A Scalable Parallel Software Volume Rendering Algorithm for Large-Scale Unstructured Data

Kangjian Wang^c and Yao Zheng^{*}

College of Computer Science, and Center for Engineering and Scientific Computation,
Zhejiang University, Hangzhou, 310027, P.R. China
kangjian.wang@hotmail.com, yao.zheng@zju.edu.cn

Abstract. In this paper, we develop a highly accurate parallel software scanned cell projection algorithm (PSSCPA) which is applicable of any classification system. This algorithm could handle both convex and non-convex meshes, and provide maximum flexibilities in applicable types of cells. Compared with previous algorithms using 3D commodity graphics hardware, it introduces no the volume decomposition and rendering artifacts in the resulting images. Finally, high resolution images generated by the algorithm are provided, and the scalability of the algorithm is demonstrated on a PC Cluster with modest parallel resources.

Keywords: Parallel volume rendering, cell projection, software volume rendering.

1 Introduction

Traditionally, parallel volume rendering algorithms were designed to run on expensive parallel machines like SGI Power Challenge, IBM SP2, or SGI Origin 2000 [1, 2, 3, 4]. Recently, however, the decreasing cost and high availability of commodity PCs and network technologies have enabled researchers to build powerful PC clusters for large-scale computations. Scientists can now afford to use clusters for visualization calculations either for runtime visual monitoring of simulations or post-processing visualization. Therefore, parallel software volume rendering on clusters is becoming a viable solution for visualizing large-scale data sets.

We develop a highly accurate Parallel Software Scanned Cell Projection Algorithm (PSSCPA) in this paper. The algorithm employs a standard scan-line algorithm and a partial pre-integration method proposed by Moreland and Angel [5], and thus supports the rendering of data with any classification system. It could handle meshes composed of tetrahedra, bricks, prisms, wedges, and pyramids, or the complex of these types of cells. The PSSCPA runs on the distributed-memory parallel architectures, and uses asynchronous *send/receive* operations and a multi-buffer method to reduce communication overheads and overlap rendering computations. Moreover, we use a hierarchical spatial data structure and an A-Buffer [6] technique to allow early ray-merging to take place within a local neighborhood.

^{*} Corresponding author.

The remainder of this paper is organized as follows. In the next section we relate our work to previous research. Section 3 describes the PSSCPA. In section 4 we present the experiment results of the PSSCPA. The paper is concluded in section 5, where some proposals for future work are also presented.

2 Related Work

Cell projection is a well-known volume rendering technique for unstructured meshes. A scalar field is formed by specifying scalar values at all vertices of a mesh, and then visualized by mapping it to colors and opacities with feasible transfer functions.

To efficiently visualize large-scale unstructured grid volume data, parallel processing is one of the best options. Ma [3] presented a parallel ray-casting volume rendering algorithm on distributed memory architectures. This algorithm needs explicit connectivity information for each ray to march from one element to the next, which incurs considerable memory usage and computational overheads. Nieh and Levoy [7] developed a parallel volume rendering system. Their algorithm, however, was tested on a distributed-shared memory architecture, so did the PZSweep algorithm proposed by Farias et al. [8]. Farias et al. [9] soon enhanced the PZSweep routine [8] to configure it on a PC cluster, and strived to find a feasible load balancing strategy applicable for the new architecture. Chen et al. [10] presented a hybrid parallel rendering algorithm on SMP clusters, which make it easier to achieve efficient load balancing. Ma et al. [4] presented a variant algorithm without requirement for connectivity information. Since each tetrahedral cell is rendered independently of other cells, data can be distributed in a more flexible manner. The PSSCPA investigated in this paper is just originated from this algorithm. However, several improvements have been made, as introduced in Section 3.

3 The Parallel Software Scanned Cell Projection Algorithm

Fig. 1 shows the parallel volume rendering pipeline performed by the PSSCPA. The exterior faces of the volume and the volume data are distributed in a round robin fashion among processors. The image screen is divided using a simple scan-line interleaving scheme. Then a parallel k-d tree and an A-Buffer are constructed. They are used in the rendering step to optimize the compositing process and to reduce runtime memory consumption. Each processor scan converts its local cells to produce many ray segments, and send them to their final destinations in image space for merging. A multi-buffer scheme is used in conjunction with asynchronous communication operations to reduce overheads and overlap communications of ray segments with rendering computations. When scan conversion and ray-segment mergence are finished, the master node receives competed sub-images from all slavers and then assembles them for display. Our algorithm can handle convex meshes, non-convex meshes, and meshes with disconnected components.

Fig. 1. The parallel volume rendering pipeline

3.1 Data Distribution

The ideal goal of a data distribution scheme is that each processor incurs the same computational load and the same amount of memory usage. However, it is prevented being obtained by several factors. First, there are some computational costs to scan convert a cell. Variations in the number of cells assigned to each processor will produce variations in workloads. Second, cells come in different sizes and shapes. The difference in size can be as large as several orders of magnitude due to the adaptive nature of the mesh. As a result, the projected image area of a cell can vary dramatically, which produces similar variations in scan conversion costs. Finally, the projected area of a cell also depends on the viewing direction.

Generally, nearby cells in object space are often similar in size, so that grouping them together exacerbates load imbalances, making it very difficult to obtain satisfactory result. We have therefore chosen to take the round-robin scheme, dispersing connected cells as widely as possible among the processors. Thus with sufficient enough cells, the computational requirements for each processor tend to average out, producing an approximate load balance. The approach also satisfies our requirement for flexibility, since the data distribution can be computed trivially for any number of processors, without the need for an expensive preprocessing time.

We also need to evenly distribute the pixel-oriented ray-merging operations. Local variations in cell sizes within the mesh lead directly to variations in depth complexity in image space. Therefore we need an image partitioning strategy to disperse the ray-merging operations as well. Scan-line interleaving, which assigns successive image scan-lines to processors in the round-robin fashion, generally works well as long as the image's vertical resolution is several times larger than the number of processors. In our current implementation, we use this strategy.

3.2 Parallel k-d Tree

Our round-robin data distribution scheme completely destroys the spatial coherence among neighboring mesh cells, making an unstructured dataset even more irregular. We would like to restore some ordering so that the rendering step may be performed more efficiently. We are to have all processors render the cells in the same neighborhood at about the same time. Ray segments generated for a particular region will consequently arrive at their image-space destinations within a relatively short window of time, allowing them to be merged early. This early merging reduces the length of the ray-segment list maintained by each processor, which benefits the rendering process in two ways: first, a shorter list reduces the cost of inserting a ray segment in its proper position within the list; and second, the memory needed to store unmerged ray segments is reduced.

To provide the desired ordering, a parallel k-d tree should be constructed cooperatively so that the resulting spatial partitioning is exactly the same on each processor. After the data cells are initially distributed, all processors participate in a synchronized parallel partitioning process. A detailed description of the algorithm has been given by Ma et al. [4].

3.3 Creating an A-Buffer

For a convex mesh, a parallel k-d tree should be constructed for ray-segments to be merged early. However, for a non-convex mesh or a mesh with disconnected components, only the k-d tree is not sufficient. The ray-gaps, i.e. a segment of a ray between a point on an exterior face of a mesh where the ray leaves the mesh, and another such point where the ray reenters a mesh, will clag ray-segments to be merged early. Our approach to this problem is to add an assistant A-Buffer.

We can identify exterior faces of a mesh and evenly distribute them to each processor. Then the exterior faces will scan-converted, and ray-face intersections are sent to their image-space destinations. Each processor saves the ray-face intersections received from other processor in an A-Buffer type of data structure along each ray.

An A-Buffer is created, implemented as an array of pixel (PIX) lists, one per pixel in image space. Fig. 2 shows an A-Buffer on a slice across a scan line through an unstructured mesh with disconnected components.

A PIX list consists of a series of PIX list entry records (PIX entries), as described below. As each pixel p of an exterior face f is enumerated by the scan conversion, a new PIX entry is created containing the distance z from the screen to p , a pointer to f , and a next pointer for the PIX list. The PIX entry is then inserted into the appropriate PIX list in the A-Buffer, in the order of increasing z . At each pixel location, we maintain a linked list of ray segments, which are merged to form the final pixel value. The ray segments in each linked list assisted by a PIX list are allowed to be merged early in a non-convex mesh or a mesh with disconnected components.

Fig. 2. Diagram of an A-Buffer on a 2D slice

3.4 Task Management with Asynchronous Communication

To reduce computational overheads and improve parallel efficiencies, we adopt an asynchronous communication strategy first suggested by Ma et al. [4].

Fig. 3. Task management with asynchronous communications

This scheme allows us to overlap computation and communication, which hides data transfer overheads and spreads the communication load over time. During the course of rendering, there are two main tasks to be performed: scan conversion and image composition. High efficiency is obtained if we can keep all processors busy within either of these two tasks. We employ a polling strategy to interleave the two tasks, and thus achieve a good performance. Fig. 3 illustrates at a high level the management of the two tasks and the accompanying communications. Each processor starts by scan converting one 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.

Because of the large number of ray segments generated, the overheads for communicating each of them individually would be prohibitive in most parallel machines. Instead, it is better to buffer them locally and send many ray segments together in one operation. This strategy is even more effective when multi-buffers are provided for each destination. When a send operation is pending for a full buffer, the scan conversion process can be placing ray segments in other buffers.

3.5 Image Composition

Since we divide the image space using a simple scan-line interleaving scheme, the image composition step is very simple. Tiles do not overlap, and pixels are generated independently. So the tiles rendered by each processor correspond directly to sub-images, and there is no need for depth composition. When the master processor receives from other processors the sub-images rendered, it simply pastes it on the final image.

4 Experimental Results

We implemented our PSSCPA in the C++ language using MPI message passing for inter-processor communication. All the experiments presented subsequently are conducted on a Dawning PC Cluster (Intel CPUs: 48*2.4GHz; Memory: 48GB; Network: 1-Gbit Fast Ethernet) at the Center for Engineering and Scientific Computation (CESC), Zhejiang University (ZJU).

We have used two three different datasets in our experiments: G1 and FD. G1 represents the stress distribution on a three-dimensional gear. FD represents the evolution of the structure of interface in three-dimensional Rayleigh-Taylor instability. Both G1 and FD are unstructured grids composed of tetrahedral cells. G1 is a

Fig. 4. Volume rendering of G1

Fig. 5. Volume rendering of FD

Fig. 6. The rendering time of G1

Fig. 7. The rendering time of FD

Fig. 8. Speedup

Fig. 9. Parallel efficiency

non-convex mesh with 0.5M cells; FD is a convex mesh with 3M cells. The image size in our experiments is 512*512. Figs. 4 and 5 show two volume-rendered views.

Figs.6 and 7 plot the rendering time versus the number of processors. With 32 processors involved we can render 0.5M tetrahedral cells in 3.1 seconds per frame.

Figs.8 and 9 show the speedups and parallel efficiencies obtained as the number of processors varies from 1 to 32, respectively While 32 processors are involved, we achieve a speedup of 20, and a parallel efficiency of 63%, for FD.

5 Conclusions and Future work

In this paper, by combining a k-d tree and an A-Buffer with an asynchronous communication strategy, we have developed a volume renderer for unstructured meshes which employs inexpensive static load balancing to achieve good performance. Because the partial pre-integration method is adopted in the volume renderer, our system

supports the rendering of data with any classification system. By employing the A-Buffer technique, our system also can handle convex meshes, non-convex meshes, or meshes with disconnected components.

In the future, we will extend the PSSCPA to the problem of rendering time-varying data.

Acknowledgements. The authors would like to thank the National Natural Science Foundation of China, for the National Science Fund for Distinguished Young Scholars under grant No.60225009 and the Major Program of the National Natural Science Foundation of China under Grant No.90405003. The first author is grateful to the simulation data provided by Jianfeng Zou, the constructive discussions with Dabin Zhou, and the valuable suggestions from Jianjun Chen, Lijun Xie, Jian Deng.

References

1. C. Hofsetz and K.-L. Ma.: Multi-threaded rendering unstructured-grid volume data on the SGI origin 2000. In Third Eurographics Workshop on Parallel Graphics and Visualization, (2000)
2. L. Hong and A. Kaufman.: Accelerated ray-casting for curvilinear volumes IEEE Visualization'98, October (1998) 247–254.
3. K.-L. Ma.: Parallel volume ray-casting for unstructured-grid data on distributed-memory architectures. IEEE Parallel Rendering Symposium, October (1995) 23–30
4. K.-L. Ma and T. Crockett.: A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. IEEE Parallel Rendering Symposium, November (1997) 95–104
5. Moreland, K. Angel, E.: A fast high accuracy volume renderer for unstructured data. In Proceedings of IEEE Symposium on Volume Visualization and Graphics 2004, October (2004)9-16
6. L. Carpenter.: The A-buffer, an antialiased hidden surface method. In Computer Graphics Proc., SIGGRAPH'84, July (1984) 103-108
7. J. Nieh and M. Levoy.: Volume rendering on scalable shared-memory mimd architectures. In 1992 Workshop on Volume Visualization Proceedings, October (1992) 17–24
8. R. Farias, and C. Silva.: Parallelizing the ZSWEEP Algorithm for Distributed-Shared Memory Architectures. In International Workshop on Volume Graphics, October (2001) 91–99
9. R. Farias, C. Bentes, A. Coelho, S. Guedes, L. Goncalves.: Work distribution for parallel ZSweep algorithm. In XVI Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAP'03), (2003) 107- 114
10. L. Chen, I. Fujishiro, and K. Nakajima.: Parallel performance optimization of large-scale unstructured data visualization for the earth simulator. In Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization, (2002) 133-140