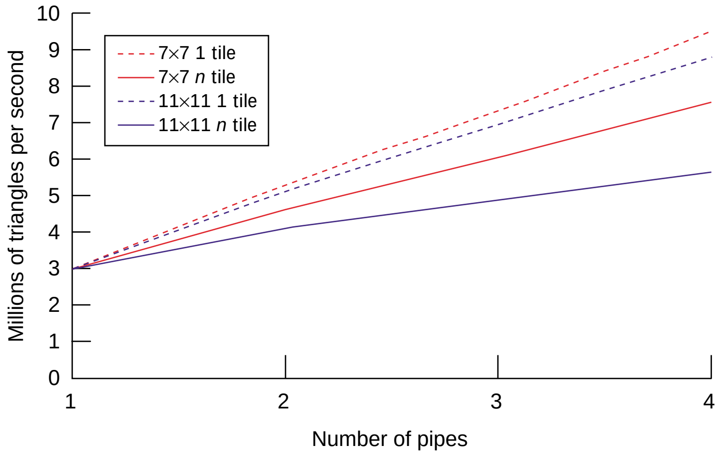
Surface optimization

The initial benchmark in Figure 3 demonstrated that operating at the highest level of abstraction doesn't impose a performance penalty. However, it also doesn't aggregate the capabilities of the multiple graphics pipes that make up the display wall. Taking the viewing frustum of each graphics pipe into account, it's possible to cull a conservative subset of the triangles that aren't visible on an individual pipe. This reduces both the amount of data and processing (transform, clipping, and rasterization) on each pipe. To make frustum culling more effective, we decomposed a test surface using regular spatial bins (see Figure 4).

Scaling results for two large surfaces appear in Figures 5 and 6. Results in Figure 5 are for an isosurface of approximately 2 million triangles rendered on one to four pipes. We compared results for the partitioned triangle meshes with the performance achieved without partitioning. We performed two tests for each pipe configuration. The one-pipe test actually uses only one of the *n* pipes. Because our surfaces are roughly evenly distributed between the graphics pipes, this tests the efficiency and overhead of the culling approach, giving an upper bound for the second test, the *n*-pipe test. This test uses all *n* pipes, measuring the actual parallel rendering performance of the display wall system. We held the pixel resolution of the images constant at 1,020 × 1,020 to eliminate pixel fill-rate influences on performance. Results in Figure 6 are for a larger surface of 8 million triangles, also rendered on 1 to 4 pipes.



5 **Performance tests for an isosurface consisting of 2 million triangles. We subdivided the surface into 25 and 49 clusters for improved culling performance.**



6 **Performance tests for an isosurface consisting of 8 million triangles. We subdivided the surface into 49 and 121 clusters for improved culling performance.**

We first observe from the results that overall performance degrades as pipes are added to the display wall if the surface is not segmented. This resembles the immediate mode benchmark presented in the previous section. When the surface is segmented into clusters, performance improves as more pipes are added to the display wall because the load on each pipe is reduced.

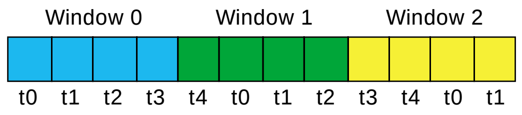
We are currently using axis-aligned bounding boxes to determine whether a particular cluster is potentially visible on a given pipe. Note that as the number of clusters increases, performance improves further. Initial results look promising, and we hope to further improve results through data-dependent clustering and improved bounding regions.

Applications: Movie player

A necessary, fundamental capability for large display walls is the ability to display precomputed animations at interactive frame rates. Despite the simplicity of the problem description, the data demands for large display walls (6400 × 3072 resolution) are daunting. The software design must balance disk storage, I/O bandwidth, graphics bandwidth, and processing requirements to provide predictable and accurate frame rates. A static segmentation of the animation isn't acceptable because of our desire to support dynamic pan-and-zoom capability for large movie frames across the entire display.

To reduce the storage and I/O demands, we incorporated both standard and customized movie file formats with a variety of compression techniques. Both I/O and decompression take place in parallel for increased image-streaming performance. The movie player is object-oriented in design, using C++ to isolate the thread management and synchronization logic in a base class. Extensions to this base class define the method for reading a frame from the animation, allowing for support of multiple formats. VDL is leveraged for the flexible display configurations and threaded display to multiple pipes.

The first challenge lies in getting frames from disk into memory as quickly as possible. We use a collection of k worker threads, with thread i responsible for frames $i, 2i, \ldots, ni$. Each thread maintains a local cache of three frames, allowing for efficient streaming in both forward and reverse directions. For common movie formats such as MPEG and QuickTime we use standard libraries to read frames from disk. This leads to a trade-off between a large number of threads for parallel decompression and a small number of I/O requests to avoid thrashing of the disk. To address this issue, we developed a simple storage format that reduces the number of I/O



7 Aggregating consecutive frames into "windows" formats animations for high-performance I/O. If the individual frames are compressed, a collection of threads performs decompression in parallel to achieve high frame rates. Note that parallel I/O results from setting the number of threads to be greater than the window size.

requests by reading multiple compressed frames at a time.

Figure 7 shows an example scenario with five decompression threads and a window length of four. The first thread to make a request for a frame in a particular window will perform a read for the entire window of compressed frames. Additional threads requiring access to the same window will block until the read is completed, after which all threads will proceed to decompress the frames in parallel. This approach has proven effective for improving scalability to larger image resolutions while maintaining high frame rates.

Performance of the I/O and decompression infrastructure appears in Figure 8. We generated a movie at a resolution of 2560 × 2048 pixels and filtered it to create a smaller version at 1280 × 1024 pixels. The frames are compressed with a variant of run-length encoding (RLE) and stored on a Fiber Channel disk array. We performed tests with a varying number of worker threads for decompression. Each test case had three I/O windows, each holding eight frames of the animation. These results demonstrate scalability for streaming of high-resolution animations on a large shared-memory system.

Once frames are decompressed in memory, the second bandwidth-limited operation involves sending the appropriate pixels to each graphics pipe. We have experimented with both pixel-transfer and texture-load operations for moving images to a graphics pipe, with similar results in each case. Culling is equally critical for 2D images as for 3D surfaces. When performing a texture load or image blit for a particular graphics pipe, the first step is to determine the smallest set of pixels required to correctly render the given view. Depending on the interpolation used, this region may extend slightly outside the viewing frustum. Given this image size, a texture of the next highest power-of-two resolution is defined. Through the use of subimage loading, the system sends a minimal amount of data to each graphics pipe. With the player's interactive pan-and-zoom capability, this tiling is performed dynamically for each frame of the animation.

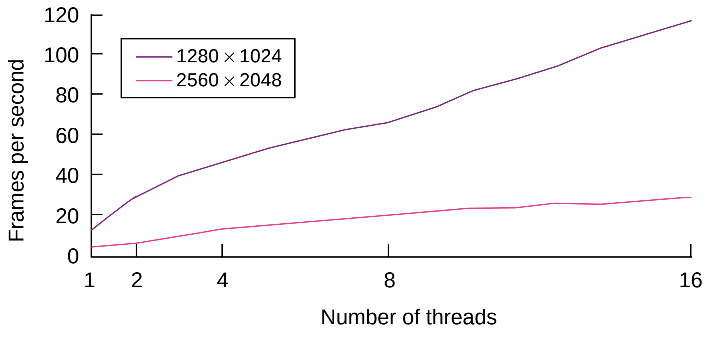
Applications: MeshTV

One goal of VDL was to provide a means for existing console applications to use the display wall. MeshTV is a scalable, parallel visualization tool developed at LLNL (http://www.llnl.gov/meshtv/). Three tasks were involved in enhancing MeshTV to render on the display wall. First we had to revise MeshTV's underlying graphics library to support the VDL interface requirements. Next we added controls to the MeshTV engine and implemented a new control window in the graphical user interface (GUI), allowing users to dynamically position graphics windows over the display wall. Finally, we modified the parallel version of MeshTV, which uses massively parallel software-rendering techniques, to leverage VDL for hardware-accelerated parallel rendering.

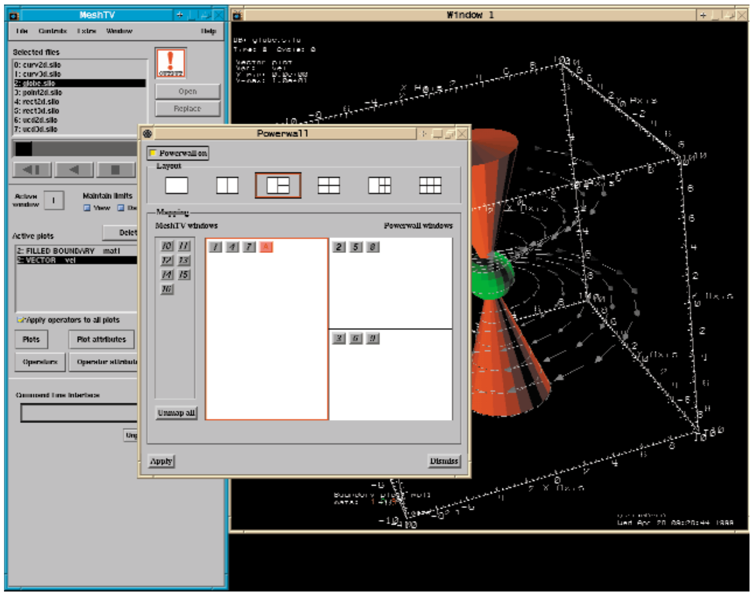
The bulk of the work in the first task involved ensuring thread safety in the graphics MeshTV driver interface, as required for parallel rendering by VDL. We derived a VDL-specific device driver from the existing OpenGL driver to utilize the VDL viewing transform and viewport manipulation functions.

The second task involved adding display wall awareness to MeshTV. The first part of this phase required modifying the MeshTV engine to route rendering commands targeting a visualization window to a display wall. This change allowed any action performed in a visualization window to echo to a display wall. In addition, we implemented new commands, letting users control how MeshTV uses a display wall. A control window for the display wall added to the GUI lets users specify the number of visualizations to appear on the display wall and their arrangement. A layout editor splits the wall into smaller areas, allowing multiple visualization windows to show simultaneously on the same display wall. The control window lets users intuitively map visualization windows via the Motif drag-and-drop mechanism (see Figure 9).

The third development task involved modifying the parallel version of MeshTV to use a display wall. Parallel MeshTV works by splitting parts of a problem among



8 Streaming performance for compressed movies stored on disk. At larger resolutions the system bandwidth limits scalability.



9 **Graphical drag-and-drop user interface for controlling layout of the display wall. The user controls the layout from a traditional console display.**

separate processes in a distributed memory environment. Each process creates its own part of the problem geometry and uses software rendering to generate an image. The images from each process are composited into a single image via the message passing interface (MPI) library and displayed in a visualization window by the master process. This sort-last approach to parallel rendering couldn't exploit the display wall's multiple graphics engines—natural candidates for a sort-first rendering approach.

To counter this, we developed a distributed parallel graphics driver. This driver transforms geometry from each distributed process into a display list representation that is communicated to the master process using MPI. This parallel graphics driver collects and buffers opcodes and data that represent the OpenGL functions that each process would have generated. The driver interprets the collective opcode/data stream and executes the appropriate OpenGL functions in multiple VDL rendering threads. Processing time decreases because the geometry is created in parallel and offers opportunities to cull the data stream to individual tiles in parallel, potentially reducing the number of primitives submitted to each graphics context.

Future directions

We continue to work in a number of areas to expand and improve on our present capabilities. A major effort now underway will incorporate view-dependent visual hierarchies into our scalable graphics system. This is a critical effort, as the data set sizes continue to increase and the current level of interaction won't suffice for head-tracked immersive displays.

With our recent addition of a FakeSpace VersaBench passive stereo display system, we have considered how to leverage our existing VDL tools in the immersive space. We plan high-level interfaces to devices such as gloves and wands for the interactive layers. We're actively pursuing natural gestural interaction interfaces, leveraging techniques from computer vision, optimization, and natural language research.

As the performance of personal-computer graphics cards improves, we will consider driving these visualization displays from clusters of coupled PCs. We hope to leverage such scalable clusters to drive these displays at even higher rates, potentially by tiling to an even greater extent and using a balance of sort-first and sort-last techniques.

Finally, we must address the issue of remote visualization. The technology is rapidly reaching the point where it's feasible to place large numbers of LCD flat-panel displays in a single office. This office could be across campus or across the country. Issues of graphics resource allocation, utilization, and image/data transport play a key role in our deploying these tools into the hands of end users. The advent of digital video and new real-time compressed digital image transport over gigabit Ethernet will open the door to a number of new forms of office visualization delivery.

Conclusions

Meeting the demands of the current terascale visualization community requires the use of large- and small-scale tiled displays. The LLNL efforts in this space have lead to the creation of one of the largest interactive tiled displays built to date and a number of new efforts for "personal" tiled displays.

We outlined the system implemented at LLNL for the creation and support of large, tiled, multipipe graphics displays. The system builds on a simple, portable, parallel API for tiling OpenGL commands in parallel to multiple contexts on a set of X servers. This API has allowed us to retrofit existing visualization and analysis codes to support these displays with very little effort. While this work represents a first step toward the ultimate goal of scalable visualization in individual office spaces, the result has been the creation of useful new visualization workspaces and tools for our users. ■

Acknowledgments

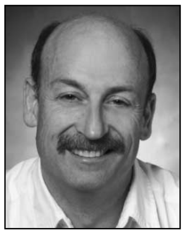
We would like to thank to a number of people who contributed to this project, including Sean Ahern, Eric Brugger, Mark Duchaineau, Dieter Fuss, Steve Louis, Kim Minuzzo, Lisa Roberts, Becky Springmeyer, Sam Uselton, and Dave Wiltzius at LLNL, and Tom Ruwart and Paul Woodward of the University of Minnesota.

This work was performed at Lawrence Livermore National Laboratory (LLNL) under contract W-7405-ENG-48 for the US Department of Energy (DOE). It was supported in part by the Accelerated Strategic Computing Initiative and the Visual Interactive Environment for Weapons Simulations (VIEWS) program.

References

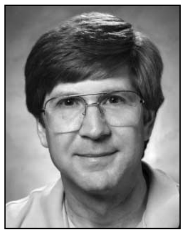
1. P.D. Heerman, "First-Generation ASCI Production Visualization Environment," *IEEE Computer Graphics and Applications*, Vol. 19, No. 5, 1999, pp. 66-71.
2. A.A. Mirin, "Performance of Large-Scale Scientific Applications on the IBM ASCI Blue-Pacific System," *Proc. Ninth SIAM Conf. of Parallel Processing for Scientific Computing*, SIAM, Philadelphia, Mar. 1999, CD-ROM.
3. C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti, "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE," *Proc. ACM Siggraph 1993*, Ann. Conf. Series, ACM Press, New York, pp. 135-142.
4. C. Just et al., "VRJuggler: A Framework for Virtual Reality Development," Immersive Projection Technology Workshop, March 1998, http://www.vrac.lastate.edu/ipt98.
5. Fakespace Systems, "VLIB: API and Software Tools for Fakespace Devices and Display Systems," Feb. 1999, http://www.fakespacesystems.com/products/vlib.html.
6. G. Humphreys and P. Hanrahan, "A Distributed Graphics System for Large Tiled Displays," *Proc. IEEE Visualization 1999*, ACM Press, New York, pp. 215-223.
7. R. Samanta at al., "Load Balancing for Multi-Projector Rendering Systems," *Proc. Siggraph Workshop on Graphics Hardware* (Eurographics 99), ACM Press, New York, Aug. 1999, pp. 107-116.
8. University of Minnesota, "The Powerwall," http://www.lcse.umn.edu/research/powerwall/powerwall.html.
9. J. Rohlf and J. Helman, "Iris Performer: A High-Performance Multiprocessing Toolkit for Real-Time 3D Graphics," *Proc. ACM Siggraph 1994*, Ann. Conf. Series, ACM Press, New York, pp. 381-395.

Daniel R. Schikore *is a senior developer at Computational Engineering International (CEI). He received his PhD from Purdue University in 1997. His interests include feature extraction, surface extraction, multiresolution data structures and algorithms, and rendering. He is a member of the IEEE and ACM.*

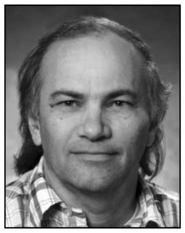


Richard A. Fischer *is a System Administrator at Lawrence Livermore National Laboratory working on the Visual Environment for Weapons Simulation Visualization project. His interests include high-performance I/O, storage, and graphics systems.*

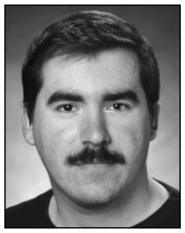
Randall Frank *is a computer scientist at Lawrence Livermore National Laboratory working on the Visual Environment for Weapons Simulation Visualization project. He received BS (1988) and MS (1996) degrees in biomedical engineering from the University of Iowa. His interests include scientific visualization, rendering, and medical imaging. He is a member of the IEEE and ACM.*



Ross Gaunt *is a computer scientist at Lawrence Livermore National Laboratory working in the Visual Environment for Weapons Simulation Visualization project. He has a BS in computer science and mathematics. His interests include visualization and digital and analog video.*



John Hobson *is a computer scientist at Lawrence Livermore National Laboratory working in the Visual Environment for Weapons Simulation Visualization Project. He received a BA in mathematics from the University of California at Santa Barbara and an MA in mathematics from the University of California at Berkeley. His interests include object-oriented software design, computer graphics, and visualization.*



Brad Whitlock *is a computer scientist at Lawrence Livermore National Laboratory working in the Visual Environment for Weapons Simulation Visualization project. He has a BS in computer science from California State University, Sacramento. His interests include visualization, parallel programming, and graphical user interface design.*

Readers may contact Schikore at Computational Engineering International, 600 Airport Blvd., Suite 500, Morrisville, NC 27560, e-mail schikore@ceintl.com. Contact Frank at Lawrence Livermore National Lab, PO Box 808, L-561, Livermore, CA 94550, e-mail rjfrank@llnl.gov.