Massively Parallel Visualization: Parallel Rendering*

Charles D. Hansen† Michael Krogh† William White‡

Abstract

This paper presents rendering algorithms, developed for massively parallel processors (MPPs), for polygonal, spheres, and volumetric data. The polygon algorithm uses a data parallel approach whereas the sphere and volume renderer use a MIMD approach. Implementations for these algorithms are presented for the Thinking Machines Corporation CM-5 MPP.

1 Introduction

In recent years, massively parallel processors (MPPs) have proven to be a valuable tool for performing scientific computation. Available memory on this type of computer is far greater than that which is found on traditional vector supercomputers. For example, a 1024 node CM-5 contains 32 gigabytes of physical memory. As a result, scientists who utilize these MPPs can execute their three dimensional simulation models with much greater detail than previously possible. Molecular dynamics simulations can consist of over 100 million atoms [8] and CFD simulations can contain over 23 million cells with numerous variables [9]. While these applications allow for better simulation of the underlying physics, they typically cause a data explosion. As the resolution of simulation models increases, scientific visualization algorithms which take advantage of the large memory and parallelism of these architectures are becoming increasingly important.

Renderers, used to transform data into images, can be classified into either geometry-based or volume-based. Geometry-based renderers are used when scientific simulations contain explicit geometry, such as material interface boundaries, or when implicit geometry is derived, such as from isosurfaces, particles, spheres, vectors, etc. While geometry extraction may be used as a lossless compression technique [1, 2], it more typically generates larger amounts of data than are present in the original dataset [4]. Volume-based approaches produce an image directly from the scientific data without utilizing explicit geometry [1, 6]. The data are rendered directly into an image through color and opacity transfer functions.

For large applications rendering on the MPP tends to be preferable to rendering on a graphics workstation due to the MPP's abundant resources: memory, disk, and numerous processors. The challenge becomes developing algorithms that can exploit these resources while minimizing overhead, typically communication costs.

1.1 Rendering

Molnar et al. provide a useful taxonomy for parallel rendering which classifies rendering methods, based on where data are sorted from object space to image space, as sort-first, sort-middle, or sort-last [5]. For our applications sort-last, a compositing methodology, has

*This work support by the DOE HPCRC Grant

†Advanced Computing Laboratory, Los Alamos National Laboratory, viz@acl.lanl.gov

‡Department of Computer Science, University of North Dakota, wwhite@acl.lanl.gov

TABLE 1
Mapping Graphics Pipeline to VP sets

Operation	Primitive	VP Set	Operation	Primitive	VP Set
transformation	vertices	polygon	scan conversion	polygons	polygon
transform to screen space	vertices	polygon	rasterization	scan lines	scan line
shading	vertices	polygon	Z-buffer	pixels	pixel

demonstrated superior performance. Its strengths are better overall load balancing and logarithmic image compositing times. Disadvantages are that each processor must have enough memory for the image buffer and that compositing involves communicating the image buffer among processors. The rest of the paper presents three sort-last renderers that we have developed: a SIMD polygonal renderer, a MIMD sphere renderer, and a MIMD volume renderer.

The polygon renderer handles complex polygons and is tuned for smaller polygons which are typical in large scientific data sets. It uses a pixel sorting approach to sort-last. Advantages of this technique include better network utilization, excellent polygon rates for large polygonal data sets, integration into existing visualization environments, and load balancing.

The sphere renderer treats spheres as primitives instead of tesselating the spheres into polygons. It uses a logarithmic image compositing approach to sort-last. Advantages of this technique include out-of-core support for arbitrarily large images and data sets, load balancing, and an optimal compositing algorithm.

The volume renderer efficiently renders very large 3D scalar fields. It subdivides the data among the processors and uses a unique compositing techniques which maximizes processor efficiency.

2 Polygon Renderer

For the CM-5, one must chose between the data parallel programming model and the MIMD message passing programming model. For polygon rendering, we have experimented with both programming models [3] and in this section, we describe our experiences with a sort-last polygon renderer written in the data parallel style.

The basic idea behind the data parallel renderer is to maximize the number of operations occurring in parallel while minimizing communication. While this trait is desirable in both data parallel and task parallel programming models, the SIMD/SPMD nature of data parallel programs imposes additional constraints. In data parallel programs, there is only one thread of control. For efficient programs, it is necessary to maximize the set of active processors at any given step in an algorithm. This is accomplished by judicious assignment of data to the processors, sometimes referred to as *layout*.

2.1 Data Layout

To determine the optimal layout for the rendering process, we examined the standard graphics pipeline with respect to data operations. Table 1 breaks down the standard graphics rendering pipeline into basic steps. Each of these operations is categorized by the primitive upon which the operation is performed. Lastly, the virtual processor (VP) set is indicated.

The first three steps operate upon vertices. Each vertex is transformed and then shaded.

In this implementation, we are optimizing for speed. Therefore, we perform simple Gouraud shading. In Gouraud shading, the shading is computed at each vertex and then linearly interpolated across an edge when forming a scan line segment and linearly interpolated across the scan line segment during rasterization, resulting in a smoothly shaded object. More advanced shading techniques are easy to implement.

The fourth step scan converts the polygons by determining which polygon edges intersect a particular scan line and interpolating the X, Z and shaded color information along the polygon edge to determine the value for a particular Y scan-line.

Hidden surface elimination is accomplished by employing a parallel Z-buffer algorithm. This is done by rasterizing the line segments produced from the scan conversion process, clipping the resulting pixels against the viewport and then Z-buffering the non-clipped pixels. The Z-buffer tiles the image plane such that independent/non-overlapping regions of the screen are assigned to individual virtual processors.

If we strictly followed this, we would remap the virtual processors from vertices to polygons to line segments to pixels. The remapping of virtual processors involves general communication which is costly. If we map each polygon to a virtual processor and then iterate over the vertices within each polygon, we can eliminate one of the communications.

The most interesting parts of the algorithm are the scan conversion and Z-buffering. The scan conversion process iterates over the maximum number of scan lines through any polygon. Since scan conversion is concurrently executed for all polygons in parallel, it is bounded by the maximum number of scan lines within any polygon. Thus number of iterations necessary to process the entire set of polygons is the maximum number of scan lines spanning any polygon. As the number of scan lines processed approaches the maximum, fewer polygons are processed, since some polygons, those with a smaller number of scan lines passing through them, will have completed the scan conversion process. We address this load balancing issue in the next subsection.

In the Z-buffering step, line segments from the previous steps are converted to pixels. The individual pixels are routed to the VP which is responsible for that particular screen region. This is accomplished through the **sendmax** operator. Where pixels from different polygons are mapped to the same image location, the hidden-surface elimination is performed by choosing the pixel with the maximal Z value.

2.2 Load Balancing

In the renderer, there are two key loops, one for scan converting polygons into line segments and one for Z-buffering the line segments. As previously noted, when virtual processors (VP) complete the scan conversion of their polygons, they become idle. VPs can also become idle if the polygon is clipped or backface culled. Idle VPs lead to a load balancing problem.

We address this problem by dynamically redistributing remaining portions of polygons to be scan converted to idle VPs. This reuses existing memory space and attempts to keep all VPs active during the scan conversion process. Redistribution of the work load is determined by keeping track of the time taken to process a scan line, and the time to redistribute work. If the saving in loop iteration is less than the time to perform the redistribution, it makes no sense to perform the redistribution.

Table 2 gives the rendering times for different partition sizes on the CM-5.

TABLE 2
Rendering of Polygons on CM-5

CM-5 Partition Size	32	64	128	256	512
Time (sec)	1.656	0.867	0.559	0.327	0.205
Polygons/sec	137,855	263,308	408,386	677,412	1,113,600

3 Sphere Renderer

As with the polygon renderer we have also experimented with both sort-middle and sort-last approaches to sphere rendering. Sort-last has proven to have superior performance for our application. Since the amount of work to process a sphere varies with its radius, we chose the MIMD model so processors can run asynchronously and maximize utilization.

3.1 Data Layout

The sphere renderer supports an unlimited number of spheres and any image size. If necessary, based on memory constraints, spheres may be handled in steps. Likewise, an image may be processed in steps. With sort-last each processor has its own image and Z-buffers for the current part of image that is being worked. The current set of spheres, which are distributed equally among processors, are rendered into local image and Z buffers.

All processors then assign their spheres a color (based on a scalar value such as kinetic energy), transform them to image space, and scale their radii for perspective. Spheres closer to the camera will appear larger than spheres of the same radius which are further from the camera.

Spheres are then scan converted, one at a time, into the image and Z buffers. The scan conversion is done by evaluating a distance equation for each pixel within the bounding box for the sphere. If the current pixel is within the sphere, then a Z-buffer comparison is made and if the pixel is not hidden, the color is determined. Lighting is approximated by including an offset, based on sphere center and current pixel location, in the color calculation. This gives the illusion of a light at fixed location from the camera.

After each processor has finished rendering, the N Z-buffers are composited, in logarithmic time using the CMMD function **CMMD_reduce_v**, with a minimum operator to select pixels, across all processors, closest to the camera. Processors then zero out pixel colors in their image buffers wherever their local Z-buffers don't match the composite Z-buffer. Image buffers are then composited using a maximum operator.

3.2 Load Balancing

Although all processors initially receive an equal amount of work, if an image is being processed in steps or if spheres are transformed out of the image, then some processors may have significant load balancing problems. This is resolved by migrating some atoms, after the object-to-image space transformation, from heavily loaded processors to lightly loaded processors. A simple approach of sorting processor loads and matching up the lightest load with the heaviest load, and so on, seems to yield acceptable results. Other approaches are being investigated.

3.3 Results

To gauge our algorithm's performance we benchmarked it on several different CM-5 partitions and a SGI Onyx with Reality Engine II graphics engine. The SGI workstation

Table 3

Sphere Rendering Times

CM-5 Partition Size	32	64	128	256	512
Time (sec)	284	160	91	53	36

SGI Tesselation Factor	1	5	9
Time (sec)	777	3066	8215

uses a simple, but optimized, program that invokes the SGI sphere drawing routine. Their sphere drawing routine tesselates a sphere into a set of triangle or quadrangle meshes (depending on user selected tesselation method). The number of polygons generated depends upon the user selected tesselation factor. High factors yield rounder spheres.

Table 3 shows time (in seconds) to render a data set containing 37,993,550 atoms on various CM-5 partition sizes and on a SGI Onyx with various tesselation factors.

4 Volume Renderer

Direct volume rendering differs from geometry rendering in that the data are rendered by compositing color/opacity pairs, derived from the data values, into an image. This can be done either by projecting the volume samples onto the image plane through a technique such as splatting [6], or by ray-casting which involves sampling data values along rays projected from the camera through the image plane and into the data set [1, 6].

Although no geometry is processed, ray-casting techniques can be classified by the same rendering taxonomy. We have developed a sort-last volume rendering algorithm which maximizes processor utilization during the compositing phase by taking advantage of data locality [7].

4.1 Data Layout

Since ray-cast volume renderers are inherently parallel, parallelization of the ray-casting is trivial when the data volume is replicated at every node. However, for large 3D scalar fields, it is not feasible to replicate the data and clever techniques for data space decomposition and final compositing are required.

We have developed a data space partitioning scheme based upon K-D trees. Each level in the K-D tree is formed by alternating binary subdivision of the coordinate planes. This leads to a block decomposition of the data volume where each node of the MPP contains a subvolume of the original data set. Each subvolume is rendered, independently and concurrently, by ray-casting from the identical view direction; and, only rays within the image region covering the corresponding subvolume are cast and sampled. This results in a partial image in each processor node for its subvolume.

We have developed a unique method for compositing these final images, called Binary-Swap Compositing, which maximizes processor utilization. The basic idea of the compositing algorithm is that a processor swaps 1/2 of its image with 1/2 of its neighbor's image. Each processor then composites its own half with that received from the neighbor. Next, the processor swaps 1/2 of that sub-image (resulting in 1/4 of the total image) with another neighbor and the quarters are composited. At the final stage, each processor will have a portion of the final image. By controlling which neighbors swap which part of the image, the final image layout among the processor nodes can be optimized.

In the early phases of the binary-swap algorithm, each processor is responsible for a

TABLE 4
Volume Rendering 256^3 *Volume on CM-5*

CM-5 Partition Size	32	64	128	256	512
Time (sec) Render	48.2005	24.4303	12.697	6.3434	3.1878
Time (sec) Composite	0.71520	0.58100	0.4272	0.3874	0.3310

large portion of the image area, but the data coverage in the image area is usually sparse because only a few processors have contributed to it. In later phases of the algorithm, as we move up the compositing tree, the processors are responsible for a smaller and smaller portion of the image area, but the density of data coverage increases because an increasing number of processors have contributed image data. In the early phases, a larger amount of data is communicated when communication is with nearest neighbors. In the later phases, a smaller amount of data is communicated which is when communication is among non-local processors. This effectively utilizes the bandwidth constraints of MPPs.

Table 4 shows the rendering and compositing times for a 256^3 3D scalar field of vorticity data volume rendered on the CM-5.

5 Conclusion

We have shown that rendering can be performed at very high rates on a MPP and is better suited, compared to graphics workstations, for handling the quantities of data produced by simulations that run on these machines.

Also, for our magnitude of data we have found that a sort-last approach performs better than a sort-middle approach and has the advantage of not relying on a hardware specific rendering engine.

References

[1] M. Levoy, *Efficient Ray Tracing of Volume Data*, ACM Transactions of Computer Graphics, 9(3), (1990).

[2] W. Lorensen and H. Cline, *A High Resolution 3D Surface Construction Algorithm*, Computer Graphics (SIGGRAPH Proceedings), 21 (1987), pp. 163-169.

[3] F. Ortega, C. Hansen, and J. Ahrens, *Fast Data Parallel Polygon Rendering*, Proceedings of Supercomputing '93, November 1993

[4] C. Hansen and P. Hinker, *Massively Parallel Isosurface Extraction*, Proceedings of Visualization '92, (1992), pp. 77-83.

[5] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, *A Sorting Classification of Parallel Rendering*, IEEE Computer Graphics and Applications, 14(4), July 1994, pp. 23-32.

[6] T. Elvins, *A Survey of Algorithms for Volume Visualization*, ACM Computer Graphics Quarterly, 26(3) (1992).

[7] K.L. Ma, J. Painter, C. Hansen, and M. Krogh, *Parallel Volume Rendering Using Binary-Swap Image Composition*, IEEE Computer Graphics and Applications, 14(4), July 1994, pp. 59-68.

[8] P.S. Lomdahl, P. Tamayo, N. Gronbech-Jensen, and D.M. Beazley, *50 GFlops Molecular Dynamics on the Connection Machine 5*, Proceedings of Supercomputing 93, November 1993, pp. 520-527.

[9] J.K. Dukowicz, R.D. Smith, and R.C. Malone, *A Reformulation and Implementation of the Bryan-Cox-Semtner Ocean Model on the Connection Machine*, Journal of Atmospheric and Oceanic Technology, 10(2) April 1993, pp. 195-208.