Irregular Grid Volume Rendering With Composition Networks

Craig M. Wittenbrink
Computer Systems Laboratory
HPL-97-51 (R.1)
January, 1998

E-mail: craig@hpl.hp.com

tetrahedral cell, parallel rendering, parallel compositing, PixelFlow

Volumetric irregular grids are the next frontier to conquer in interactive 3D graphics. Visualization algorithms for rectilinear 256^3 data volumes have been optimized to achieve one frame/second to 15 frames/second depending on the workstation. With equivalent computational resources, irregular grids with millions of cells may take minutes to render for a new viewpoint. The state of the art for graphics rendering, PixelFlow, provides screen and object space parallelism for polygonal rendering. Unfortunately volume rendering of irregular data is at odds with the sort last architecture.

I investigate parallel algorithms for direct volume rendering on PixelFlow that generalize to other compositing architectures. Experiments are performed on the Nasa Langley fighter dataset, using the projected tetrahedra approach of Shirley and Tuchman. Tetrahedral sorting is done by the circumscribing sphere approach of Cignoni et al. Key approaches include sort-first on sort-last, world space subdivision by clipping, rearrangeable linear compositing for any view angle, and static load balancing. The new world space subdivision by clipping provides for efficient and correct rendering of unstructured data by using object space clipping planes. Research results include performance estimates on PixelFlow for irregular grid volume rendering. PixelFlow is estimated to achieve 30 frames/second on irregular grids of 300,000 tetrahedra or 10 million tetrahedra per second.

Internal Accession Date Only

To be published in the *Proceedings of SPIE, The International Society for Optical Engineering, Visual Data Exploration and Analysis V*, Conference 3298, January 1998

Irregular grid volume rendering with composition networks

Craig M. Wittenbrink
Hewlett-Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304

Volumetric irregular grids are the next frontier to conquer in interactive 3D graphics. Visualization algorithms for rectilinear 256^3 data volumes have been optimized to achieve one frame/second to 15 frames/second depending on the workstation. With equivalent computational resources, irregular grids with millions of cells may take minutes to render for a new viewpoint. The state of the art for graphics rendering, PixelFlow, provides screen and object space parallelism for polygonal rendering. Unfortunately volume rendering of irregular data is at odds with the sort last architecture.

I investigate parallel algorithms for direct volume rendering on PixelFlow that generalize to other compositing architectures. Experiments are performed on the Nasa Langley fighter dataset, using the projected tetrahedra approach of Shirley and Tuchman. Tetrahedral sorting is done by the circumscribing sphere approach of Cignoni et al. Key approaches include sort-first on sort-last, world space subdivision by clipping, rearrangeable linear compositing for any view angle, and static load balancing. The new world space subdivision by clipping provides for efficient and correct rendering of unstructured data by using object space clipping planes. Research results include performance estimates on PixelFlow for irregular grid volume rendering. PixelFlow is estimated to achieve 30 frames/second on irregular grids of 300,000 tetrahedra or 10 million tetrahedra per second.

Keywords: tetrahedral cell, parallel rendering, parallel compositing, PixelFlow

1. INTRODUCTION

Scientific visualization as a research discipline has faced two main challenges: manipulation of large datasets and creation of easy to understand pictures from hard to understand data. Large may be defined as the amount of data it is possible to store on a system. Creation of easy to understand pictures from hard to understand data is difficult. Additionally, there are people who understand complex data without visualizations. One of the interesting problems in visualization that has always expanded the definition of large datasets and hard to understand data has been volume rendering.[1] There are numerous tools with which to investigate volume data, such as cutplanes and isosurfaces. Direct volume rendering creates higher information densities by accumulating contributions along view rays. For some datasets, this type of visualization is very useful in providing context, understanding, and mapping to visual cues. Unfortunately the advantages of direct volume rendering come with some costs. Volume rendering takes an order of magnitude more time and memory than cutplanes and isosurfaces. Figure 2 shows four types of volume datasets: regular, rectilinear, curvilinear, and unstructured.

For regular and rectilinear datasets, a combination of algorithm developments, advances in silicon gate densities, and special purpose hardware, have solved many of these problems. Interactive volume rendering is a reality today with either a software only solution,[2] a hardware assisted solution,[3] or through a special purpose volume graphics hardware.[4,5] These solutions trade some fidelity for performance, but they provide changing viewpoints with interactive 1-30 frames/second.

For irregularly gridded volume datasets there are software only renderers,[6] and polygon accelerated hardware renderers.[7] But, interactivity is possible only for lower fidelity approaches or small datasets.[8] A challenge is interactive performance for changing viewpoints in volume rendering of large irregularly gridded volume datasets. Such datasets are computed in computational fluid dynamics and finite element analysis.

Providing interactivity requires applying optimally tuned graphics hardware and software to accelerate the features of interest. Many researchers in the past[9] have commented that creation of special purpose hardware would solve the slowness of handling irregular grids. And, general purpose hardware is not adequate, because the operations per second requirement is simply too high.

Other author information: (E-mail: craig@hpl.hp.com)

The fundamental algorithm, developed by Shirley and Tuchman[9] is the projected tetrahedra algorithm. Tetrahedra are projected at the screen plane, and subdivided into triangles. Figure 3 shows the four classes of projections, where from one to four triangles result, right column. Polygonal approximation[9] is a good starting point for accelerated rendering of irregularly gridded datasets, because the majority of the work, the scan conversion and compositing are accelerated by existing graphics hardware. The basic algorithm is as follows:

```
I. Preprocess dataset
For a new viewpoint:
II. View sort cells
   For every cell in sorted back-to-front order:
   III. (H) View transform cell
   IV. Project and split cell unique frontback faces
   V. Compute color and opacity for thick vertex
   VI. (H) Scanconvert new triangles
```

Figure 1. Projected tetrahedra algorithm pseudo-code

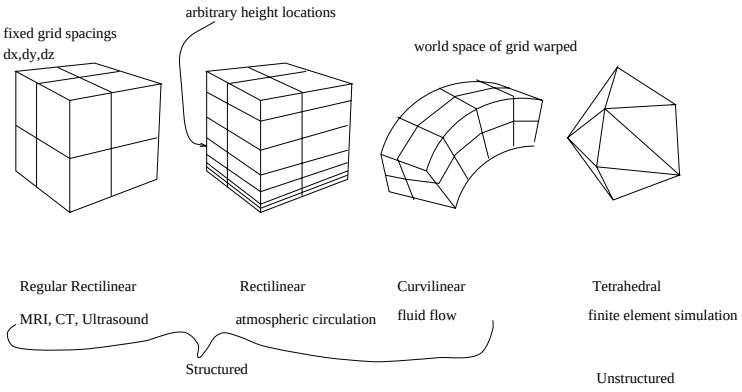

Figure 2. Example grid types.

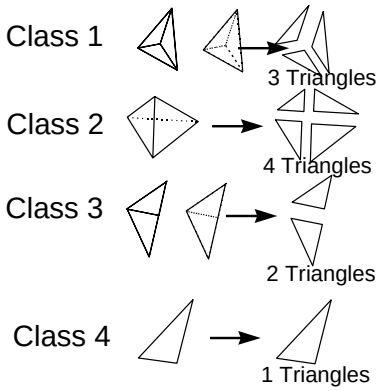

Figure 3. Projected tetrahedra, 4 classes.

Preprocessing is necessary to calculate colors and opacities from input data, setup for depth sorting of primitives, and creation of plane equations used for determining the class a tetrahedra belongs to for a viewpoint. A depth sorting of cells (step II) is done, so that the proper compositing results. The cells need to be visibility sorted for each viewpoint.[10] Figure 3 shows that new vertices are introduced, during steps III and IV. The new vertex requires a color and opacity to be calculated, step V. Then the triangles are drawn and scan converted, step VI.

Current graphics hardware is not interactive for moderate sized datasets of thousands to millions of cells. Special purpose volume hardware or reprogrammable graphics hardware are ways to accelerate the projected tetrahedra algorithm. PixelFlow[11,12] is a massively parallel heterogeneous computer that uses screen space and object space parallelism for scalability in polygonal rendering. Because of its reprogrammability it can also be tuned for volume rendering of irregular data.

The work in adapting parallel algorithms for direct volume rendering on PixelFlow can be used for other compositing architectures. Several compositing architectures have been designed and built.[13–15] The key advantage is the ability to partition work in the depth direction, and the partitioning of input primitives. There are several strategies for rendering of irregularly gridded data on a composition network. The primary differences are in how the data set is visibility sorted. To volume render, volume cells must be composited in the proper order. The approaches are 1) render cells with assigned object space regions and composite layers, 2) render distributed cells and sort and composite primitives, and 3) allow out of order compositing.

For PixelFlow, I have determined that the first approach is the best. I discuss using an object space partitioning, local sorting, and a following compositing of layers. A new method of object space clipping of tetrahedra across

planes provides accurate and efficient partitioning. Such an algorithm has expected performance on PixelFlow (36 renderers) of 30 frames/second on 300,000 tetrahedra or 10 million tetrahedra/second. Composition network graphics computers are reviewed in Sect. 2. The proposed algorithm is detailed in Sect. 4. Performance estimates are developed in Sect. 5, and the paper concludes in Sect. 6.

2. COMPOSITION NETWORK GRAPHICS COMPUTERS

In parallel rendering, the typical approach to exploiting parallelism has been either pixel level parallelism or pipeline parallelism. When driven to create higher scalability, the next realm of parallelism is screen depth parallelism which is possible when you use a compositing network.

The basic composition architecture is shown in Figure 4. The geometry and rasterization pipelines operate in parallel for many separate primitives. Figure 4 shows four way parallelism. The parallelism results in dividing the database to be rendered amongst the four separate rendering pipelines without requiring them to be carefully depth sorted. Each pipeline has a geometry processing stage (G) and a rasterization stage (R). The depth sort is done by brute force in the compositing layers (C). In this way any viewpoint can be rendered with a static database assignment, as the sorting is performed after rasterization (R). This provides the same performance irrespective of viewpoint if the data has been divided roughly evenly amongst the pipelines, and eliminates the need for sorting or moving geometry. The composition nodes may be augmented to support volume rendering by adding opacity compositing. As compositing architectures for polygonal rendering were not designed to support volume rendering, they actually only do z-depth comparisons. They do not perform opacity or alpha compositing, which is required to combine the many layers in performing a volume rendering.

An early example of an image compositing architecture, and one that was successfully applied to volume visualization is the Pixar computer.[16,1] Likely one of the most well known of the compositing architectures is PixelFlow designed by Molnar et al.[12,11] A successor to the PixelPlanes[17] technologies, using enhanced raster memories, PixelFlow incorporates a unique backplane bus that provides z compositing between many image layers. The z comparisons are done prior to shading, so that once all comparisons have been done, and the appropriate normals, material values, and colors have been passed to the shading board, each pixel is shaded only once, called deferred shading. Compositing parallelism allows rendering of many millions of primitives, because the shading effort is fixed to the screen size, and if more primitives need to be rendered, more rendering boards are added which can all composite their results on the backplane.

Other compositing architectures have been developed since PixelFlow, such as the VC-1,[14] MUSIC,[15] and Talisman.[18] The VC-1 uses an image composition architecture, and uses virtual local frame buffers to reduce the amount of storage necessary on each. A token ring network can be used to communicate the data necessary for compositing. The MUSIC architecture provides intelligent communication controllers to augment a token ring communication bus to perform image composition or depth sorting of primitives. The Microsoft Talisman architecture also does compositing of layers, and includes a special chip to composite many different layers. But, the Talisman approach uses compositing mostly to exploit temporal coherence so that data does not need to be rerendered.

In parallel rendering for volume visualization, parallel compositing has been considered in detail.[19,20,13] In volume rendering the amount of compositing required is very high, because the image depth complexity is the number of samples taken through the dataset. In polygonal rendering this typically is not the case. A more fully parallel compositing can be done by binary tree combining as shown through the binary swap technique,[20] via a parallel product fully parallel compositing as we developed,[19] and by a dedicated binary tree swapping architecture as developed by Malzbender.[13]

To apply a compositing network to volume rendering requires splitting the volumetric database amongst nodes. This must be done carefully as severe load imbalances can occur for particular viewpoints. Solutions to load balance include either data replication (may be costly) efficient subdivision, or data migration. Each architecture has limitations and restrictions for volume visualization. The VC-1, uses a token ring network, which has a limited amount of bandwidth. When image complexity reaches a certain level, the bus bandwidth requires slowing down the frame generation time. The frame generation time is also affected if the smaller frame buffer available on a given pipeline overflows, and the system goes to the virtual memory pool. PixelFlow has a fixed composite area size, necessary to sustain the high bandwidths on the compositing bus, but which restricts granularity choices on how compositing is performed. PixelFlow also uses a custom SIMD rasterization array that was designed for polygonal graphics.

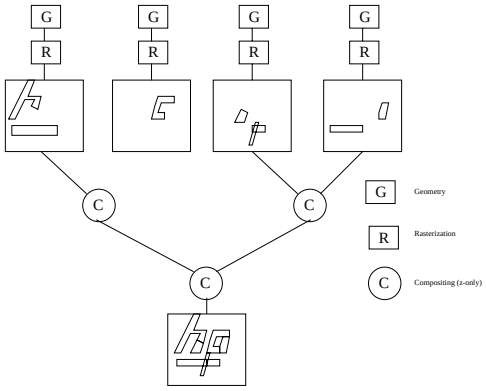

Figure 4. Composition network.

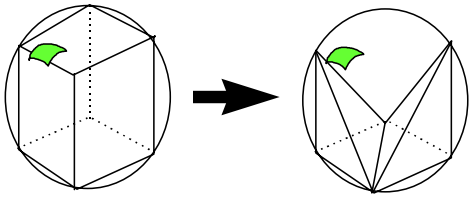

Figure 5. Worst case circumscribing scenario.

The SIMD array is reprogrammable, but because of memory access patterns and instructions designed for polygon rendering, is not easily applied to more general problems. Talisman was designed for handling images, but only images with a restricted depth complexity. With many scenes and polygonal worlds the designer can craft the depth complexity, but with volumetric rendering, the depth complexity is necessarily high. MUSIC, also uses a token ring bus, which limits compositing bandwidth. The architecture also uses only general purpose processors, which severely limits the rate at which polygonal or volumetric data may be resampled and rasterized.

3. EXISTING WORK IN IRREGULAR GRID VOLUME RENDERING

Volume visualization has been thoroughly researched in many aspects and applications. I briefly cover some of the main background here for completeness, but encourage readers to refer to more extensive surveys and background coverage.[21,22] Volume rendering is a class of algorithms that are used to visualize 3D or higher dimensional grids of data. Many of the algorithms are based upon the light interaction with modeled particles in a volume. Early particle models were adapted by Blinn[23] and extended by Kajiya et al.[24] to nonhomogeneous densities of particles. The common algebraic technique for combining ray samples is compositing derived separately by Blinn[23] and Porter et al..[25] Modern direct volume rendering algorithms are largely based on Levoy,[26] Drebin et al.,[1] Sabella,[27] and Upson et al.[28]

The rendering of grids through cell based approaches is traceable from early in the development of volume rendering. Scan conversion of surface and volume primitives into voxels was done by Kaufman.[29] Rectilinear grids were rendered with a cell based method by Upson et al.[28] Similar to Upson et al.,[28] Wilhelms et al. researched rendering of regular grids as hexahedral (six sided or cube) cells.[30] Drawing voxels by drawing entire row/column planes as sheets was done by Hibbard et al.[31] This approach is also referred to as Brick of Bytes (BOB).[32] Then in parallel a flurry of developments for cellular based rendering and ray tracing of irregular grids was done at the Volume Visualization Symposium.[33,9,34] Peter Williams developed extensions to Shirley et al.[9] tetrahedral renderer, and provided simplified cell representations to give greater interactivity.[8]

A related approach is to consider the volume data not as cells, but as faces, and then process by scanlines, saving on having to break up cells into unique front and back faces as done by Lucas.[35] This works well for irregular gridded data. The alternative is adapting the ideas taken from the cell based rendering of regularly gridded data such as the extensions to Van Gelder and Wilhelms earlier work[36] which uses hexahedral irregular data instead of the tetrahedral cells used in Shirley et al..[9] Ray tracing has been used for rendering of irregular data as well.[34] Parallel rendering of irregular grids has been pursued.[37,38] Most recently there has been work on multiresolutional irregular volume representations[39] distributed volume rendering algorithms on massively parallel computers.[40] And, optimization using texture mapping hardware,[41] optimizing sweep plane algorithms,[42] and optimal isosurface extraction.[43,44] Software scan conversion implementations are also being researched.[45,6]

The rendering of unstructured grids can be done by supporting cell primitives, at a minimum tetrahedra, which when drawn in the appropriate order, can be accumulated into the frame buffer for volume visualization. There is a disadvantage of using only tetrahedral cells to support the hexahedral and other cells, mainly a 5x explosion in data

as a hexahedral cube requires 5 tetrahedra at a minimum to be represented, and the tiling of adjacent cells when subdividing may cause problems in matching edges including cracking, where a small crack may appear between hexahedral cells that have been subdivided into tetrahedra.[9]

For proper compositing either the cells viewpoint ordering can be determined from the grid or a depth sorting is performed. For regular grids the viewpoint ordering is straightforward. For unstructured data, the viewpoint ordering must be determined for each viewpoint by comparing the depth of cells. Schemes that use cell adjacency data structures can help to reduce the run time complexity of the sort.[10,9]

The basic approach to compute cell's contributions is three dimensional scan conversion. Several variants were investigated in Wilhelms and Van Gelder.[30] The first variant is averaging opacity and color of front and back faces then using a nonlinear approximation to the exponent. The second variant is averaging opacity and color of front and back faces, then exactly computing the exponential. And, the third variant is, using numerical linear integration of color and opacity between front and back faces, and exactly computing the exponentials. Gouraud shading hardware is used by interpolating colors and opacities across faces which is possible if the exponential approximation is evaluated at the vertices.[9]

4. PROPOSED ALGORITHM FOR PIXELFLOW

There are several strategies for rendering of irregularly gridded data on a composition network. The primary differences are in how the data set is visibility sorted. To volume render, volume cells must be composited in the correct order. This places a requirement on either the distribution of the data across graphics pipelines or on the combining of pipeline's results. In surface based graphics, there are similar tradeoffs, and the most popular technique (currently) is a brute force depth comparison technique of z-buffering. To apply the same approach to volume graphics would require a complete volume frame buffer, and memory costs and bandwidth are not low enough for this approach. Rendering of transparent geometry faces similar challenges. If only a single graphics pipeline is applied, this is sufficient, but limits the parallelism to pipeline and pixel/primitive parallelism. With PixelFlow there are separate graphics pipelines that compute on separate primitives, Figs. 4 and 11. There are three alternatives for sorting in irregular gridded volume rendering on a compositing network.

1. Object space partitioning: render regions and sort layers.
2. Cell partitioning: render and sort cells.
3. Cell partitioning no sorting: render cells and combine by adding.

Alternative 1, a sort first plus sort last approach, is the best candidate for PixelFlow. Alternative 2, a sort last approach, has the potential for the highest efficiencies, but cannot easily be implemented in PixelFlow which does not directly implement alpha compositing, and composites screen regions and not primitives. Alternative 3, a sort almost approach, may provide very high performance for x-ray, maximum intensity, and other projections. Alternative 1 is the best fit to PixelFlow, because the object space partition results in image layers and provides complete distribution of the data. Figure 8 shows a 2D volume subdivided into four object space regions. Consider each region to be assigned to a separate geometry, rasterization, rendering pipeline. Figure 8 also shows the shared volume primitives as those inlined by the color code of the region: R0 (red), R1 (blue), R2 (green), and R3 (turquoise). With different data distributions different object space partitionings may be used, such as BSP (binary space partitioning) trees or K-d trees.

As primitives, or tetrahedra are rendered within each object space region, they are clipped to their region. Clipping planes are defined at the six boundaries of the object space. Figure 10 shows how a tetrahedra shared across the partition contributes its effects without explicitly calculating new primitives. The depth through the primitive goes outside of the object space region on the left, so that the primitive's full volume is taken into consideration. This is a new approach, and avoids creating new cell primitives, is correct, and simple to implement. It also avoids a costly resorting as used by Ma and Crocket.[46]

A parallel version of the algorithm given in Sect. 1 is given below. Figures 8 and 11 show the object space subdivision used for parallelization. As before we process the volume dataset to divide cells into tetrahedra, if necessary, classify data values to colors and opacities, create plane equations for each cell face, and compute circumscribing

```
I. Preprocess dataset, object space distribute dataset across renderers
For a new viewpoint:
II. View sort cells, within each renderer object region
   For every cell in sorted back-to-front order:
   III. evaluate cell plane equation using eye position to determine class
   IV. Split into 1 to 4 triangles and determine new vertex
   V. Compute color and opacity for thick vertex
   VI. Scanconvert new triangles into local renderer's layer
VII. Final composite of object space layers on composition network
```

Figure 6. Parallel projected tetrahedra algorithm Pseudo-code

spheres for each tetrahedra. The plane equations are used to determine a cell's classification as shown in Fig. 3. The circumscribing spheres are used to view sort the tetrahedra, as described by Cignoni et al.[47]

The algorithm sorts (step II) and renders (step III, IV, V, VI) each object space region independently. A final compositing (step VII) combines the object space layers. The layers may be combined as we showed earlier in Wittenbrink and Somani,[19] if alpha compositing of layers is done by reading data off of the composition network and into the SIMD processors. The final compositing of layers may even be done sequentially on PixelFlow because of the high bandwidth of the compositing network (100Gbit/s). Circumscribing spheres are used, because a numerical sort of the tangential distance to the sphere from the eye point, gives a correct topological view ordering of the cells for a Delaunay triangulation. Cignoni et al.[47] showed this sort to be superior to the adjacency list depth first directed acyclic graph search.[10] Figure 5 shows the primary shortcoming, that if the tetrahedralization is not a Delaunay triangulation, then the sort may not be correct. The pathological case of a subdivided regular grid cell, gives five tetrahedra that all have the same circumscribing sphere. Figure 5 shows two such tetrahedra. Figure 7 shows shows spheres and tetrahedra for the first 300 primitives in the NASA Langley fighter data set, from the same rendering perspective as that given in Fig. 9, lower row.

Figure 11 shows the physical and virtual layout of a 9 board system. The daisy chained composition network allows all boards to communicate screen regions to each other. The proposed algorithm needs only the downstream connections for compositing. The object space regions shown would be mapped to each renderer. The primitives crossing boundaries would be shared as Fig. 8 shows. And upon completion of all cells within all renderers, regions would be sent to the last board in the chain to composite. Because of the image region processing of PixelFlow, the rendering and final compositing may be pipelined to be fully overlapped. More details are presented in Sect. 5.

I have implemented an OpenGL version of the projected tetrahedra algorithm, and worked with a PixelFlow software simulator. PixelFlow runs OpenGL, but fast support of the proposed algorithm requires rewriting some of the lower layer software which has not been completed. As part of the performance characterization, and proof of concept I include the following renderings. Figure 9 shows two volume rendered views of the NASA Langley fighter dataset. The views on the left show the defined surfaces in the dataset (jet green, half plane red, half cylinder purple), and the vertices of all tetrahedra as yellow. The volume renderings are shown on the right of the density field, mapped from blue intensities 0 to 1 and alpha 0 to 1. The spheres in Fig. 7 are from the same view as Fig. 9 lower row. The data set has 13,832 datapoints, 70,125 tetrahedra, and the surfaces shown have 7,262 triangles. The figure was rendered using HP's software OpenGL, on an Hewlett-Packard J210 workstation. Volume rendering of the same dataset (1280x1024 resolution) on an HP UX C240 with an FX-6 OpenGL hardware accelerator provides about 2 frames/second; an HP Kayak Windows-NT PC (300 MHz Pentium II) with an FX-4 OpenGL hardware accelerator provides about 1 frame/second; and an HP Vectra XU (200 MHz Pentium Pro) with a Diamond FireGL 1000 OpenGL hardware accelerator provides about 10 seconds/frame.

5. PERFORMANCE ESTIMATES ON PIXELFLOW

In order to fully evaluate the proposed algorithm's performance direct implementation and performance measurements would be ideal. No direct run time results of unstructured grid volume rendering on PixelFlow are available at time of

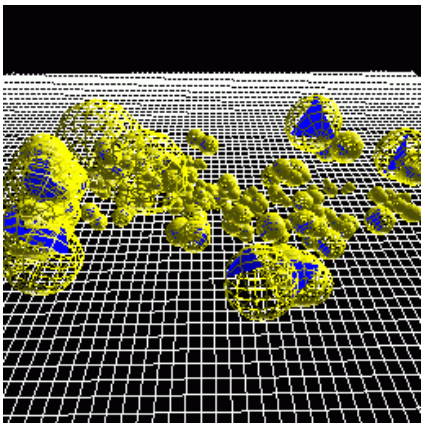

Figure 7. Circumscribing spheres of tetrahedra.

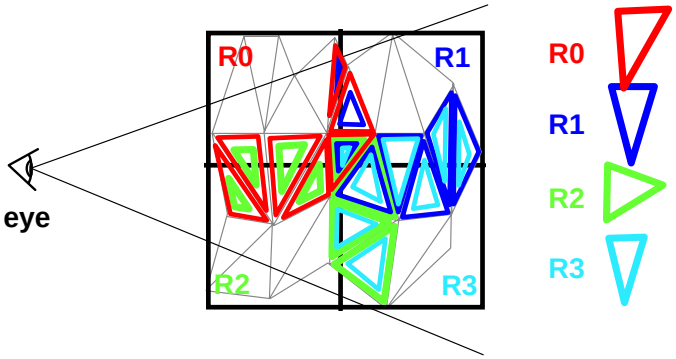

Figure 8. Object space subdivision of tetrahedra.

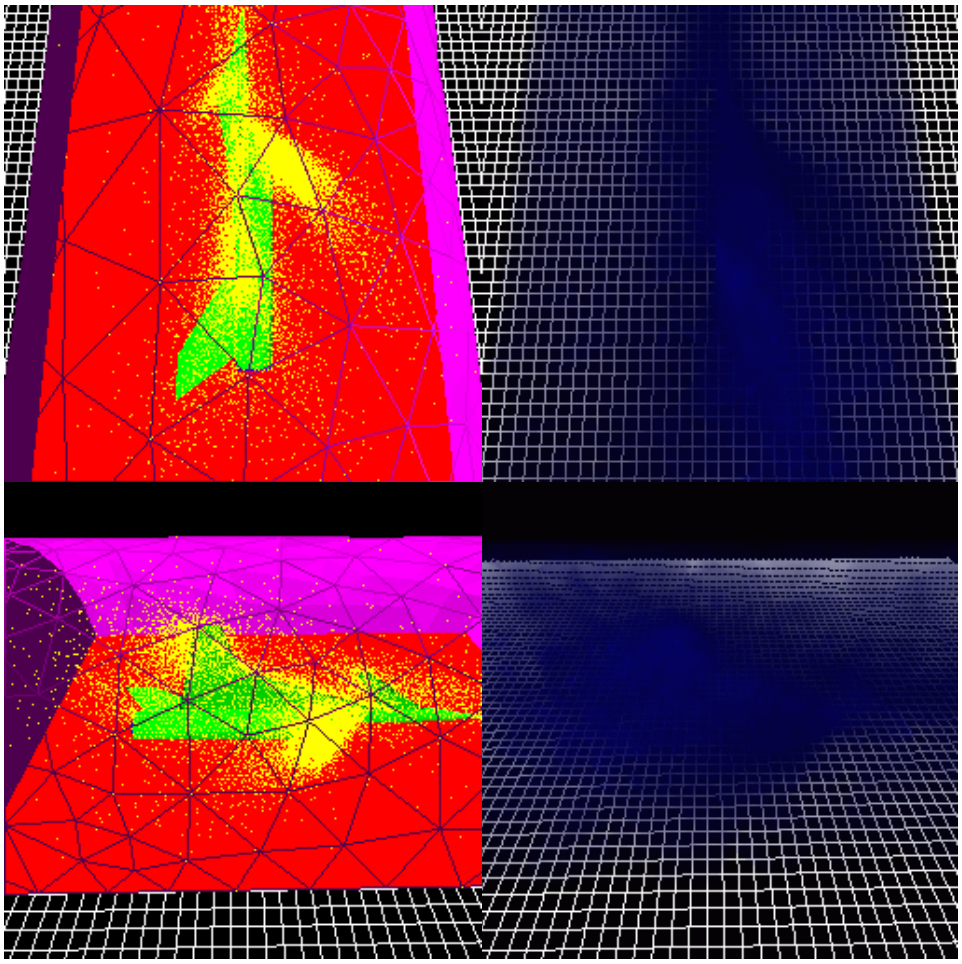

Figure 9. Volume rendering of the NASA Langley fighter, left, surfaces, right, volume rendering.

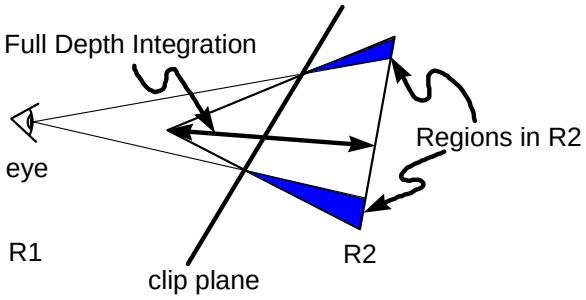

Figure 10. Clipping of a projected tetrahedra.

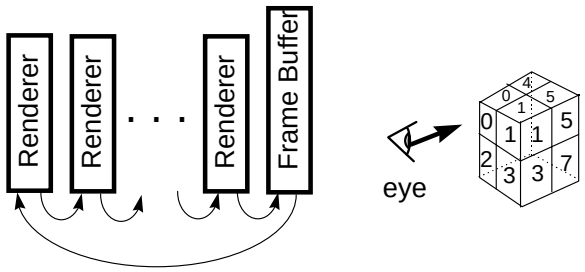

Figure 11. PixelFlow compositing network.

publication. Therefore a characterization of the expected performance is developed. I take into account the possible bottlenecks throughout the system, using the performance as specified by Eyles et al.[11]

PixelFlow prototypes with up to 36 renderer boards have been integrated. A chassis has 9 boards, and a 4 chassis system has 36 boards. Figure 4 and Figure 11 show schematic views of the architecture. Possible bottlenecks are host to system transfer, geometry processing, rasterization processing, or compositing network bandwidth. Key performance characteristics from Eyles et al.[11] are the rasterizer/renderer throughput (2.8 million Gouraud shaded triangles/second or 1.4 million textured Phong shaded triangles/second), and the composition network throughput 2x5.96 Gbytes/second. PixelFlow has also demonstrated performance of 43 million triangles/second on 36 boards (SIGGRAPH'97 demonstrations).

Initially a pairwise compositing approach that takes on the order of $\log R$ for R renderers was examined. But, because of the complexity of implementing new backplane rendering recipes, alternatives were investigated. The simplest alternative is to simply send the results of each renderer to a single shader board that composites the layers together. Because one board would be receiving all of the data, only 1/2 of the compositing network bandwidth would be used. But to transmit a typical resolution of 1280x1024 (160, 64x128 screen regions on PixelFlow) would take just:

$$\frac{6.25 \times 2^{20} \text{ bytes/image}}{5.96 \times 2^{30} \text{ bytes/second transfer rate}} = 0.001 \text{ seconds} \tag{1}$$

So, at 1 millisecond per image layer, all image layers for a 36 board system could be sent in 0.036 seconds or 27 frames/second. For different architectures, obviously the network may not be so powerful (such as the VC-1), and therefore a pairwise combining scheme can be implemented even with a static object space partitioning. Note, that the composition network would be used only for transfer of the data, and the blending would occur on the SIMD array processors of the designated board.

How many primitives can be rendered at the 27 frames/second rate is now dependent upon the throughput of the renderers. If each renderer can sustain 2 million triangles/second,[11] then a 36 board system could achieve 72 million triangles/second. Unfortunately a tetrahedra results in several triangles. In rendering of the Nasa Langley fighter, the percentages of tetrahedral classes are as follows for a collection of views spinning about the dataset: class 1, 34 %, 3 triangles, class 2, 64 %, 4 triangles, class 3, 1 %, 2 triangles, class 4, 0%, 1 triangle. This results in an average of 3.64 triangles/tetrahedra. Because performance varies with features, a factor of 2 slowdown may be expected for alpha compositing triangles, to result in 1 million tetrahedra/second for interactive rendering on a board. For 30 frames/second rendering a board would process

$$(\frac{1 \text{ million triangles}}{\text{second}})(\frac{1 \text{ tetrahedra}}{3.64 \text{ triangles}}) = 277,777 \frac{\text{tetrahedra}}{\text{second} \times \text{board}} \tag{2}$$

for interactive rendering. The throughput of a 36 board system is the following

$$(\frac{277,777 \text{tetrahedra}}{\text{second} \times \text{board}})(\frac{36 \text{ boards}}{\text{system}}) = 9.9 \frac{\text{million tetrahedra}}{\text{second} \times \text{system}} \tag{3}$$

This rate could be driven up even higher by directly scan converting tetrahedral primitives using the planar face equations. At most, PixelFlow would achieve 72 million tetrahedra/second, but there is a factor of 3 possible speedup by adding a new primitive to PixelFlow's programmable rasterizer library.

An additional bottleneck to be considered is the numerical sorting of primitives based on view. Sorting is a task in addition to the geometry processing of each primitive. PixelFlow has 2 PA RISC 8000 processors (180MHz) on each renderer board. These would need to be used for computing Step II of the algorithm. I have benchmarked a quicksort on the 70,125 cell NASA dataset on a PA RISC 7200 (120 MHz) workstation to be 109,570 cells/second. Sorting that takes advantage of view coherency may reduce this cost, and the dual PA 800's may provide a 3X speedup. This would provide sorting of 328,711 cells/second exceeding the cell rendering rate of EQ 2. second quicksort of the 9157 tetrahedra, a 35 f/s rate. If the tetrahedral primitive were added, then the rasterization may no longer be the bottleneck, and, the sorting approach would have to be improved. I am investigating improved sorting approaches.

Load balancing on PixelFlow is achieved through round robin distribution of primitives to renderers. So that the object space subdivision, sort first approach, defeats the load balancing. Object space partitioning needs to be empirically evaluated for datasets of interest, and further tuning is likely necessary to achieve the performance estimates especially considering the screen bucket and screen region processing that PixelFlow uses.

6. CONCLUSIONS

The challenge pursued in this paper is interactive volume rendering of unstructured data. Brute force parallelism is one way to create interactivity, but there are numerous difficulties including fast view sorting, using a sort last architecture with sort first and sort last, effectively load balancing, and providing partitioning and combining approaches. PixelFlow has been demonstrated to be the highest performance scalable graphics computer, but for opaque geometry. Transparency creates problems. Because PixelFlow's compositing network does only z-comparisons and not alpha compositing, and composites only screen regions and not primitives, I used an approach of object space subdivision. The object space subdivision requires properly handling boundary conditions and compositing of layers. I discussed how to handle this with clipping planes, replication of primitives that cross boundaries, and post compositing of image layers from each renderer.

PixelFlow's composition network is of such high performance that a simple sequential send of renderer's frames to a single compositing board still achieves 27 frames/second on a system of 36 boards. Performance estimates are 10 million tetrahedra/second on a 4 chassis, 36 renderer board system, and 2.5 million tetrahedra/second on a 1 chassis 9 renderer board system at 27 frames/second update rate. Alternatively a 300,000 tetrahedral dataset may be viewed at 30 frames/second.

The contributions of this paper are the examination of unstructured rendering with parallel special purpose graphics rendering hardware, the proposal to use a sort first-sort last, hybrid approach on PixelFlow, the object space boundary clipping planes characterization, and a performance estimate for PixelFlow. The clipping of volume primitives to object space regions is a new and efficient approach. Performance timing studies will be possible with availability of PixelFlow hardware. And, this approach should also be directly applicable to other compositing graphics computers.

ACKNOWLEDGEMENTS

I would like to thank the assistance of the PixelFlow development team, especially Brad Ritter, Maureen Hoffert, and Stephen Molnar. I would also like to thank Sam Uselton for help on the data sets, and support of my Manager Tom Malzbender in this effort. The Langley fighter dataset is provided by Robert Neely and Jack Betina of Langley Research Center, NASA.

REFERENCES

1. R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *Computer Graphics*, pp. 65–74, Aug. 1988.
2. P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," in *Proceedings of SIGGRAPH 94*, pp. 451–458, ACM, (Orlando, FL), July 1994.

3. A. Van Gelder and K. Kim, "Direct volume rendering with shading via 3D textures," in *ACM/IEEE Symposium on Volume Visualization*, (San Francisco, CA), October 1996. See also technical report UCSC-CRL-96-16.
4. H. Pfister and A. Kaufman, "Cube-4-a scalable architecture for real-time volume rendering," in *Proceedings 1996 Symposium on Volume Visualization*, pp. 47–54, (San Francisco, CA), Oct. 1996.
5. G. Knittel, "A PCI-based volume rendering accelerator," in *Proceedings of the 10th Eurographics Hardware Workshop*, pp. 73–82, (Maastricht, NL), Aug 1995.
6. J. Wilhelms, A. V. Gelder, P. Tarantino, and J. Gibbs, "Hierarchical and parallelizable direct volume rendering for irregular and multiple grids," in *Proceedings of Visualization*, pp. 57–64, (San Francisco, CA), Oct. 1996.
7. J. Wilhelms and A. V. Gelder, "A coherent projection approach for direct volume rendering," in *Proceedings of SIGGRAPH 91*, pp. 275–284, 1991.
8. P. L. Williams, "Interactive splatting of nonrectilinear volumes," in *Proceedings Visualization*, pp. 37–44, (Boston), Oct. 1992.
9. P. Shirley and A. Tuchman, "A polygonal approximation to direct scalar volume rendering," in *1990 Workshop on Volume Visualization*, pp. 63–70, (San Diego, CA), Dec. 1990.
10. P. L. Williams, "Visibility ordering meshed polyhedra," *ACM Transactions on Grahpics* **11**(2), pp. 103–126, 1992.
11. J. Eyles, S. Molnar, J. Poulton, T. Greer, A. Lastra, N. England, and L. Westover, "PixelFlow: The realization," in *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pp. 57–68, ACM, (Los Angeles, CA), Aug. 1997.
12. S. Molnar, J. Eyles, and J. Poulton, "Pixelflow: High-speed rendering using image composition," in *Proceedings of SIGGRAPH '92*, pp. 231–240, ACM, (Chicago, IL), July 1992.
13. T. Malzbender, "Apparatus and method for volume rendering." US Patent 5,557,711, 1996.
14. S. Nishimura and T. L. Kunii, "VC-1: A scalable graphics computer with virtual local frame buffers," in *Proceedings of SIGGRAPH '96*, pp. 365–372, ACM, (New Orleans, LA), Aug. 1996.
15. B. Baumle, P. Kohler, and A. Gunzinger, "Interactive parallel rendering on a multiprocessor system with intelligent communication controllers," in *Proceedings of 1995 Parallel Rendering Symposium*, pp. 89–95, IEEE, (Atlanta, GA), Oct. 1995.
16. A. Levinthal and T. Porter, "Chap-a SIMD graphics processor," in *Computer Graphics (ACM Siggraph Proceedings)*, pp. 77–82, 1984.
17. H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-Planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories," in *Computer Graphics (ACM Siggraph Proceedings)*, vol. 23, pp. 79–88, 1989.
18. J. Torborg and J. T. Kajiya, "Talisman: Commodity realtime 3D graphics for the PC," in *Proceedings of SIGGRAPH*, pp. 353–363, ACM, (New Orleans, LA), Aug. 1996.
19. C. M. Wittenbrink and A. K. Somani, "Time and space optimal parallel volume rendering using permutation warping," *Journal of Parallel and Distributed Computing* **In Press**, 1997. Available as Technical Report, Univ. of California, Santa Cruz, UCSC-CRL-96-33.
20. K. L. Ma, J. Painter, C. D. Hansen, and M. F. Krogh, "Parallel volume rendering using binary-swap compositing," *IEEE Computer Graphics and Applications* **14**(4), pp. 59–67, 1994.
21. A. Kaufman, ed., *Volume Visualization*, IEEE Computer Society Press, Los Alamitos, CA, 1991.
22. N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics* **1**, pp. 99–108, June 1995.
23. J. Blinn, "Light reflection functions for simulations of clouds and dusty surfaces," in *Computer Graphics*, pp. 21–29, July 1982.
24. J. T. Kajiya and B. V. Herzen, "Ray tracing volume densities," in *Proceedings of SIGGRAPH*, pp. 165–174, July 1984.
25. T. Porter and T. Duff, "Compositing digital images," in *Computer Graphics*, pp. 253–259, Aug. 1984.
26. M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications* **8**, pp. 29–37, May 1988.
27. P. Sabella, "A rendering algorithm for visualizing 3d scalar fields," in *Computer Graphics*, pp. 51–58, Aug. 1988.
28. C. Upson and M. Keeler, "V-buffer: Visible volume rendering," in *Computer Graphics*, vol. 22, pp. 59–64, July 1988.

29. A. Kaufman, "Efficient algorithms for 3D scan-conversion for parametric curves, surfaces, and volumes," in *Computer Graphics (ACM Siggraph Proceedings)*, vol. 21, pp. 171–179, July 1987.
30. J. Wilhelms, "Decisions in volume rendering." In SIGGRAPH'91 Course Notes 8, July 1991.
31. W. Hibbard and D. Santek, "Interactivity is the key," in *Chapel Hill Workshop on Volume Visualization*, pp. 39–43, Department of Computer Science, UNC Chapel Hill, May 1989.
32. K. Chin-Purcell, "Minnesota supercomputer center, brick of bytes (bob) volume renderer." http://www.arc.umn.edu/gvl-software/bob.html, 1992.
33. N. Max, P. Hanrahan, and R. Crawfis, "Area and volume coherence for efficient visualization of 3D scalar functions," *ACM Computer Graphics (Proceedings of the 1990 Workshop on Volume Visualization)* **24**(5), pp. 27–33, 1990.
34. M. P. Garrity, "Raytracing irregular volume data," in *Symposium on Volume Visualization*, (San Diego, CA), November 1990.
35. B. Lucas, "A scientific visualization renderer," in *Proceedings of Visualization*, pp. 227–234, IEEE, Oct 1992.
36. A. V. Gelder and J. Wilhelms, "Rapid exploration of curvilinear grids using direct volume rendering," in *Proceedings of Visualization*, pp. 70–77, 1993. Available as University of California, Santa Cruz Technical Report UCSC-CRL-93-02.
37. J. Challinger, "Scalable parallel volume raycasting for nonrectilinear computational grids," in *Proceedings of the Parallel Rendering Symposium*, pp. 81–88, IEEE, Oct 1993.
38. C. Giertsen and J. Petersen, "Parallel volume rendering on a network of workstations," *IEEE Computer Graphics and Applications* **13**(6), pp. 16–23, 1993.
39. P. Cignoni, L. D. Floriani, C. Montani, E. Puppo, and R. Scopigno, "Multiresolution modeling and visualization of volume data based on simplicial complexes," in *Proceedings of the Symposium on Volume Visualization*, pp. 19–26, 1994.
40. K.-L. Ma, "Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures," in *Proceedings of the Parallel Rendering Symposium*, pp. 23–30, IEEE, (Atlanta, GA), Oct 1995.
41. R. Yagel, D. M. Reed, A. Law, P.-W. Shih, and N. Shareef, "Hardware assisted volume rendering of unstructured grids by incremental slicing," in *ACM/IEEE Symposium on Volume Visualization*, pp. 55–62, (San Francisco, CA), October 1996.
42. C. T. Silva, J. S. Mitchell, and A. E. Kaufman, "Fast rendering of irregular grids," in *ACM/IEEE Symposium on Volume Visualization*, pp. 15–22, (San Francisco, CA), October 1996.
43. P. Cignoni, C. Montani, E. Puppo, and R. Scopigno, "Optimal isosurface extraction from irregular volume data," in *ACM/IEEE Symposium on Volume Visualization*, pp. 31–38, (San Francisco, CA), October 1996.
44. H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson, "Isosurfacing in span space with utmost efficiency (issue)," in *Proceedings of Visualization*, pp. 287–294, (San Francisco, CA), Oct. 1996.
45. P. Tarantino, "Parallel direct volume rendering of intersecting curvilinear and unstructured grids using a scan-line algorithm and k-d trees," Master's thesis, University of California, Santa Cruz, June 1996.
46. K.-L. Ma and T. W. Crockett, "A scalable parallel cell-project volume rendering algorithm for three-dimensional unstructured data," in *Proceedings of the Parallel Rendering Symposium*, pp. 95–104, IEEE, (Phoenix, AZ), Oct. 1997.
47. P. Cignoni, C. Montani, D. Sarti, and R. Scopigno, "On the optimization of projective volume rendering," in *Proceedings of the Eurographics Workshop, Visualization in Scientific Computing'95*, R. Scaneni, J. van Wijk, and P. Zanarini, eds., pp. 59–71, (Chia, Italy), May 1995.