Scalable Rendering on PC Clusters

Brian Wylie, Constantine Pavlakos, Vasily Lewis, and Ken Moreland
Sandia National Laboratories

Sandia National Laboratories use PC clusters and commodity graphics cards to achieve higher rendering performance on extreme data sets.

Today's PC-based graphics accelerators achieve better performance—both in cost and in speed. A cluster of PC nodes where many or all of the nodes have 3D hardware accelerators is an attractive approach to building a scalable graphics system. We can also use this approach to drive a variety of display technologies—ranging from a single workstation monitor to multiple projectors arranged in a tiled configuration—resulting in a wall-sized ultra high-resolution display. The main obstacle in using cluster-based graphics systems is the difficulty in realizing the full aggregate performance of all the individual graphics accelerators, particularly for very large data sets that exceed the capacity and performance characteristics of any one single node.

Based on our efforts to achieve higher performance, we present results from a parallel sort-last implementation that the scalable rendering project at Sandia National Laboratories generated. Our sort-last library (libpglc) can be linked to an existing parallel application to achieve high rendering rates. We ran performance tests on a 64-node PC cluster populated with commodity graphics cards. Applications using libpglc have demonstrated rendering performance of 300 million polygons per second—approximately two orders of magnitude greater than the performance on an SGI Infinite Reality system for similar applications.

Achieving performance

The Department of Energy's Accelerated Strategic Computing Initiative (DOE ASCI) is producing computations of a scale and complexity that are unprecedented.[1,2] High-fidelity simulations, at high spatial and temporal resolution, are necessary to achieve confidence in simulation results. The ability to visualize the enormous data sets produced by such simulations is beyond the current capabilities of a single-pipe graphics machine. Parallel techniques must be applied to achieve interactive rendering of data sets greater than several million polygons. Highly scalable techniques will be necessary to address projected rendering performance targets, which are as high as 20 billion polygons per second in 2004[2] (see Table 1).

ASCI's Visual Interactive Environment for Weapons Simulations (Views) program is exploring a breadth of approaches for effective visualization of large data. Some approaches use multiresolution and/or data reduction to simplify data to enable real-time viewing and/or a better match between data complexity and target display (for example, the number of pixels). Such approaches generally require special preparation of the data prior to rendering, which can be computationally intensive. While such approaches prove useful, they also can raise concerns regarding the possible loss of information as data becomes simplified. An alternative is to increase the raw power of our rendering systems, eliminating the need for special data preparation. In fact, Views is exploring the combination of these approaches to address the large data visualization problem.

To develop scalable rendering technology, Views has

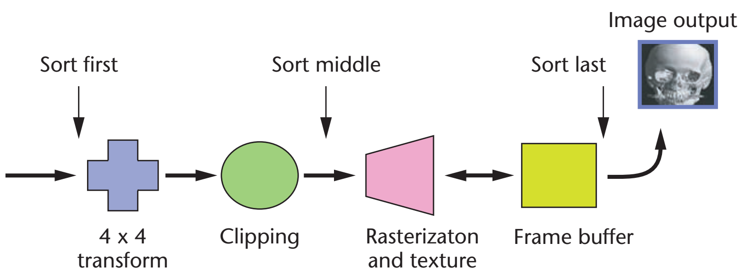
Table 1. The ASCI/Visual Environment for Weapons Simulation (Views) visualization needs.

	Today's High-end Technology	**2004 Needs**
Surface Rendering	About 2.5 million polygons per second per SGI/Infinite Reality pipe	20 billion polygons per second (aggregate)
Volume Rendering	About .1 Gpixel per SGI/Infinite reality raster manager	200 Gpixels (aggregate)
Display Resolution	16 Mpixels	64 Mpixels

0272-1716/01/$10.00 © 2001 IEEE

undertaken a collaborative effort, involving the three DOE ASCI laboratories (Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratories), university partners (including Stanford University and Princeton University), and various industrial partners, to explore the use of cluster-based rendering systems to address these extreme data sets. The intent is to leverage widely available commodity graphics cards and workstations in lieu of traditional, expensive, specialized graphics systems. While the scope of this overall scalable rendering effort is broad, this article emphasizes work and early results for polygonal rendering at Sandia National Laboratories (SNL).

A variety of approaches can be taken, and are being considered, for rendering polygonal data in parallel. Molnar[3] first proposed a classification scheme of parallel rendering algorithms based on the order of transformation, rasterization, and distribution of the polygons. Molnar's taxonomy of rendering algorithms consists of three categories: sort first, sort middle, and sort last (see Figure 1). Currently, our visualization group at Sandia is working on libraries in the sort-last category (See "Related Work" sidebar for other research on this topic).



1 Polygon rendering pipeline showing the three sorting based classifications.

Related Work

A substantial amount of work preexists in this area, especially with regard to software implementations on parallel computers.[1-5] As with our work, these efforts have been largely motivated by visualization of big data, with an emphasis on demonstrating scalability across significant numbers of computer processors. However, these software-based efforts have yielded relatively modest raw performance results when compared to hardware rendering rates.

Others have designed highly specialized parallel graphics hardware—such as the PixelFlow system[6]—that scales and is capable of delivering extensive raw performance, but such systems haven't been commercially viable. At the same time, certain architectural features of these systems may be realizable on clustered-graphics machines, especially as interconnect performance rises.

The desire to drive large, high-resolution tiled displays has recently become an additional motivation for building parallel rendering systems. ASCI partners, including Princeton University[7] and Stanford University[8,9]—as well as the ASCI labs themselves[10]—are pursuing the implementation of such systems. Both Stanford and Princeton implemented a scalable display system using PC-based graphics clusters.

Efforts to harness the aggregate power of such commodity-based clusters for more general-purpose scalable, high-performance graphics are now also underway. One such effort proposes the use of special compositing hardware to reduce system latencies and accelerate image throughput.[11]

References

1. S. Whitman, "A Task Adaptive Parallel Graphics Renderer," *1993 Parallel Rendering Symp. Proc.*, IEEE CS Press, Los Alamitos, Calif., Oct. 1993, pp. 27-34.
2. T.W. Crockett and T. Orloff, "A MIMD Rendering Algorithm for Distributed Memory Architectures," *1993 Parallel Rendering Symp. Proc.*, IEEE CS Press, Los Alamitos, Calif., Oct. 1993, pp. 35-42.
3. T. Lee et al., "Image Composition Methods for Sort—Last Polygon Rendering on 2-D Mesh Architectures," *1995 Parallel Rendering Symp. Proc.*, IEEE CS Press, Los Alamitos, Calif., Oct. 1995, pp 55-62.
4. S. Whitman, "A Load Balanced SIMD Polygon Renderer," *1995 Parallel Rendering Symp. Proc.*, IEEE CS Press, Los Alamitos, Calif., Oct. 1995, pp. 63-69.
5. T. Mitra and T. Chiueh, *Implementation and Evaluation of the Parallel Mesa Library*, tech. report, State Univ. of New York, Stonybrook, N.Y., 1998.
6. S. Molnar et al., "PixelFlow: High-Speed Rendering Using Image Composition," *Proc. Siggraph '92*, ACM Press, New York, July 1992, pp. 231-240.
7. R. Samanta et al., "Load Balancing for Multi-Projector Rendering Systems," *Proc. Siggraph/Eurographics Workshop on Graphics Hardware*, ACM Press, New York, Aug. 1999, pp. 107-116.
8. G. Humphreys, and P. Hanrahan, "A Distributed Graphics System for Large Tiled Displays," *Proc. Visualization*, ACM Press, New York, Oct. 1999, pp. 215-227.
9. G. Humphreys et al., "Distributed Rendering for Scalable Displays," *Proc. IEEE/ACM Supercomputing Conf.* (SC 2000), CD-ROM, ACM Press, New York, Nov. 2000.
10. D. Schikore et al., "High-Resolution Multiprojector Display Walls and Applications," *IEEE Computer Graphics and Applications*, vol. 20, no. 4, July/Aug. 2000, pp. 38-44.
11. A. Heirich and L. Moll, "Scalable Distributed Visualization Using Off-the-Shelf Components," *1999 IEEE Parallel Visualization and Graphics Symp. Proc.*, IEEE CS Press, Los Alamitos, Calif., Oct. 1999, pp. 55-59.

PC clusters

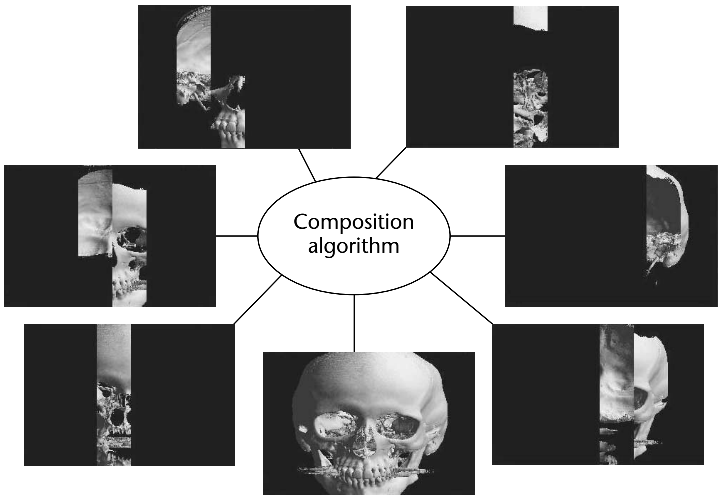
Many institutions in the past few years have built a wide variety of clusters; people use these clusters for various tasks, including database manipulation, computation, and of course, parallel rendering.

Researchers at Sandia performed their earliest efforts on a cluster of 16 SGI 320s known as *Horizon*. The cluster's interconnect is a Gigabit Ethernet comprised of Alteon network interface cards (NICs) and a Foundry Big Iron 8000 switch. Horizon's nodes each have one Intel 450-MHz Pentium III processor with 512 Mbytes RAM, running the Windows 2000 operating system. The SGI 320s use the Cobalt graphics chip set, and a unified memory architecture (UMA) that's unique for a PC product (see http://www.sgi.com/Products/PDF/1352.pdf). The total cost for Horizon at the time of pro-

2 Ric in SNL's computation annex.

and racks; approximately $150,000 for switches and network interface cards).

High-fidelity simulations, of course, require high-fidelity displays. Sandia is currently constructing a second generation Views data visualization corridor that includes a 4 × 4, 16-tile display, expandable up to 48 tiles (12 × 4). The tiled display(s) will use rear projection and be coupled with the new graphics cluster.



3 Parallel composition with libpglc, a sort-last library for parallel OpenGL composition.

Sort-last approach

Our sort-last approach statically distributes the 3D model data only once before any viewing transformations or rendering occur. For each new viewing transformation, every node performs the transformations and rasterization of its subset of triangles. A user-specified algorithm reads the frame buffer and the corresponding *z*-buffer data from each node and performs a pixel-by-pixel *z*-buffer comparison in parallel according to a user-specified composition algorithm. The composition algorithm gathers the final composited image onto a single node (Figure 3). Because the single node now has the image in memory, it isn't tied to displaying the image locally. It can, if desired, instead send the data to an alternate display.

Libpglc, a sort-last library for parallel OpenGL composition, originates from the Parallel Mesa library[4] software, developed jointly by the State University of New York (SUNY) and Sandia. The goal of libpglc is to provide arbitrary composition operations (*z*-buffer compare, XOR, and so on) through arbitrary patterns of communication (tree, binary swap, and so on) among nodes in a parallel environment. To this end, libpglc includes an implementation of the communication algorithms found in Parallel Mesa but at the same time abandons any dependencies on Mesa and will work with any OpenGL compliant environment. This allows for exploiting OpenGL hardware acceleration where available. We wrote libpglc in C and it uses the message passing interface (MPI) for interprocess communication. It runs under X11 (usually Unix) and Win 32 windowing environments. Minor modifications would be needed to run on any other OpenGL-capable environment that has an MPI implementation. It's possible to link libpglc into an existing parallel application and the application program interface (API) is extremely simple. The API seen below are also shown in context in the example C code that follows.

curement was approximately $190,000.

More recently, Sandia constructed a 64-node graphics cluster known as Ric (Figure 2), comprised of Compaq 750 nodes. Sandia equipped each of these nodes with a Pentium III 800-MHz processor, 512 Mbytes of main memory, and 4X accelerated graphics port (AGP) versions of Nvidia's Geforce 256 graphics chip set. The graphics cluster is actually part of a larger 136-node cluster known as Vicky, which includes data manipulation nodes used for supporting the complete data analysis and visualization process. Switched fast Ethernet interfaces on each node in Vicky provide an administrative and general purpose TCP/IP channel to the cluster. The message passing and parallel communications interconnect between the nodes consists of a high-speed system area network (SAN). As of spring 2001, this network has Compaq/Tandem's Servernet II Virtual Interface Architecture (VIA)-based hardware. We'll also investigate other emerging scalable SAN technology on Vicky. The total cost for the 64-node graphics cluster is approximately $500,000 (approximately $350,000 for nodes

```
int  pglc_init(int width,int height);
unsigned char*  pglc_flush(int
     display_type, int com_type);
int  pglc_finalize (void);

#include <pglc.h>
 void main(int argc, char *argv[])
 {
 pglc_init(1024, 768);

 while(unfinished) {
     Computation
            .
            .
            .
     OpenGL Calls
            .
            .
            .
     pglc_flush(COMPRESSED_TREE);
 }
 pglc_finalize( );
 }
```

With these three API functions, libpglc lets the calling application create its own rendering context and enable any platform-specific extensions. This gives the application control over the actual rendering process while leaving libpglc free to work with the rendered object. This provides flexibility to the application, letting it take advantage of hardware-specific optimizations, such as triangle-strip display lists for SGI Infinite Reality pipes, or compiled vertex arrays for Nvidia Geforce. The only argument to pglc_flush() specifies which compositing algorithm to use. We currently provide both the binary-tree and binary-swap compositing methods.[5] Hooks provided in the source of libpglc let users implement custom compositing algorithms beyond those supplied with libpglc. This allows for algorithms that match local network topologies and the performance characteristics of unique environments beyond those anticipated with the libpglc distribution. Each call to pglc_flush() performs the specified parallel composition and then returns a memory pointer to the address of the fully composited frame buffer on the root node. The frame buffer can then display locally or pass to an arbitrary, alternate display system.

Data decomposition

A benefit of the sort-last approach is that we can use relatively simple strategies to partition data for parallel rendering. The application can distribute an arbitrary subset at initialization and, as long as the model data remains unchanged, no additional retransmitting or repartitioning is necessary. As the size of the data set grows, the ability to do static decomposition becomes increasingly important. The application can assign each rendering node an arbitrary set of T/N triangles where T is the total number of triangles and N is the number of rendering nodes. In practice, this scheme demonstrates excellent load-balancing characteristics; as the number of polygons increase, the rasterization imbalances tend to become relatively small.

A drawback of sort-last is in the handling of transparency. To do proper transparency, polygons must render and/or accumulate in precise order for each pixel in the final image. In the case of sort-last across multiple hardware-rendering pipelines, the model data would need to partition spatially in a manner that lets the resulting partial image contributions, once they've been rendered properly, blend together in a predetermined order that's view dependent.

Note that libpglc doesn't impose any particular data decomposition on the application. The application can freely partition data as it chooses. At the same time, our current implementation doesn't support transparency.

Optimizations

The three principal operations associated with the sort-last approach are color and depth read-back, color and depth communication, and z-buffer compositing. Our team looked into the three principal operations to determine what optimizations could be made on each step.

Frame read-back. The rate at which libpglc retrieves image data depends on the acceleration paths provided by the graphics card manufacturer. For our tests on the Elsa Gloria 2 (Geforce) cards, the read-back requires 17 ms for RGBA and 17 ms for the depth buffer. The only optimization for read-back is the avoidance of unnecessary format conversions; for instance, reading back the color channels in float format gives a 30X performance decrease. Below we show the fast path formats for the OpenGL glReadPixels() function.

```
Color     GL_BGRA_EXT
                 GL_UNSIGNED_BYTE
24-depth  GL_DEPTH_COMPONENT
                 GL_UNSIGNED_INT
16-depth  GL_DEPTH_COMPONENT
                 GL_UNSIGNED_SHORT
```

Network communication. The next optimization focus was on the communication phase. All of our tests involved 1024×768 images. At this resolution the color and depth information is 4.5 Mbytes for each image. For a 64-node run this translates into 288 Mbytes of image data that must be transferred across the network. To reduce the size of the image data, we first applied a run-length encoding (RLE) algorithm[6] to both the color and depth buffers before transmitting. On further investigation we realized that the RLE was compressing the background and not much else. The complexity of the images were such that most active pixels had run lengths of one, and storing all the run lengths was inefficient in the final stages of composition where almost all pixels were active. Our solution was to replace the RLE with an active pixel encoded (APE) data structure. The data structure stores indexes and run lengths for active pixel sequences. The active pixels are then stored in native form with no additional overhead. This approach follows the SL-sparse model described by Molnar3, effectively compresses out any inactive pixels, and has minimal overhead (8 bytes for a fully active image). The amount of compression with APE varies, based on the data distribution and viewing parameters. However, because each node only renders a small fraction of the data (approximately 1.5 percent on a 64-node run) a 10 to 15 times reduction is common for initial stages in the composition.

The APE format also greatly reduces the time to compress and decompress images. In initial testing, the compress–decompress overhead of RLE actually nullified the network savings of sending less data. As part of our conversion to APE, we no longer adhered to traditional strategies used by RLE. Instead of splitting the RGBA channels and compressing separately, we left them interleaved and treated each 4-tuple as a single 4-byte quantity. The four-component aggregation means the algorithm requires no format conversions, performs four times fewer operations, and word-aligns all data. The current compression time for color and depth on a 1024×768 image is about 10 ms.

Image composition. Combining the rendered images requires a depth comparison and color assignment for each pixel. The composition operation requires 786,000 comparison operations for a 1024×768 image. For a 64-node run, libplgc must perform 50 million comparisons.

Generally the application compresses the color and z-buffers while sending them to the receiving node, then it decompresses and composites the images. The sequence of decompression followed by composition is wasteful in two aspects. The decompression involves expanding a low-redundancy format into a high-redundancy format. The composition algorithm, which could have taken advantage of the run length information, now performs more operations to traverse the expanded data. A solution is to run the compositing algorithm directly on the incoming compressed data and skip the decompression altogether.

To efficiently perform composition on the APE image, we combined the RGBA and z channels into the same compressed data structure. Any active pixels will modify both the color and depth channels so that aggregating the two channels doesn't affect compression. In addition, the overhead of storing indexes and run lengths will now amortize over both channels.

The algorithm to compose the incoming image with the existing local image is straightforward. The incoming APE image is merged into a local uncompressed image. As active pixel segments are processed from the incoming APE image, the following actions are taken:

1. Any nonoverlapping active pixel segments are simply block copied into the uncompressed destination image.
2. Any overlapping segments are handled in the normal manner. Each of the incoming pixels is depth-compared to the existing image and then merged accordingly.

The APE composition algorithm reduces the number of memory references in direct proportion to the compression ratio. It also reduces the number of depth comparisons to only those pixels that are active in both images. The merging of APE images also means that the compression step is now part of the composition. Our experiments have shown that composition using APE images provides a large performance benefit: it eliminates the decompression time—the composition is, on average, 50 percent faster—and it also eliminates recompression. Composition time for color and depth on a 1024×768 image is about 12 ms.

Combined results. The following list categorizes the main optimizations applied to our sort-last library:

- *Initial RLE image compression*. This greatly reduces the network bandwidth at the cost of increased CPU time for data reformatting and operating on off-word boundaries.
- *Color component aggregation and conversion to APE*. These dramatically minimize CPU time by leaving data in native format, working only on word boundaries and reducing the number of operations by a factor of 4. The use of component aggregation significantly minimizes the overhead associated with dense images.
- *Composition directly on compressed data*. This improved composition time by an average of 50 percent and completely eliminated the decompression for both the color and depth channels.

After combining the above optimizations, the performance benefits are substantial; we reduce compositing overheads associated with our sort-last library by 64 percent (500 ms to 180 ms for 64 nodes at an image resolution of 1024×768).

Performance evaluation

We gathered all of our performance results from runs using our 64-node cluster. Each of our Compaq 750 rendering nodes contains one 800-MHz Intel Pentium III. Our current interconnect is ServerNet II with a peak throughput of 95 Mbps from point-to-point using VIA protocols. In tests using the MPIPro software, our rates drop to 68 Mbps over MPI. In practice, the single point-to-point rate is extremely hard to achieve when running a large job and can vary based on communication topology, message sizes, and switch traffic. The graphics performance of the nodes, using vertex arrays in OpenGL, is about 5.5 million triangles per second on our applications.

Although our compositing optimizations are independent of rendering performance, we feel a brief discussion of rendering issues is informative. Several factors affected our serial rendering performance and could increase performance by addressing the following issues.

- The large sizes of our vertex arrays excluded the use of locked vertex array extensions such as wglAllocateMemoryNV.
- The isosurfaces used for testing gave relatively short triangle strips (strips 5 to 6 triangles long were common).
- The noncontiguous indices lead to poor vertex cache coherency.

With the large isosurfaces and desktop image sizes, our hardware pipelines became bottlenecked in the translation; lighting stages and rasterization wasn't a contributing factor. (Note that the system even considers subpixel triangles for rasterization based on point sampling as described in the OpenGL specification.)

Sort-last library

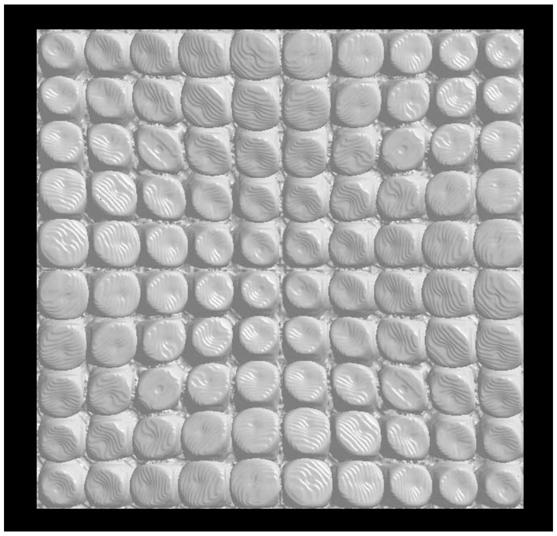
To evaluate the performance of libpglc, we wrote a parallel application that links to libpglc. The program reads in arbitrary fractions of the data set from disk and then makes function calls as demonstrated in the "Description" section. During the performance evaluations we used data provided by Lawrence Livermore National Laboratories[7] from a large turbulence simulation. The data included a range of isosurfaces, the largest being 469 million triangles (see Figures 4 and 5).

The performance numbers we obtained from an application using libpglc appear in Figures 6 through 9 (next page). Figure 6 shows that with a smaller data set, the inherent overhead associated with the image compositition starts to hinder performance as the number of processors increases. Figures 7 and 8 demonstrate that as the size of the data set increases, the fixed overhead of composition becomes relatively small compared to rendering times, and we see dramatic performance improvements. In Figure 9 the application reached an average performance of 300 million triangles per second with 64 nodes.

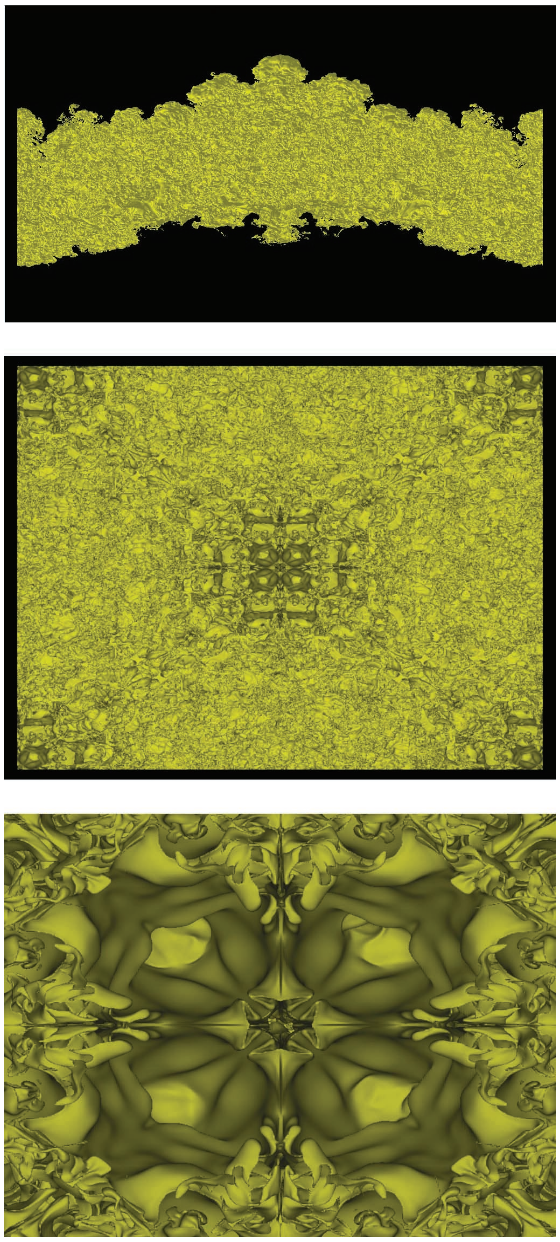
The missing data points in Figures 8 and 9 result from the fact that the larger data sets didn't fit in memory when using fewer processors. The 469 million triangle data set only ran on 32- and 64-node configurations (32 nodes required heavy use of virtual memory).

We can see in Figure 10 (on page 69) that the total compositing overhead remains basically constant and is independent of the data size. The relatively fixed overhead—around .2 second for 64 nodes—means an upper frame rate of approximately 5 Hz. Figure 11 (page 69) shows that we're getting excellent scalability as the size of the data increases. The maximum cumulative rendering rate of our 64 graphics cards is 352 million triangles per second (5.5 million triangles per second for each card). Our application achieves 90 percent utilization of the total aggregate performance when running on 32 nodes with 235 million triangles and 85 percent utilization when running on 64 nodes with 469 million triangles. For extremely large data sets, the use of a sort-last library has the following advantages:

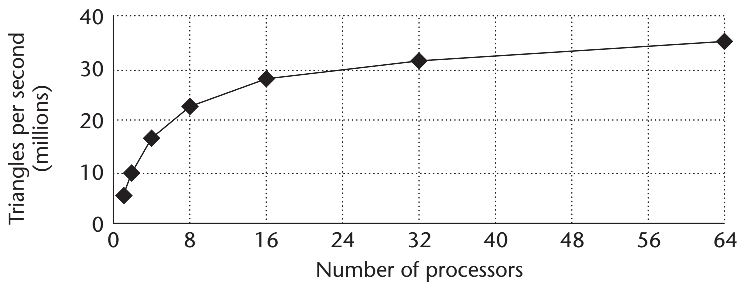
- Image composition incurs a fixed overhead that doesn't vary with data set size.
- The polygonal data doesn't have to be moved from node to node when changing viewpoints.
- Data partitioning is static and provides excellent load balancing in the rendering phase.
- Memory use is maximized. Because no dynamic memory requirements exist, as with view-dependent algorithms, the library needs no extra memory buffers. Each node can fully load all available RAM with polygonal data.



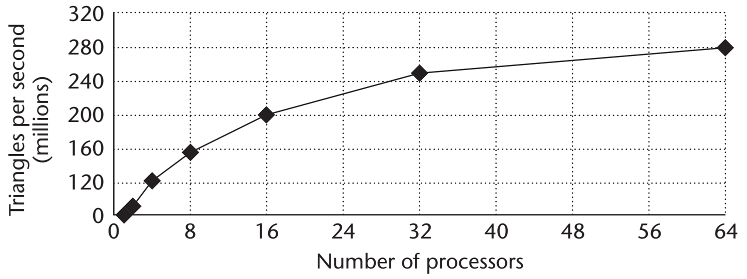
4 Lawrence Livermore isosurface data (7 million triangles).[7] Image covered by LLNL: UCRL-MI-142527.



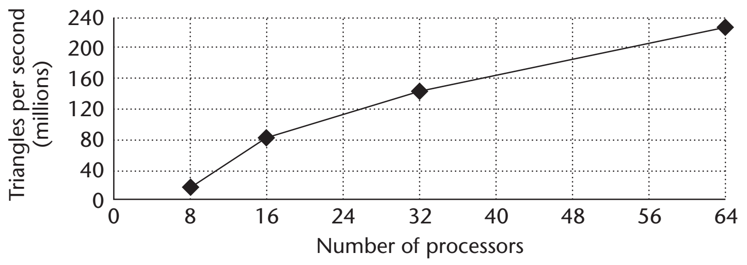
5 Various views of the Lawrence Livermore isosurface data (469 million triangles).[7] We show the center section in more detail as we zoom in (bottom image). Image covered by LLNL: UCRL-MI-142527.



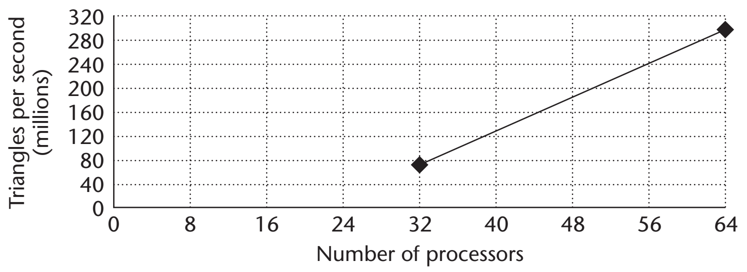
6 **Results for a 7 milion triangle data set (10 percent utilization at 64 nodes).**



7 **Results for a 30 milion triangle data set (28 percent utilization at 64 nodes).**



8 **Results for a 117 milion triangle data set (67 percent utilization at 64 nodes).**



9 **Results for a 469 milion triangle data set (85 percent utilization at 64 nodes).**

Open issues

One drawback to the sort-last approach is that, nominally, each graphics node must compute a full-size image. This becomes problematic when the intent is to drive a high-resolution display, such as a display wall. The sort-last approach has at least a couple of problems:

- commodity graphics cards can only generate images up to conventional pixel resolutions (for example, 1280 × 1024), and
- communications overhead associated with the compositing step grows with the image size.

We're considering variations to sort-last that would overcome these limitations.

While sort-first approaches prove useful for driving tiled displays, the strict association of a dedicated renderer per tile presents load-balancing issues. In the worst case, for example, the current view might be such that all of the data projects onto a single tile, in which one renderer has to do all the work. Clearly, we need some mechanism for separating the rendering function from the display. We're working with our ASCI collaborators to explore such solutions, as well as applications of sort-first.

We've shown that cluster-based systems have a lot of promise for rendering large data. But can they ultimately compete with more specialized, tightly integrated graphics systems for high frame-rate applications, such as visual simulation? This remains to be seen and is certainly tied to the performance of interconnect technologies. An overall objective, however, for such systems is to somehow ensure that the system's parallel resources can be dynamically allocated according to data size in such a way that applications experience monotonically increasing (or at least nondecreasing) performance as we apply more and more resources.

Certain pragmatic challenges also exist. Administering cluster-based systems is nontrivial, and we need mechanisms for ensuring that such systems appear to be robust enough and reliable enough to support production work. In this sense, graphics clusters are no different than other clusters. However, graphics clusters also place an additional demand on accessibility as a dynamically shared resource to support highly interactive work, presenting some unique challenges for resource management.

An interesting issue has arisen from the complexity of our largest data sets, relating to image quality, correctness, and the precision of commodity-graphics hardware, especially with regard to *z*-buffer depth. Typically, hardware-rendering algorithms use a *z* buffer. As data sets increase in size and complexity, the 24-bit precision provided by commodity graphics cards can be insufficient for global depth comparisons. We've observed this phenomenon. (To guarantee correct rendering of extremely complex data sets, we need additional techniques for ensuring exact depth sorting.)

Conclusions and future work

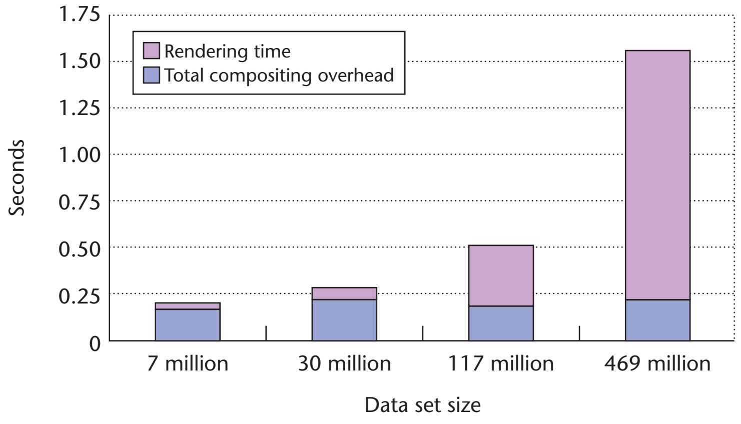
We've demonstrated the use of a sort-last architecture on a PC cluster. We've developed optimizations that have enhanced the performance of our sort-last implementation. Our work provides promising results toward the viability of graphics clusters in the area of large-scale, high-performance rendering. The efficient use of 64 commodity graphics cards enabled us to establish pace-setting rendering performance of 300 million triangles per second on extremely large data. We believe that extensions of this work will be an essential part of the technology used to address the rendering performance targets set by the ASCI program. We're also participating in an effort, together with some of our ASCI collaborators, to deliver a common OpenGL-based API for parallel rendering and an open-source reference implementation.

We expect to continue to explore the use of commodity-based graphics clusters for high-performance graphics. In so doing, we expect to investigate many, if not all, of the issues discussed in the previous section. Other work we anticipate includes

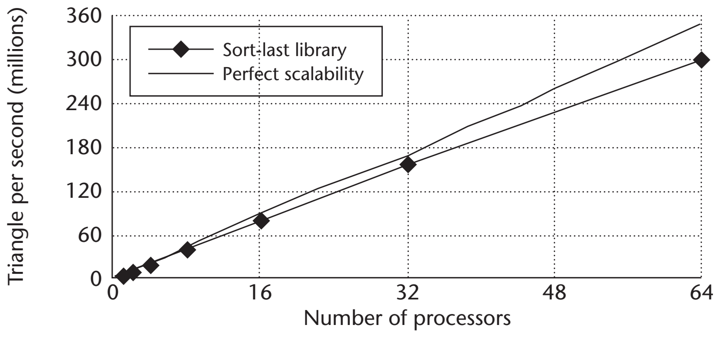
- continued optimization of our current software;
- the consideration of hybrid sorting schemes and, perhaps, other more novel architectural approaches to rendering;
- performing scalability assessments on larger graphics clusters;
- processing time-dependent data and addressing issues related to feeding data to the parallel rendering system; and
- integrating graphics clusters into our end-to-end high-performance computing environments.

We're already leveraging the approaches presented in the "Optimizations" section to accelerate the use of sort-last techniques on high-resolution, multiple-tile displays.[8] We're targeting display resolutions of 64 million pixels as early as 2002. We aim to provide techniques for rendering images on tiled displays with frame rates comparable to those for a single display.

Please visit our Web site for the latest developments and information on libpglc and other scalable rendering work at http://www.cs.sandia.gov/VIS/SR. ■



10 **A breakdown of the total time for runs on 64 nodes.**



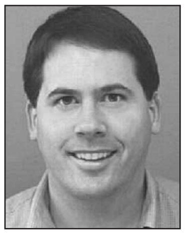
11 **Demonstration of scalability. As the data size doubles we double the number of processors. Data size ranges from 7 to 469 million triangles.**

Acknowledgments

Funding was provided by the Accelerated Strategic Computing Initiative's Visual Interactive Environment for Weapons Simulations (ASCI/Views) program. Thanks to Milt Clauser and Steve Monk for their work on the visualization and data manipulation clusters. Thanks to LLNL for the large isosurface data (particularly Randy Frank and Dan Schikore, now with CEI). Thanks to Pat Crossno for her vast technical library and extremely helpful suggestions, Dan Zimmerer for his research and development on cluster support, Phil Heermann for his inspiration and motivation, and Lisa Ice for her work on ParallelMesa (PMesa). The US Department of Energy supported this work under contract DE-AC04-94AL85000.

References

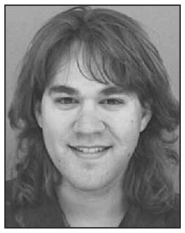
1. P. Heermann, "Production Visualization for the ASCI One TeraFLOPS Machine," *Proc. Visualization 98*, IEEE CS Press, Los Alamitos, Calif., Oct. 1998, pp. 459-462.
2. P.H. Smith and J. van Rosendale, *Data and Visualization Corridors*, tech. report, 1998 DVC Workshop Series, California Inst. of Technology, Pasadena, Calif., 1998.
3. S. Molnar et al., "A Sorting Classification of Parallel Rendering," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, July 1994, pp. 23-32.
4. T. Mitra and T. Chiueh, *Implementation and Evaluation of the Parallel Mesa Library*, tech. report, State Univ. of New York, Stonybrook, N.Y., 1998.
5. K.L. Ma et al., "Parallel Volume Rendering Using Binary-Swap Image Composition," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, July 1994, pp. 59-68.
6. J. Ahrens and J. Painter, "Efficient Sort-Last Rendering Using Compression-Based Image Compositing," *Proc. Second Eurographics Workshop Parallel Graphics and Visualization*, Univ. of Bristol, Bristol, United Kingdom, 1998, pp. 145-151.
7. A. Mirin, "Performance of Large-Scale Scientific Applications on the IBM ASCI Blue-Pacific System," *Proc. Ninth SIAM Conf. Parallel Processing for Scientific Computing*, CD-ROM, Soc. for Industrial and Applied Mathematics, Philadelphia, Mar. 1999.
8. K. Moreland, B. Wylie, and C. Pavalakos, "Sort-Last Tiled Rendering for Viewing Extremely Large Data Sets on Tiled Displays," submitted to *IEEE Symp. Parallel and Large Data Visualization and Graphics*, 2001.



Brian Wylie *is a senior member of the technical staff at Sandia National Laboratories. He's currently working on the visual environment for the weapons simulation visualization project. His interests include scientific visualization, scalable rendering, and information visualization. He received a BS in computer engineering and an MS in computer science from the University of New Mexico.*

Constantine Pavlakos *is a senior member of the technical staff at Sandia National Laboratories, working in computer graphics, scientific visualization, and data analysis for approximately 20 years. He's currently Sandia's principal investigator for ASCI/Views-related research and development. His research interests include scalable visualization, large data visualization, data services, and visualization environments in support of high-performance computing. He received his BS in mathematics in 1976 and his MS degree in computer science in 1978, both from the University of New Mexico. He is a member of the ACM Siggraph. Further information about Pavlakos is available at http:/www.cs.sandia.gov/VIS/dino.html).*

Vasily Lewis *is a seasoned undergraduate working on contract for Sandia National Laboratories. He's currently working on the visual environment for the weapons simulation visualization project. His interests include distributed parallel computing, real-time photorealistic graphics, and network intrusion detection.*



Kenneth Moreland *is a member of the technical staff at Sandia National Laboratories. He's currently working on the visual environment for weapons simulation visualization project. He received BS degrees in electrical engineering and computer science from the New Mexico Institute of Mining and Technology and an MS in computer science from the University of New Mexico. His research interests include high-performance parallel rendering, desktop delivery for cluster computing, and large-scale data services.*

Readers may contact Wylie at Sandia National Laboratories, PO Box 5800, MS 0318, Albuquerque, NM 87185, email bnwylie@sandia.gov.

For further information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.