Title
Parallel Visualization of Large-Scale Aerodynamic Calculations: A Case Study on T3E

Authors
Ma, Kwan-Liu
Crockett, Tom W.

Publication Date
1999

Peer reviewed

Parallel Visualization of Large-Scale Aerodynamics Calculations: A Case Study on the Cray T3E

Kwan-Liu Ma
University of California, Davis

Thomas W. Crockett
Institute for Computer Applications in Science and Engineering

Abstract

This paper reports the performance of a parallel volume rendering algorithm for visualizing a large-scale unstructured-grid dataset produced by a three-dimensional aerodynamics simulation. This dataset, containing over 18 million tetrahedra, allows us to extend our performance results to a problem which is more than 30 times larger than the one we examined previously. This high resolution dataset also allows us to see fine, three-dimensional features in the flow field. All our tests were performed on the SGI/Cray T3E operated by NASA's Goddard Space Flight Center. Using 511 processors, a rendering rate of almost 9 million tetrahedra/second was achieved with a parallel overhead of 26%.

CR Categories and Subject Descriptors: D.1.3 [Programming Techniques]: Concurrent Programming - Parallel Programming; I.3.3 [Computer Graphics] Picture/Image Generation - Viewing Algorithms; I.3.8 [Computer Graphics] Applications; J.2 [Physical Sciences and Engineering].

Additional Keywords: parallel rendering, volume rendering, scientific visualization, parallel algorithms, unstructured grids, computational fluid dynamics, T3E

1 INTRODUCTION

Leading-edge scientific computations with demanding memory and processing requirements are increasingly being performed on massively parallel supercomputers. As an example, researchers at ICASE and NASA Langley Research Center are performing large-scale unstructured mesh computations for three-dimensional high-lift aircraft analysis on state-of-the-art parallel supercomputers such as the Cray T3E and SGI Origin2000 [8]. The computational meshes they are using each contain several million data points. The largest one is a transport take-off configuration which uses up to 24.7 million grid points to derive good predictions of lift and drag for varying angles of attack.

Author's addresses: Kwan-Liu Ma, Department of Computer Science, UC Davis, One Shields Avenue, Davis, CA 95616-8562, ma@cs.ucdavis.edu; Thomas W. Crockett, ICASE, MS 132C, NASA Langley Research Center, Hampton, VA 23681-2199, tom@icase.edu.

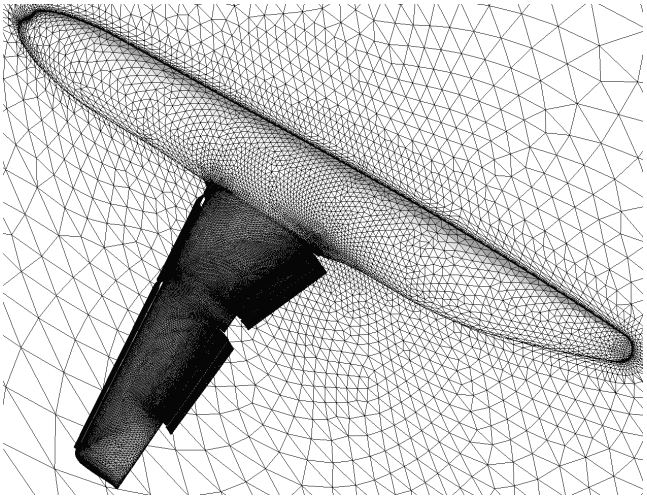

Figure 1. Illustration of the surface grid for a high-lift aircraft configuration. The mesh resolution near the wing surface is particularly fine. Over 90% of mesh elements are in the vicinity of the wing surface.

Visualizing the solution data from this type of calculation is particularly challenging because the associated unstructured meshes are typically large in size and irregular in both shape and resolution. Figure 1 displays a surface mesh used in the high-lift analysis work [8], and Figure 2 shows a close-up view of the same mesh. The corresponding volume mesh would be too cluttered to view directly. Visualizing unstructured-grid data has been an active area of research in recent years [2, 5, 6, 7, 9, 10, 11, 12, 13]. However, interactive performance for high-fidelity visualization of large datasets, such as the high-lift analysis solutions, can only be obtained with the help of parallel computers.

At ICASE, we have been developing a parallel volume renderer for unstructured-grid data. Our design is based on cell-projection rendering and a multiplexed asynchronous communication algorithm. Effective static load balancing is achieved with a round robin distribution of volume data cells among the processors, combined with a fine-grained interleaved partitioning of the image. A spatial partitioning tree is used to ensure locality during the rendering process, thereby improving the performance of the image compositing step and reducing memory consumption. Communication cost is reduced by buffering messages and by overlapping communication with rendering calculations as much as possible.

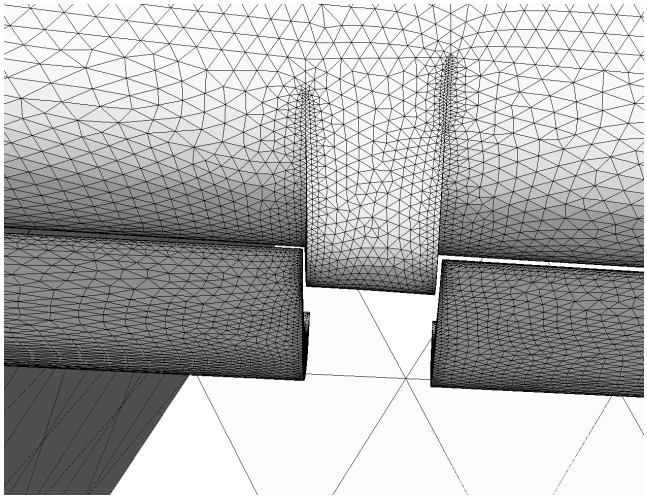

Figure 2. A close-up view of the surface grid for the high-lift configuration, which shows the mesh structure near the two flaps.

In [6], we show that this algorithm scales well with increasing numbers of processors on the IBM SP2. Parallel efficiencies of 70% or better were maintained for up to 128 processors. However, our tests used a relatively small dataset containing only 103,064 data points (567,863 tetrahedra). We did not know if the same algorithm would scale well with data size, or whether increased numbers of ray segments would lead to communication bottlenecks. The size of the high-lift analysis solution data allows us to verify the scalability of our algorithm for larger datasets. This paper presents test results for a dataset containing 3,107,075 grid points (18,216,138 tetrahedra), which is about 32 times larger than the one we used previously. The size of this dataset also allows us to profitably increase the image resolution, providing an opportunity to study performance as a function of image size, and to produce visualization results which reveal fine details in the modeled phenomena.

The rest of the paper is organized as follows. Section 2 reviews the basic parallel rendering algorithm. Section 3 provides a brief overview of the T3E architecture. Section 4 then presents experimental results obtained on the Cray T3E using up to 511 processors.[1] Section 5 illustrates some of the visualization results we have obtained with our renderer, and we conclude this case study in Section 6 with a summary of our results and directions for future research.

2 A PARALLEL VOLUME RENDERING ALGORITHM FOR 3D UNSTRUCTURED-GRID DATA

A more thorough description of our parallel volume rendering algorithm can be found in [6]. In this section, we only highlight the design principles. The basic algorithm performs a sequence of tasks:

- Distributing data and visualization parameters
- Space partitioning
- Viewing transformation
- Scan conversion of tetrahedral cells
- Merging of ray segments
- Assembly and output of final images

The volume data is distributed in round robin fashion with the intention of dispersing nearby cells as widely as possible among processors. This is because the data cells come in different sizes and shapes. The difference in size can be as much as several orders of magnitude due to the adaptive nature of the unstructured mesh. As a result, the projected image area of a cell can vary dramatically, which produces similar variations in scan conversion costs. Cells which are in proximity tend to have similar sizes, so dispersing them helps to average out load imbalances due to cell size.

Once the volume data is distributed, a preprocessing step performs a parallel, synchronized partitioning of the volume data to produce a hierarchical representation of the data space. We use a *k*-d tree [1] because of its ability to adapt to the structure of the data. The *k*-d tree is used in the rendering step to restore locality which is lost in the data distribution step, resulting in more efficient image compositing and reducing runtime memory requirements.

The principal difference between the current algorithm and the one described in [6] is in the image partitioning strategy. Because the types of unstructured grids we deal with can have small regions of very high cell density, we found that our original scanline interleaving scheme exhibited load imbalances as the number of processors approached the number of scanlines. In contrast, the current algorithm uses a very fine-grained pixel interleaving scheme which effectively distributes high density regions over more processors, resulting in better load balancing and improved scalability.

A cell projection method is used to render the volume data. However, cells are not pre-sorted in depth order. Instead, each processors traverses the *k*-d tree in the same fixed order, scan converting its local cells to produce ray segments. The ray segments are then routed to their final destinations in image space for merging. A double-buffering scheme is used in conjunction with asynchronous *send* and *receive* operations to amortize communication overheads and to overlap communication of ray segments with rendering computations. Scan conversion of data cells and merging of ray segments proceed together in multiplexed fashion. When scan conversion and ray-segment merging are finished, each processor sends its completed subimage to a host processor which assembles them for display.

Logically, the scan conversion and image merging operations represent separate threads of control, operating in different computational spaces and using different data structures. For the sake of efficiency and portability, however, we have chosen to interleave these two operations using a polling strategy. Each processor starts by scan converting one or more data cells. Periodically the processor checks to see if incoming ray segments are available; if so, it switches to the merging task, sorting and merging incoming rays until no more input is pending. The resulting communication pattern is both view- and data-dependent, but gen-

[1] Although the Goddard T3E contains 1048 computational processors, the per-job limit is 512. Our current implementation uses one processor to coordinate data distribution and image assembly tasks, leaving a maximum of 511 processors available for rendering computations.

erally requires each processor to communicate with most, if not all, of the other processors.

Due to the asynchronous nature of the rendering algorithm, individual processors are not able to determine on their own when a frame is complete. Hence a distributed termination detection protocol is employed. Our original renderer used a straightforward procedure in which the host processor collected information from each rendering processor and then notified them all when it determined that the overall rendering operation was complete. Our current version improves on this using a binary merging algorithm based on ray-segment counts. The new approach runs in logarithmic, rather than linear, time, and does not involve the host, making it more efficient and scalable to larger numbers of processors.

We have identified a dozen different variables which can affect the performance of this algorithm on any given architecture. Some of these depend on the contents of the input data; others are determined by the viewing and visualization parameters specified by the user; and still others are parameters of the algorithm. We will discuss several of these issues in more detail in Section 4, but first we provide a brief overview of the T3E architecture.

3 SGI/CRAY T3E

All of the tests reported here were performed on the SGI/Cray T3E computer operated by NASA's Goddard Space Flight Center. The T3E is a distributed-memory massively parallel computer system. Although memory is attached directly to each processor (physically distributed), it is globally addressable. In the interest of portability, we have chosen not to exploit this feature directly, preferring instead to rely on MPI message passing for interprocessor communication

In Goddard's T3E, each PE contains a 300MHz DEC Alpha 21164 microprocessor with peak performance of 600 MFLOPS, and 128 megabytes of local memory. About 120 megabytes per processor can be used by the application program. The system as a whole contains 1088 processors, of which 1048 are available for application workloads, with a per-job maximum of 512 PEs. All PEs are connected by a bidirectional 3D torus communication network with peak data bandwidth of 480 megabytes per second in every direction. A recent cross-platform study of a parallel polygon renderer with communication characteristics similar to those of our volume rendering algorithm concluded that the T3E delivered performance which was superior to that of its contemporary competitors [3].

4 TEST RESULTS

To study the scalability of our rendering algorithm, we performed a series of tests using both the large dataset containing 18,216,138 tetrahedra, and the small dataset containing 567,863 tetrahedra. We used the small dataset in our previous study [6] to examine in detail each component of the parallel overhead and to fine tune our algorithm. In this study, we focus particularly on the following parameters:

- number of processors
- data size
- image size

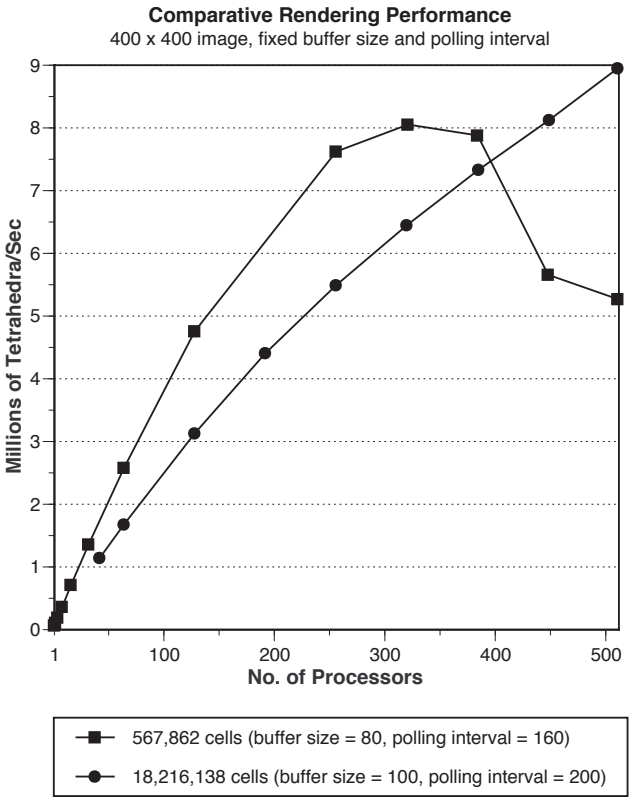

Figure 3. Rendering rates (tetrahedra/second) on the T3E using up to 511 processors.

- ray-segment buffer size
- polling frequency

For each test, three different views were used, and the average rendering time was recorded. Color Plates 1 and 2 show the view sequences used with the small and large datasets respectively. With the current steady-state solutions, data input and distribution is performed only once, and is therefore not included in the rendering time. To expedite our experiments, the large dataset used in our tests has been reduced from double precision to single precision, and contains only a single scalar quantity at each grid point. It occupies 325 megabytes of space on disk; reading and partitioning it among 128 processors requires approximately 33 seconds. We have made no attempt to optimize our image assembly and display procedures, so these times are excluded from the rendering rates as well.

4.1 Performance and Scalability

The first set of test results is summarized in Figure 3. We compare the rendering rates for the large and the small datasets. Because of memory requirements, a minimum of 42 processors is needed to render the large dataset. The plots show that performance with the large dataset increases steadily through 511 processors. The large number of tetrahedral cells entails enough computational load to keep the parallelization overhead manageable. Performance with the small dataset is also good through 256 processors, but peaks around 320 processors and deteriorates beyond that point. With 128 processors, our current implementa-

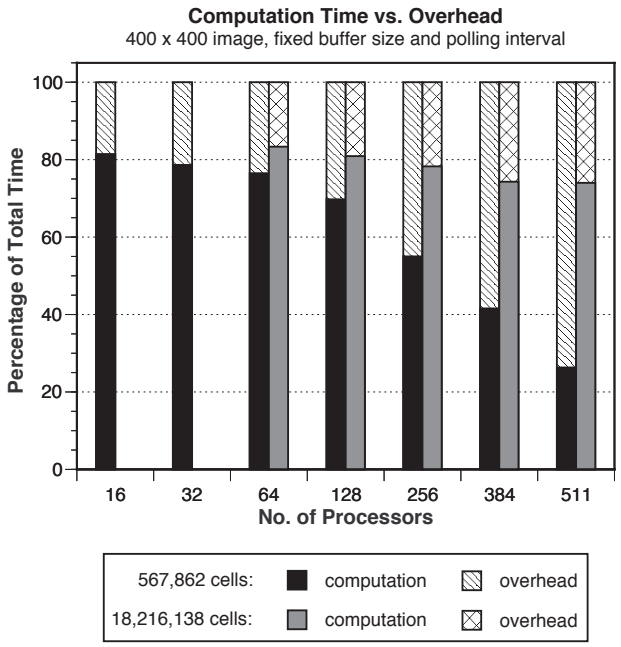

Figure 4. Computation time and parallel overheads for varying numbers of processors. Buffer sizes and polling intervals are the same as in Figure 3.

tion on the T3E renders the small dataset more than 4 times faster than its predecessor on the IBM SP2 [6].

Figure 4 compares the relative contributions of computation and parallel overhead to the total rendering time for varying numbers of processors. Computation includes frame initialization, tree traversal, scan conversion, ray-segment merging, and control flow. Overhead represents additional costs incurred due to the parallel implementation, and includes data copying, communication, termination detection, and idle time due to load imbalance and network congestion. For the large dataset, the overheads comprise about 17% of the time on 64 processors, gradually increasing to 26% of the time with 511 processors. This slow rate of growth suggests that even more processors could be used effectively for rendering datasets of this size. For the smaller dataset, useful computation drops below 50% at about the same point where performance peaks in Figure 3, around 320 processors.

Table 1 provides a more detailed breakdown of the computation and overhead components for the 511-processor case. Ray-segment merging is by far the most expensive operation, costing more than four times as much as scan conversion. Communication costs (data copying, send and receive latencies, polling, and synchronization) seem to be well under control. The dominant overhead appears to be the end-of-frame termination detection protocol, but this is partly an artifact of the way termination time is measured. Once a processor has finished scan converting all of its cells, it enters a polling loop, waiting for either incoming ray segments from other processors, or termination messages. The test for the latter requires a trip through the termination detection routine on each iteration of the loop, so that much of the reported termination cost could be interpreted instead as polling overhead and/or idle time (receive wait). Given this interpretation, the primary overhead then becomes wait time, which is mainly a reflection of load imbalance.

Component	Time (secs)	Percentage
Overheads		
Ray-segment copying	0.040414	1.98
Send latency	0.046632	2.29
Receive latency	0.042881	2.11
Send wait	0.000000	0.00
Receive wait	0.161666	7.94
Polling	0.035943	1.77
Termination detection	0.187978	9.24
Synchronization	0.013291	0.65
Total overhead	*0.528805*	*25.98*
Computation		
Initialization	0.097967	4.81
Scan conversion	0.235613	11.58
Ray-segment merging	0.949170	46.64
Other	0.223697	10.99
Total computation	*1.506447*	*74.02*
Total	***2.035252***	***100.00***

Table 1: Execution time components for 18,216,138 cells on 511 processors at 400 x 400 pixels.

Together, the results in Figures 3 and 4 and Table 1 demonstrate that the T3E is well-equipped to handle the massive communication generated by this application. With the large dataset on 511 processors, an average of more than 44.6 million ray segments have to be communicated per frame. At 24 bytes per ray segment, the aggregate communication volume is about 1.1 gigabytes. One of the advantages of our algorithm is that this load does not get injected into the network all at once, but is spread out over the duration of the frame time, determined in part by the choice of buffer size. Nonetheless, with an advertised bisection bandwidth of 122 GB/s in a 512-processor configuration, it seems unlikely that we would ever tax the communication capabilities of the T3E.

4.2 Image Size

While an image size of 400 x 400 may provide adequate resolution for smaller volumetric datasets, it doesn't do justice to the larger problem considered here. To gauge the impact of higher image resolutions on performance, we have repeated our experiments with image sizes up to 1200 x 1200 pixels. The results are shown in Table 2. As can be seen, performance tracks the increase in image resolution closely. This is to be expected given that the principal execution time components are dependent on the number of ray segments generated. The lower overhead fraction for the 1200 x 1200 case is due to a larger ray segment buffer (1,000 vs. 100), which results in more efficient communication with this high resolution image. The polling interval was set to 200 for all three cases.

	Image Size		
	400 x 400	800 x 800	1200 x 1200
Ray segs. (millions)	44.7	178.0	400.0
Overhead time	1.111	4.355	5.613
Compute time	4.711	16.865	39.721
Total time	5.822	21.220	45.334
Overhead percentage	19.1%	20.5%	12.4%

Table 2: Rendering performance at different image resolutions using 128 processors with the large (18.2 million cell) dataset. Times are in seconds.

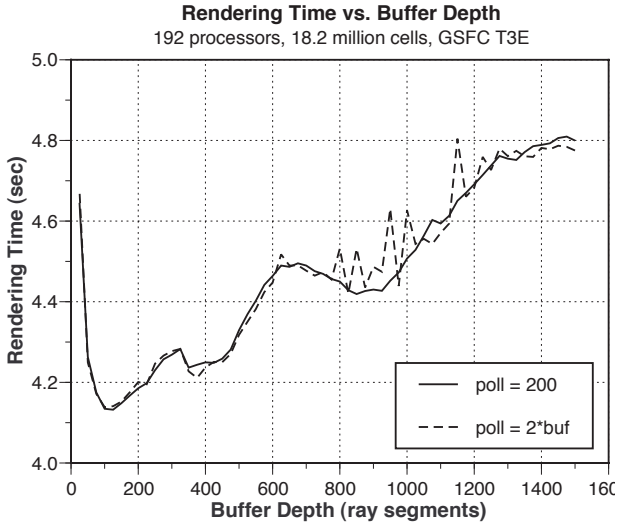

Figure 5. Rendering time as a function of buffer depth for two different polling strategies. Image size is 400 x 400.

4.3 Communication Parameters

As our results from Section 4.2 suggest, the choice of communication parameters (buffer depth and polling interval) can have a significant impact on performance. Although this problem has been studied in some detail in the context of parallel polygon rendering algorithms [4], the situation here is more complex due to the interaction of additional parameters such as the depth of the *k*-d tree and the choice of opacity transfer functions, both of which can have a significant impact on communication and computation performance. Thus guidelines for selecting optimal communication parameters are far from obvious, and a detailed analysis is the subject of ongoing investigation.

Figure 5 displays the impact of buffer depth on performance with the large dataset using 192 processors. The buffer size varies from a minimum of 25 up to 1.25 times the *expected useful maximum*. We define the expected useful maximum as the average number of ray segments which need to be communicated from each processor to every other processor. This value is highly problem-dependent, varying as a function of the input data, viewing parameters, opacity mapping, and number of processors. For this case, the expected useful maximum is empirically determined to be an even 1200.

Normally one expects that as buffer size increases, performance will improve, since message-passing overheads are amortized over more data items. However, using a buffer size which is too large can eliminate the benefits obtained by spreading the communication load over time, particularly on bandwidth-limited systems [4]. For buffer sizes at or above the expected useful maximum, the behavior is equivalent to a simpler algorithm in which all of the communication is deferred to the end of the frame, and the advantages of the asynchronous approach are lost.

This loss of performance with increasing buffer size can be clearly seen in Figure 5, although it appears at smaller buffer depths than would be expected given the high communication bandwidth on the T3E. We suspect that this premature degradation is caused by loss of locality in the ray merging operations. Larger buffers will tend to defeat the purpose of the *k*-d tree by delivering many ray segments at once which fall on a wider area of the image. Fewer opportunities for early ray merging arise, and ray segment lists will grow longer, with a corresponding increase in list insertion time and memory consumption. Note, however, that the vertical scale on the graph has been chosen to highlight the effect—the total variation in performance is only about 16%.

Our experience with parallel polygon renderers indicates that the choice of polling interval is far less critical than buffer depth, and the results here seem to bear that out. The interval between polling operations should be big enough to amortize the cost of the polling call, but beyond that, just about any value will do. This is in part due to the deadlock avoidance properties of our algorithm. If a processor is blocked from sending, it automatically reverts to a receiving mode, whether or not the polling interval has been reached.

Figure 5 also shows the results of two different polling strategies. In the first one, we pick a fixed polling interval of 200, i.e., the renderer will check for incoming data after scan converting 200 cells. In the second strategy, we set the polling interval to twice the buffer size. As can be seen, performance is very similar in either case, although the fixed polling interval is better behaved with larger buffer sizes.

5 VISUALIZATION RESULTS

Finally, we show a few visualization examples with the large dataset. Color Plate 3 shows direct volume rendering of flow density surrounding the aircraft's wing. Color Plates 4 and 5 show visualizations of velocity magnitude. Direct volume rendering of this high resolution data elicits many fine features in the flow field which would be invisible with conventional two- or three-dimensional contour plots. In particular, in Plates 4 and 5 we can clearly verify the low pressure region (red spherical cloud) above the wing, and the high pressure region (yellow and orange blobs) below the wing. These two images also show the extreme low velocity values on the flaps (white stripes), and the complex flow patterns ahead of the leading edge and behind the trailing edge of the wing. None of these detailed phenomena could be seen with either low resolution data or low resolution rendering.

Some additional white patches appear as intermittent linear features near the upper and lower edges of the fuselage surface (which is also a grid boundary). We have yet to determine whether these artifacts are generated by the simulation, or are due to numerical problems in the renderer. Although the simulation produces double precision results, the dataset used for our tests

has been reduced to single precision in order to save disk space, reduce I/O time, and conserve memory in the renderer. It is conceivable that this loss of precision is causing erroneous values to appear during the scan conversion process.

6 CONCLUSIONS

We have conducted a series of performance tests with one of the largest unstructured-grid datasets used to date in parallel volume rendering research. Performance and scalability are good, and the T3E appears to be very well suited to the task. In particular, initial concerns about the volume of communication generated by such a large dataset appear to be unfounded, at least for this architecture. The primary impediment to interactive performance is ray merging time, suggesting that the additional communication and memory needed to support some form of early ray termination might be worth the cost. However, the visualizations shown here have very few truly opaque cells in them, so the utility of early ray termination is likely to be problem-dependent.

While the results presented here show good performance, they are most likely sub-optimal, given the complexities of tuning the algorithm to a particular dataset and a particular architecture. For maximum performance with minimum user intervention, a predictive, adaptive self-tuning strategy is needed so that the renderer can respond dynamically to changes in the input data, viewing parameters, or hardware configuration.

We also plan to study an even larger dataset which contains nearly 150 million tetrahedra. It is clear that direct, brute-force rendering will not provide interactive response for datasets of this size, even with massively parallel architectures. Consequently, we are investigating ways to integrate a multiresolution scheme into the rendering step.

In addition, a new generation of the high-lift analysis code is using mixed grids composed of prisms, pyramids, tetrahedra, and hexahedra in order to achieve higher efficiency. Our cell-projection rendering algorithm can be easily generalized to handle such mixed grids.

We have also conducted preliminary tests with the same algorithm on SGI's Origin2000 architecture. Our initial results were poor compared to the T3E, consistent with the findings in [3]. It is possible that our algorithm fares poorly on distributed shared memory architectures, or that deficiencies in memory management or message passing software are inhibiting scalability. We are designing new experiments for a more comprehensive study.

In the meantime, the renderer has also been ported to ICASE's 32-node Linux-based PC cluster, where it outperforms the T3E for comparable numbers of processors. Given that communication performance is much lower in the cluster, this difference is attributable primarily to faster processors (400 MHz Pentium II's), and we would expect performance scalability to be much more limited than on the T3E.

Acknowledgements

This work was supported by the National Aeronautics and Space Administration under Contract No. NAS1-97046 while the authors were in residence at ICASE. Access to the Goddard T3E was provided by NASA's HPCCP/CAS project, with assistance from Cathy Schulbach. We thank the system support personnel at NASA Goddard Space Flight Center for their help in using the system. The large dataset was provided by Dimitri Mavriplis and S. Pirzadeh. The small dataset was provided by Paresh Parikh.

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, vol. 18, no. 8, Sept. 1975, pp. 509–517.

[2] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 4, Oct.–Dec. 1997, pp. 352–369.

[3] T. W. Crockett. Portability and cross-platform performance of an MPI-based parallel polygon renderer. *Proceedings of HPCCP/CAS Workshop 98*, C. Schulbach and E. Mata, eds., NASA CP–1999-208757, NASA Ames Research Center, Jan. 1999, pp. 251–256.

[4] T. W. Crockett and T. Orloff. Parallel polygon rendering for message-passing architectures. *IEEE Parallel and Distributed Technology*, vol. 2, no. 2, Summer 1994, pp. 17–28.

[5] K.-L. Ma. Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures. *Proceedings of the 1995 Parallel Rendering Symposium*, ACM SIGGRAPH, Oct. 1995, pp. 23–30.

[6] K.-L. Ma and T. W. Crockett. A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. *Proceedings of the 1997 Symposium on Parallel Rendering*, ACM SIGGRAPH, Oct. 1997, pp. 95–104.

[7] X. Mao. Splatting of non-rectilinear volumes through stochastic resampling. *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 2, June 1996, pp. 156–170.

[8] D. J. Mavriplis and S. Pirzadeh. Large-scale parallel unstructured mesh computations for 3D high-lift analysis. *AIAA Journal of Aircraft*, 1999, to appear.

[9] C. Silva, J. Mitchell, and A. Kaufman. Fast rendering of irregular volume data. *Proceedings of the 1996 Symposium on Volume Visualization*, ACM SIGGRAPH, Oct. 1996, pp. 15–22.

[10] R. Westermann and T. Ertl. The VSBUFFER: visibility ordering of unstructured volume primitives by polygon drawing. *Proceedings Visualization '97*, IEEE CS Press, Oct. 1997, pp. 35–42.

[11] J. Wilhelms, A. Van Gelder, P. Tarantino, and J. Gibbs. Hierarchical and parallelizable direct volume rendering for irregular and multiple grids. *Proceedings Visualization '96*, IEEE CS Press, Oct. 1996, pp. 57–64.

[12] P. Williams, N. L. Max, and C. M. Stein. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, Jan.–Mar. 1998, pp. 37–54.

[13] R. Yagel, D. M. Reed, A. Law, P.-W. Shih, and N. Shareef. Hardware assisted volume rendering of unstructured grids by incremental slicing. *Proceedings of the 1996 Symposium on Volume Visualization*, ACM SIGGRAPH, Oct. 1996, pp. 55–62.

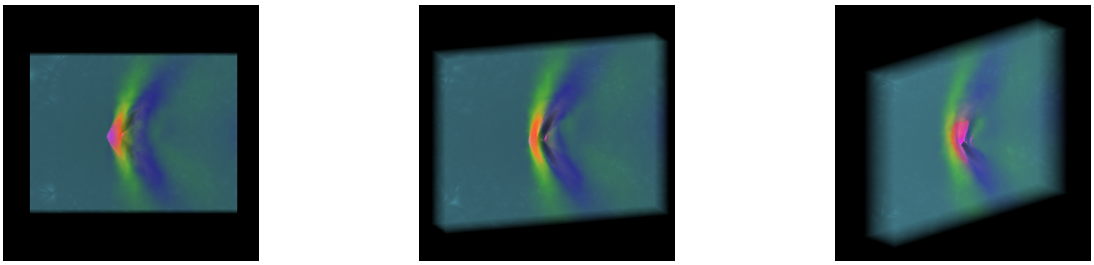

Plate 1. Test sequence for the small dataset (567,862 cells). Flowfield over an aircraft wing with a missile attached.

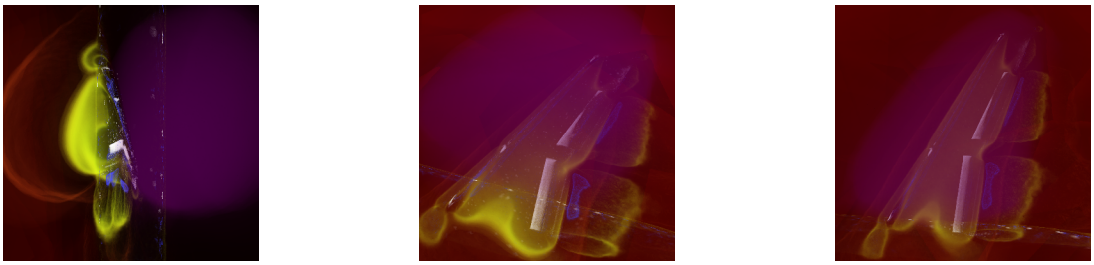

Plate 2. Test sequence for the large dataset (18,216,138 cells). Velocity field around a high-lift wing configuration.

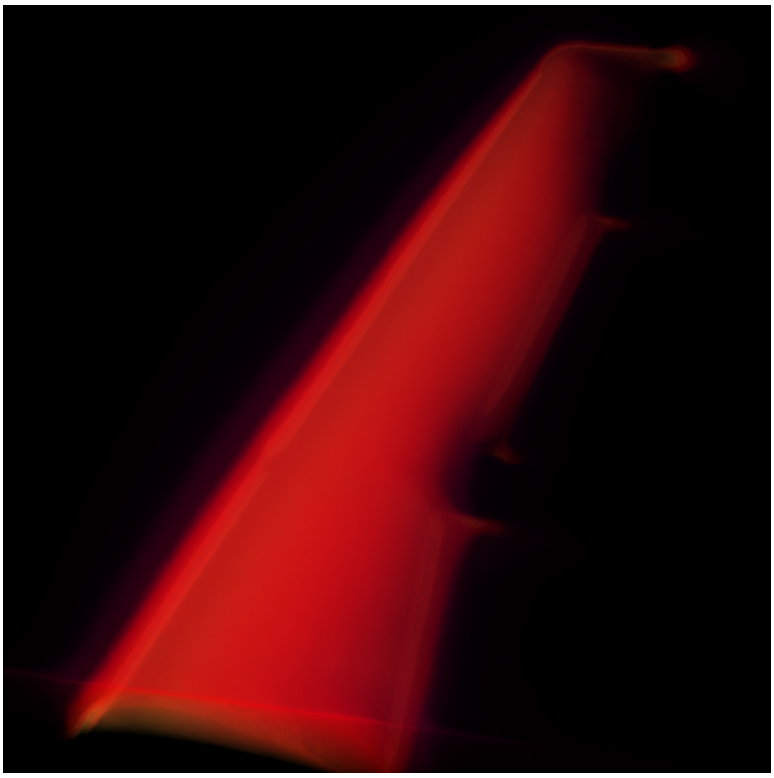

Plate 3. Density; a view from above the wing.

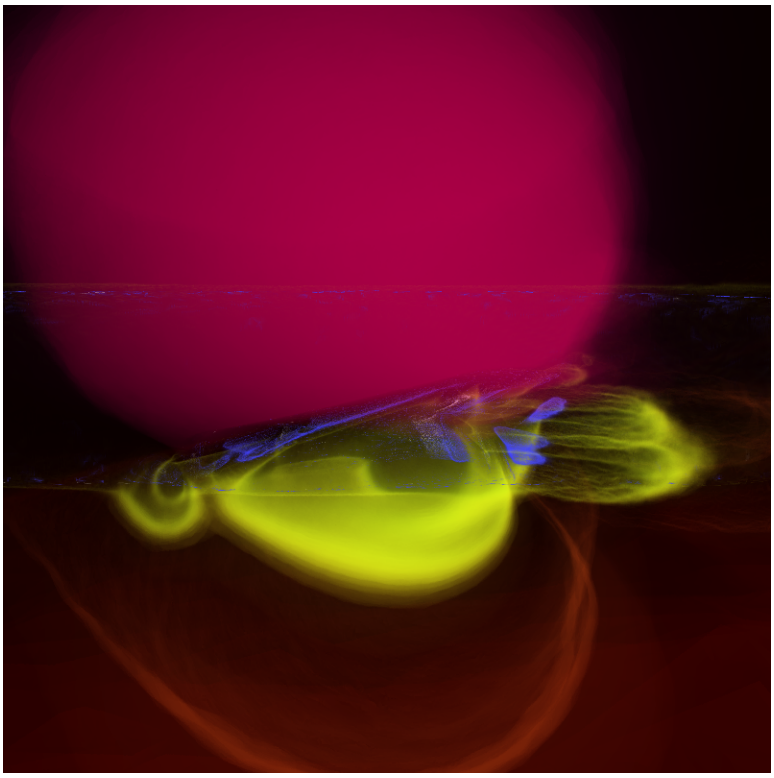

Plate 4. Velocity magnitude; a view from the fuselage toward the tip of the wing.

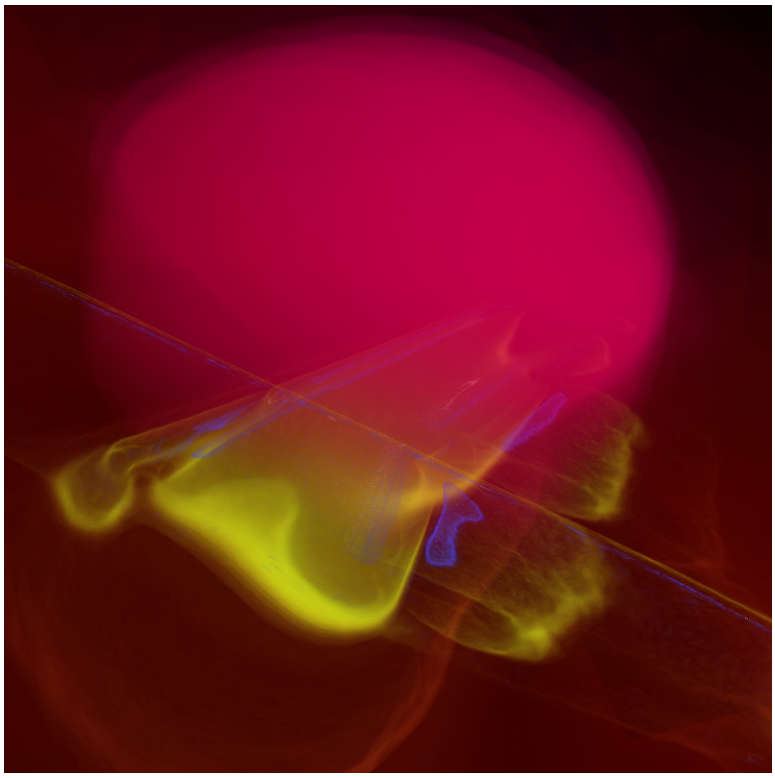

Plate 5. Velocity magnitude; a view from above and behind the wing.