Scalable Rendering on PC Clusters

Brian Wylie, Vasily Lewis, David N Shirley*, Constantine Pavlakos
Sandia National Laboratories
Albuquerque, NM 87185

ABSTRACT

This case study presents initial results from research targeted at the development of cost-effective scalable visualization and rendering technologies. The implementations of two 3D graphics libraries based on the popular sort-last and sort-middle parallel rendering techniques are discussed. An important goal of these implementations is to provide scalable rendering capability for extremely large datasets (>> 5 million polygons). Applications can use these libraries for either run-time visualization, by linking to an existing parallel simulation, or for traditional post-processing by linking to an interactive display program. The use of parallel, hardware-accelerated rendering on commodity hardware is leveraged to achieve high performance. Current performance results show that, using our current hardware (a small 16-node cluster), we can utilize up to 85% of the aggregate graphics performance and achieve rendering rates in excess of 20 million polygons/second using OpenGL® with lighting, Gouraud shading, and individually specified triangles (not t-stripped).

1 INTRODUCTION

The Department of Energy's Accelerated Strategic Computing Initiative (ASCI) is producing computations of a scale and complexity that are unprecedented[1,2]. High fidelity simulations, at high spatial and temporal resolution, are needed to achieve the necessary confidence in simulation results. The ability to visualize the enormous datasets 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 datasets greater than several million polygons. Highly scalable techniques will be necessary to address projected rendering performance targets, which are as high as 19 billion polygons per second in 2004[2]. As part of a broader effort in ASCI's Visual Interactive Environment for Weapons Simulations (VIEWS) program, Sandia National Laboratories (SNL) is exploring the development of cluster-based rendering systems to address these extreme datasets. The intent is to leverage widely available commodity graphics cards and workstations in lieu of traditional, expensive, specialized graphics systems.

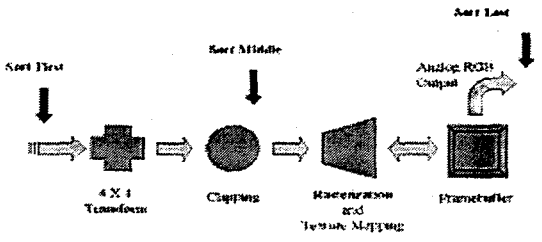


Figure 1: Polygon rendering pipeline showing the three sorting based classifications.

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 for both the Sort-middle and Sort-last categories.

2 RELATED WORK

A substantial amount of work pre-exists in this area, especially with regard to software implementations on parallel computers[4,5,6,7,8]. 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 compute processors. However, these software-based efforts have yielded relatively modest raw performance results when compared with hardware rendering rates.

Others have designed highly specialized parallel graphics hardware, such as the PixelFlow system[9], that scales and is capable of delivering extensive raw performance, but such systems have not yet proven to be commercially viable. At the same time, certain architectural features of such 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[10] and Stanford University[11], as well as the ASCI labs themselves[12], are pursuing the implementation of such systems. Princeton, in particular, has implemented a scalable display system using a PC-based graphics cluster.

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 has proposed the use of special compositing hardware to reduce system latencies and accelerate image throughput[13].

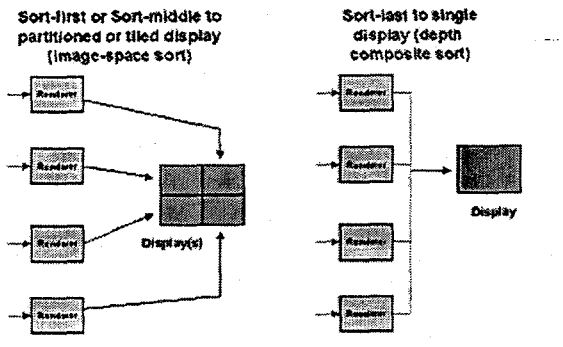


Figure 2: Parallel rendering schemes.

* ABBA Technologies

3 PLATFORM AND SOFTWARE

3.1 Cluster Specifics

Many institutions in the past few years have built a wide variety of clusters: these clusters are used for tasks including database manipulation, computation and, of course, parallel rendering. Our visualization group is currently working on a research cluster of 16 SGI 320's. The message passing interconnect used for our cluster is a Gigabit Ethernet comprised of Alteon NIC's and a Foundry 'Big Iron' 8000 switch.

The cluster nodes each have one Intel 450MHz PIII processor with 512 MB Ram. All of the machines including the fileserver are running the Windows 2000™ operating system. Each node has a Gigabit Ethernet interface for message passing and an integrated Fast Ethernet interface for I/O. The SGI 320's use the Cobalt™ graphics chipset and are currently configured with 64MB of the available 512MB of ram. Total cost of our cluster at time of procurement was approximately $190K.

The 320's, while commodity oriented, do have some special hardware, notably the usage of the Unified Memory Architecture (UMA). The advantages of using machines with UMA as the platform for parallel rendering software will be discussed in section 4.1. Although this hardware is somewhat unique, the software makes no specific assumptions about the underlying hardware and will scale equally well on other combinations of PC/graphics cards.

High fidelity simulations, of course, require high fidelity displays. Our Sort-middle library will be targeted for large multi-tile displays. As part of the construction of a 2nd Generation Visualization Corridor, Sandia National Laboratories will build a 4x4 16-tile display expandable up to 48 tiles (12x4). These tiled displays will use rear projection and be coupled to the next generation cluster hardware.

3.2 Sort Last Software

The Sort-last approach distributes the 3D model data only once before any transformations or rendering have occurred. This distribution does not take into account the viewpoint or object coordinates. Each rendering node gets ***T/N*** triangles where ***T*** is the total number of triangles and ***N*** is the number of rendering nodes. Each node performs the transformations and rasterization of their own triangles. The frame buffer and the corresponding z-buffer data is then read from each node and a pixel-by-pixel z-buffer comparison is performed in parallel according to the user specified composition algorithm. The final composited image data is gathered into a contiguous chunk of memory on single node. Since the single node now has this image in memory, it is not tied to displaying the image locally. It can, if desired, instead send the data to an alternate display

The PGLC (Parallel OpenGL Compositor) Sort-last library originates from the Parallel Mesa Library[8] software, developed both at the State University of New York (SUNY) and SNL PGLC abandons any dependencies on Mesa and will work with any OpenGL compliant environment. This allows for the exploitation of OpenGL hardware acceleration where available. The PGLC library is written in C and uses MPI for interprocess communication. PGLC runs on both the Unix and Windows platforms and can be linked to existing parallel applications. The API is extremely simple as can be seen from the following example template usage.

```
#include <pglc.h>
void main(int argc, char *argv[])
{
    pglc_Init();
    pglc_Wincreat(width,height,xPos,yPos,title);

    while (1) {
        Computation
        .
        OpenGL Calls
        .
        frameBuffer = pglc_Flush(COMPRESSED_TREE);
    }
}
```

The PGLC library consists of only three API functions. The first two functions initialize data structures and create a platform independent window of a specified dimension and position as well as an OpenGL context which is bound to this window. The last function 'pglc_Flush' performs the parallel composition and returns a memory pointer to the location of memory that contains the fully composited image. The image can be then be displayed in the window specified by 'pglc_Wincreat' or sent to alternate display systems as needed. As the single argument to 'pglc_Flush' the user can specify a variety of different composition algorithms. The choice of algorithm will depend on the topology and performance of the interconnect used on a specific cluster.

One important concept here is that the method of rendering is left entirely up to the application; this flexibility allows the application to use hardware specific optimizations such as triangle-strip display lists for SGI IR pipes or compiled vertex arrays for GeForce™ cards.

3.3 Sort Middle Software

The Sort-middle classification distributes the 3D model primitives based on the current viewpoint and object coordinates. For a new viewpoint each processor transforms its current store of primitives and distributes those that no longer fall within its viewing quadrant. When all processors have their current working set of primitives, that set of polygons is then sent to the graphics hardware where each polygon is transformed and rasterized to produce a completed tile of the final image. At this point, if the tiles in your display hardware are directly linked to your working nodes, you are done. In the case of having N nodes and M display tiles, the additional task of distributing sub-images, and then stitching or splitting them, has to be performed.

The Tiled Display Library (TDL) implementation and exposed API are similar to PGLC. The TDL library is written in C and uses MPI for interprocess communication. TDL runs on both the Unix and Windows platforms and can be linked to existing parallel applications. An example usage of the API is:

```
#include <tdl.h>
void main(int argc, char *argv[])
{
  TDL_Init(numXtiles,numYtiles,myX,myY);
  TDL_SetGeometry(coords,normals,colors,
                  triangles,numTriangles);
  while (1) {
    TDL_SetViewMatrix(vMatrix);
    TDL_GetGeometry(&coords,&normals,&colors,
                    &triangles,&numTriangles);
                .
    OpenGL Calls
  }
}
```

The TDL library consists of 4 API functions. The first function allows the application to specify the topology of the tiled display and that particular nodes position within the display. Typically the parameters to 'TDL_Init' reflect the physical layout of the display, but a logical tiling could also be used. The use of

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

As Graph 4 demonstrates, TDL scales fairly well up to about 8 processors, but more than 4 displays caused load-balancing delays. Although the pure polygonal performance of TDL is well below that of PGLC, the resolution of the tiled display is much greater. In Graph 4 the last data point represents an average performance of 5.1 million polygons/sec at a resolution of approximately 8 mega pixels.

5 ISSUES

Many challenges still remain with regard to making effective production resources out of cluster-based graphics systems. 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 project onto a single tile, in which one renderer has to do all of the work. Clearly, some mechanism is needed for separating the rendering function from the display. We have shown that cluster-based systems show 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 closely 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 non-decreasing) performance as more and more resources are applied.

Certain pragmatic challenges also exist. The administration of cluster-based systems is nontrivial, and mechanisms are needed 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 upon accessibility as a dynamically shared resource to support highly interactive work, presenting some unique challenges for resource management.

6 FUTURE WORK

We expect to thoroughly 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.
- Scalability assessments on larger graphics clusters (we are currently in the process of procuring a 64-node cluster; we expect to demonstrate rendering on the order of 100 million polygons per second later this year).
- Processing of time-dependent data and addressing issues related to feeding data to the parallel rendering system.
- Integration of graphics clusters into our end-to-end high performance computing environments.

7 ACKNOWLEDGEMENTS

Funding was provided by the Accelerated Strategic Computing Initiative's Visual Interactive Environment for Weapons Simulations (ASCI/VIEWS) program. Thanks to LLNL for 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 R&D cluster support, Phil Heermann for his inspiration and motivation, and especially Lisa Ice for her excellent work on PMESA. This work is supported by the United States Department of Energy under contract DE-AC04-94AL85000.

REFERENCES

[1] Heermann, P. Production Visualization for the ASCI One TeraFLOPS Machine. *Proceedings of Visualization '98*, pages 459-462. IEEE, October 1998.

[2] Smith, P. H. and van Rosendale, J. *Data and Visualization Corridors, Report on the 1998 DVC Workshop Series*. Caltech, 1998.

[3] Molnar, S. et al. A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics and Applications*, pages 23-32. July 1994.

[4] Whitman, S. A Task Adaptive Parallel Graphics Renderer. *1993 Parallel Rendering Symposium Proceedings*, pages 27-34. IEEE. October 1993.

[5] Crockett, T. W. and Orloff, T. A MIMD Rendering Algorithm for Distributed Memory Architectures. *1993 Parallel Rendering Symposium Proceedings,* pages 35-42. IEEE. October 1993.

[6] Lee, T. et al. Image Composition Methods for Sort-Last Polygon Rendering on 2-D Mesh Architectures. *1995 Parallel Rendering Symposium Proceedings*, pages 55-62. IEEE. October 1995.

[7] Whitman, S. A Load Balanced SIMD Polygon Renderer. *1995 Parallel Rendering Symposium Proceedings, pages 63-69*. IEEE. October 1995.

[8] Mitra, T. and Chiueh, T. Implementation and Evaluation of the Parallel Mesa Library. *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. December 1998.

[9] Steven Molnar, John Eyles and John Poulton, *``PixelFlow: High-Speed Rendering Using Image Composition",Proceedings of SIGGRAPH '92, Chicago, Illinois, July 1992, 231-240.*

[10] Samanta, R. et al. Load Balancing for Multi-Projector Rendering Systems. *SIGGRAPH/Eurographics Workshop on Graphics Hardware.* August, 1999.

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

[12] Schikore, D. et al. High-resolution multi-projector display walls and applications. Accepted for publication in *Computer Graphics and Applications.*

[13] Heirich, A. and Moll, L. Scalable Distributed Visualization Using Off-the-Shelf Components. *1999 IEEE Parallel Visualization and Graphics Symposium* Proceedings, pages 55-59. IEEE. October 1999.

logical tiling (smaller than physical and interlaced) would help alleviate the load balancing issues inherent to Sort-first and Sort-middle architectures. Currently we have not investigated the use of TDL with logical tiles.

The second function 'TDL_SetGeometry' allows the user to specify the initial partitioning of the primitives. An application may want to specify some optimal initial partitioning to minimize the communications during the very first distribution phase, but in practice we pay no attention to this and let the library sort it out. With these initializations out of the way, there are two functions that get repeatedly called. The 'TDL_SetViewMatrix' function simply takes the current 4x4 view matrix as its argument. The real work happens in the 'TCL_GetGeometry' function; this function partitions the data based on the current viewing transformation and hands back pointers to the geometry within the tile's view frustum. At this point, as with PGLC, the method of rendering is left up to the application.

4 PERFORMANCE EVALUATION

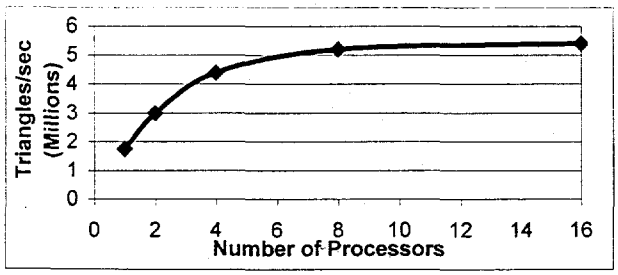
4.1 System Performance

Our 16-node cluster was primarily designed as a research vehicle for the exploration of scalable rendering software and algorithms. The opportunity to study the advantages of UMA based machines lead us to the selection of the SGI 320's. When performing the image compositing necessary for Sort-last, the application must read the frame buffer. With UMA based machines, the frame buffer resides in main memory and thus this part of the composition phase is greatly accelerated. The graphics performance of the nodes, using vertex arrays in OpenGL is about 1.75 million triangles/sec in our applications.

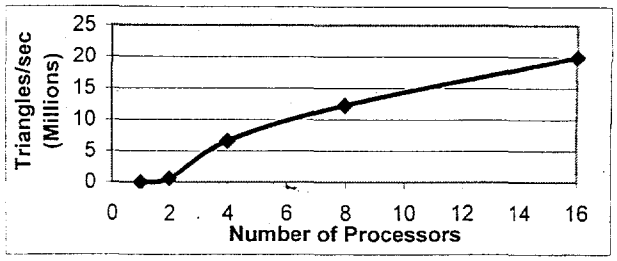
These machines each contain one 450MHz Intel processesor and have comparable performance to similar models from other companies. Our network is a Gigabit Ethernet without Jumbo packets. Currently, because of various equipment issues, we are only getting a peak throughput of 270Mbits/sec from node to node. Our interconnect's sluggish performance leads to fairly high overheads during communication and consequently we are utilizing compression for certain data transfers.

4.2 Sort Last

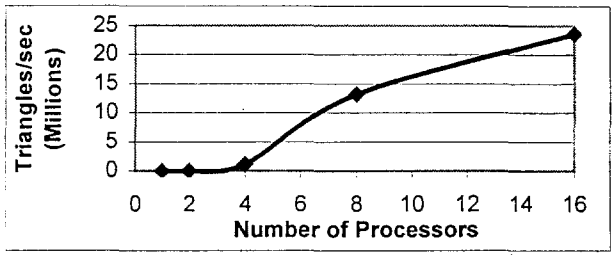
To evaluate the performance of both the PGLC and TDL libraries, we wrote two parallel applications that link to the respective libraries. The programs read in fractions of the dataset from disk and then make function calls as demonstrated in sections 3.2 and 3.3. During the performance evaluations the largest available dataset contained 1.3 million triangles. In order to determine scalability, the dataset vertices were replicated. We are in the process of obtaining larger isosurface datasets.



Graph 1: PGLC performance on a 1.3 million triangle dataset.



Graph 2: PGLC performance on a 13 million triangle dataset.



Graph 3: PGLC performance on a 26 million triangle dataset.

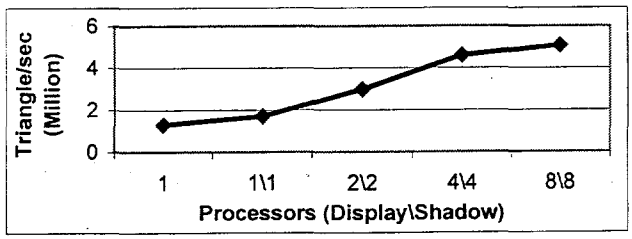
The performance numbers obtained from an application using the PGLC library are given in the above graphs. Graph 1 shows that with a smaller dataset, the inherent network overhead associated with the image compositition starts to hinder performance as the number of processors increases. As we can see from Graphs 2&3, as the size of the dataset increases, PGLC shows dramatic performance improvements. In Graph 3 PGLC reached an average performance of 23.6 million triangles per second with 16 nodes. Graph 3 also demonstrates that with this performance we are getting 94% utilization of the aggregate performance when running on 8 nodes and 84% utilization when running on 16 nodes (assuming 1.75 Mtris/sec for each node).

The super linear behavior seen in both Graph 2 and Graph 3 are due to the fact that the larger size datasets ran extremely poorly or not at all on 1,2, and 4 node configurations.

4.3 Sort Middle

The use of large tiled displays is becoming common. High fidelity simulations demand visualizations with greater screen area and increased resolution. The TDL library provides scalable rendering and large tiled display functionality.

In order to maximize the utilization of the display nodes the TDL library employs the use of 'shadow' nodes. These shadow nodes are not connected to the displays and serve as the directors of data traffic. As the display nodes are rendering the current frame the shadow nodes are computing the data distribution needed for the upcoming frame. For small configurations (2-display/2-shadow and 4-display/4-shadow) we are measuring between 70% and 85% utilization of the aggregate performance of the graphics hardware.



Graph 4: TDL performance on a 1.1 million triangle dataset.