CellFast: Interactive Unstructured Volume Rendering

Craig M. Wittenbrink

Hewlett-Packard Labs
1501 Page Mill Road
Palo Alto, CA 94304

Abstract

Shirley and Tuchman's projected tetrahedra approach is a fast algorithm for unstructured volume visualization, because it generates triangles that may be rendered by hardware acceleration. In this paper, I explore optimizations using OpenGL triangle fans, customized quicksort, memory organization for cache efficiency, display lists, and tetrahedral culling. The optimizations improve the performance of projected tetrahedra rendering to provide 1 frame/second for 240,122 tetrahedral cells, 3 frames/second for 70,125 tetrahedral cells, and 15 frames/second for 12,936 tetrahedral cells. Resolutions of 3840x1024 were rendered by using the HP-Visualize Center at full frame rates.

Keywords: tetrahedral cell, irregular grid, volume visualization, OpenGL, visibility sorting, quicksort, triangle fan

1 INTRODUCTION

Interactive volume rendering has always been a challenging problem. This paper is all about making unstructured rendering as interactive as possible on available hardware platforms. The projected tetrahedra algorithm of Shirley and Tuchman [5] uses the geometry acceleration and rasterization of polygon rendering to maximum advantage. The majority of the work, the scan conversion and compositing, are accelerated by existing graphics hardware. In earlier work, I investigated hardware acceleration with parallel compositing on PixelFlow [10]. OpenGL hardware acceleration is now widely available in desktop systems, and using the earlier implementation as a starting point I investigated further optimizations to improve desktop rendering performance. Figure 1 provides pseudo-code of Shirley and Tuchman's [5] projected tetrahedra algorithm, where tetrahedra are projected at the screen plane, and subdivided into triangles. Figure 3 shows the four classes of projections that result in one to four triangles.

Preprocessing is necessary to calculate colors and opacities from input data, setup for visibility sorting of primitives, and creation of plane equations used for determining the class a tetrahedra belongs to for a viewpoint. Cells are visibility sorted (step II) for proper compositing for each viewpoint [9]. Figure 3 shows that new vertices are introduced during steps III and IV. The new vertex requires a color and opacity to be calculated in step V. Then the triangles are drawn and scan converted in step VI. Figure 2 shows the output of my renderer for the Phoenix, NASA Langley Fighter, and F117 datasets.

```
I. Preprocess dataset
For a new viewpoint:
II. Visibility sort cells
   For every cell in sorted back-to-front order:
   III. Test plane equations to determine class (1,2,3,4)
   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 pseudo-code

In order to achieve the highest possible performance for interactive rendering of unstructured data, many software optimizations are necessary. In this paper, I explore software optimizations using OpenGL triangle fans, customized quicksort, memory organization for cache efficiency, display lists, and tetrahedral culling. The optimizations can vastly improve the performance of projected tetrahedra rendering to provide interactive rendering on datasets of hundreds of thousands of tetrahedra. These results are an order of magnitude faster than the best in the literature for unstructured volume rendering [11, 7]. The sorting is also an order of magnitude faster than the fastest sorting timing reported for Williams MPVONC [2].

2 OPTIMIZATIONS

Triangle strips. Triangle strips can greatly improve the efficiency of an application. Creating triangle strips from triangular meshes of data can be done through greedy algorithms. But, in projected tetrahedra, the split cases illustrated in Figure 3 can directly create triangle fans. Each new vertex given with *glVertex* specifies a new triangle, instead of redundantly passing vertices. Figure 3 gives triangle strips for the four classes of projections.

In experiments on the NASA Langley fighter dataset, with 70,125 tetrahedra, there are an average of 3.4 triangles per tetrahedra (a sum of the percentage of Class 1–60% 3 triangles, 2–40% 4 triangles, 3, and 4). Fewer vertices are transmitted for the same geometry and many fewer procedure calls are made to OpenGL. For n tetrahedra there are $10.8n$ procedure calls with fans versus $20.4n$ procedure calls without fans, a factor of 2 reduction. Williams also investigated triangle stripping using Iris-GL [8].

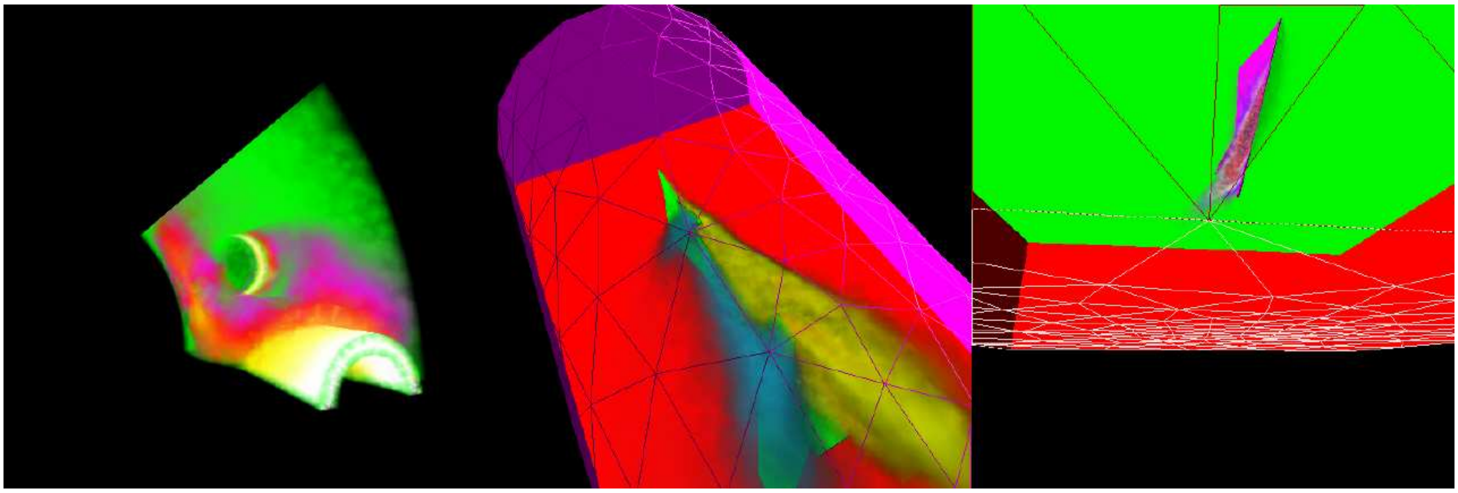

Figure 2: Unstructured volume rendering of Phoenix (left), NASA Langley Fighter (middle), and F117 (right).

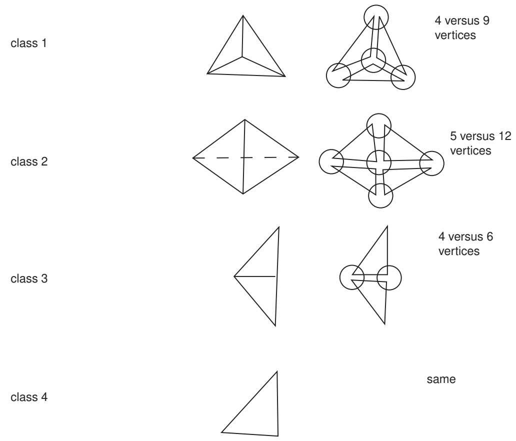

Figure 3: Number of vertices for fan versus triangles.

Display Lists For Static Geometry. Display lists allow the graphics drivers to optimize vertices, colors, and triangle strips for the hardware. I converted all static geometry into display lists. Figure 4 shows examples of vertices, surfaces, and a background mesh. The primary impact of these changes is to eliminate any slowdown when these auxiliary data are also rendered. Unfortunately, the projected tetrahedra recomputes the thick vertex for every new viewpoint so that the volume data cannot be placed in display lists. Also, because the thick vertex is recalculated for each new view, vertex arrays cannot be used.

Visibility Sorting—Customized Quicksort. Visibility sorting of cells is done through Cignoni et al. and Karasick et al.'s [1, 3] technique of sorting the tangential distances to circumscribing spheres. I have compared a custom coded quicksort to the C library utility *qsort()* and found an improvement of 75% to 89%. A generic sorting routine cannot as efficiently handle the data structures that are to be sorted. Table 1 shows the custom coded quicksort, derived from [4], and *qsort* performance. Numbers were captured on an HP J5000 with PA-RISC 8500 at 440MHz. Both the $l2$ value (float) and the tetrahedral index (unsigned long int) are moved.

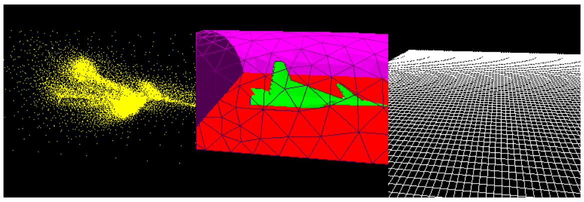

Figure 4: Points (left) surfaces (middle) and mesh (right) stored in display lists.

Taking advantage of view coherence. The proper choice of pivots gives an efficient sorting of sorted and nearly sorted lists. I previously resorted the same input for each view. But, using the sorted values from the previous view speeds up the sorting. The program was also modified to use smaller viewpoint changes, and run times were improved by an additional 18%. Both sorts are much faster on sorted data. This property is exploited for view coherence. The rate for the custom quicksort varies between 600,000 to 2 million cells per second depending on how sorted the list is. For comparison, recently reported results for MPVONC [9] are from 185,000 to 266,000 cells per second [2]. In my earlier work, I showed that quicksort achieved 109,570 cells/second on a PA RISC 7200 (120 MHz) [10], without the the view coherence and data structure optimizations discussed here.

routine	input	10,000	100,000	1 mil.	10 mil.	100 mil.
qsort	random	0.04	0.48	5.71	66.03	755.56
qsort	sorted	0.02	0.24	2.89	35.46	443.94
quicksort	random	0.01	0.13	1.47	16.26	165.82
quicksort	sorted	0.01	0.05	0.58	6.07	48.96
improvement	random	75%	73%	74%	75%	78%
improvement	sorted	50%	79%	80%	83%	89%

Table 1: Sorting times in seconds and percent improvement of custom routine (mil. = million).

Cache coherency. Because the tetrahedral data structures are randomly accessed, a high percentage of time is spent in the first fetch of each tetrahedra's data. Tetrahedra are accessed by their view and not memory storage order. Reordering the tetrahedra when performing the view sort eliminated cache stalls when rendering data, but the sorting routine was slowed down by moving more data. There was an overall slowdown, so future work is needed to find effective caching strategies.

Culling. Tetrahedra whose opacity are zero are removed from sorting and rendering. This is classification dependent, but yields 20% and 36% reduction in tetrahedra for the Langley Fighter and F117 datasets. The runtime decreases accordingly.

3 RESULTS

Recently HP released the J5000 HP-UX workstation. Performance measurements were made on a J5000 with an fx-6 Pro OpenGL graphics accelerator. Table 2 shows the seconds to render a frame and the frame/second for five datasets. The J5000 renders the NASA Langley dataset of 70,125 tetrahedra at 3.0 frames/second. The Kayak XW, PIII 300 MHz/fx-4 renders at 1.1 frames/second, the HP C240/fx-6 renders at 1.5 frames/second, and the HP Visualize X Class, PIII Xeon 550 MHz/fx-6+ renders at 2.2 frames/second. The HP Visualize Center is a three screen projection system driven using three workstations. The number of pixels rasterized is tripled, which enables much more data to be seen. Performance measurements were made on a Visualize Center—three J5000s each with an fx-6 Pro accelerator— and a Visualize Workgroup—a single J5000 with 2 fx-6 Pro accelerators. Table 3 provides the performance in megapixels/second for four datasets, and three resolutions 1280x1024 (J5000), 3840x1024 (Visualize Center), and 2560x1024 (WorkGroup).

In a comparison to related work, the tetrahedra/second rendering rate is more than an order of magnitude faster than those in the literature. The fastest numbers found are those I estimated to be possible on the complete PixelFlow configuration, a 9.9 million tetrahedra/second[10]. The rates in this paper are 199,015 tetrahedra/second on Phoenix, 210,586 tetrahedra/second on Langley Fighter, 248,316 tetrahedra/second on F117, and 123,370-147,026 tetrahedra/second on head and torso. The fastest numbers in the literature for rendering are 25,000 tetrahedra/second by Yagel et al. [11] and Van Gelder et al. [7] who achieve those rates through either coarse cutting of only 50 planes of slices, or by 95% culling of tetrahedra from the dataset. My algorithm averages 7 times better performance, while rendering all cells. Further work is needed to determine a fair comparison amongst the wide number of platforms and implementations used.

data set	culled	tetrahedra	sec	frame/sec
phoenix	0%	12,936	0.065	15.3
langley	20%	70,125	0.333	3.0
f117	36%	240,122	0.967	1.03
torso	0%	1,293,238	8.796	0.11
head	0%	2,443,013	19.801	0.05

Table 2: Rendering performance in seconds and Hz.

data set	J5000	VisCenter	WorkGroup
phoenix	20.16	42.74	29.79
langley	3.94	7.28	4.86
torso	0.15	0.36	0.24
head	0.07	0.16	0.11

Table 3: Performance megapixels per second.

4 CONCLUSIONS

I have shown several optimizations used to achieve interactive rendering of unstructured volume data with the projected tetrahedra algorithm. The principal speedup results from using hardware triangle rasterization, and the use of fast depth sorting. The sorting method was explored by Cignoni et al.

[1] and Karasick et al. [3] and proves to be much faster than other sorting approaches, even though its asymptotic run time complexity is apparently higher ($O(n \log n)$ versus $O(n)$). The tangential distance sort is only correct for Delaunay triangulations. For datasets that are not Delaunay triangulations, a new triangulation can be recalculated to use the fast sorting presented. For constrained Delaunay, vertices may be added so that the constrained is a subset of a Delaunay triangulation. Other optimizations included using OpenGL as efficiently as possible—such as triangle fans—, culling of zero opacity tetrahedra, and caching efficiency. The results show that rendering of large unstructured datasets with projected tetrahedra is truly interactive. Rendering the NASA Langley fighter dataset is interactive on a variety of platforms, where previously published results were 10's of seconds [6, 11, 7] to render similar datasets. Smaller datasets, such as the Super Phoenix nuclear reactor, are rendered at truly interactive rates, 15 frames/second, versus 3.5 seconds per frame for just the sorting in [6]. Additional results are the demonstration of interactive unstructured rendering at very high resolution, such as the 3840x1024 display of the HP Visualize Center. Many software renderers are limited to reduced resolutions such as 500x500 [7].

ACKNOWLEDGEMENTS

Data sets are thankfully acknowledged: for the NASA Langley Fighter, Neely and Batina; for the Super Phoenix Nuclear reactor, Bruno Nitrosso, Electricite de France; for the F117, Robert Haimes, MIT; and for the head and torso datasets, the Center for Scientific Computing and Imaging, University of Utah. I also thank Dean Brederson, Claudio Silva, Vivek Verma, and Peter Williams for help in getting data and classifications.

References

[1] P. Cignoni, C. Montani, D. Sarti, and R. Scopigno. On the optimization of projective volume rendering. In R. Scaneni, J. van Wijk, and P. Zanarini, editors, *Proceedings of the Eurographics Workshop, Visualization in Scientific Computing'95*, pages 59–71, Chia, Italy, May 1995.

[2] J. Comba, J. T. Klosowski, N. Max, J. S. Mitchell, C. T. Silva, and P. L. Williams. Fast polyhedral cell sorting for interactive rendering of unstructured grids. In *Proceedings of Eurographics'99*, Milan, Italy, Sept. 1999.

[3] M. Karasick, D. Lieber, L. Nackman, and V. Rajan. Visualization of three-dimensional delaunay meshes. *Algorithmica*, 19(1-2):114–128, Sept.-Oct. 1997.

[4] A. Kelley and I. Pohl. *An Introduction to Programming in C*. Benjamin Cummings, Menlo Park, CA, 1984.

[5] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. In *1990 Workshop on Volume Visualization*, pages 63–70, San Diego, CA, Dec. 1990.

[6] C. T. Silva, J. S. Mitchell, and P. L. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *ACM/IEEE Symposium on Volume Visualization*, pages 87–94, Research Triangle Park, NC, October 1998.

[7] A. Van Gelder, V. Verma, and J. Wilhelms. Volume decimation of irregular tetrahedral grids. In *Proceedings of Computer Graphics International*, pages 222–230,247, Canmore, Alta., Canada, June 1999.

[8] P. L. Williams. *Interactive Direct Volume Rendering of Curvilinear and Unstructured Data*. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[9] P. L. Williams. Visibility ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, 1992.

[10] C. M. Wittenbrink. Irregular grid volume rendering with composition networks. In *Proceedings of IS&T/SPIE Visual Data Exploration and Analysis V*, volume 3298, pages 250–260, San Jose, CA, Jan. 1998. SPIE. Available as Hewlett-Packard Laboratories Technical Report, HPL-97-51-R1.

[11] 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*, pages 55–62, San Francisco, CA, October 1996.