PARALLEL VOLUME RENDERING METHOD FOR OUT-OF-CORE NON-UNIFORMLY PARTITIONED DATASETS

Jian Xue, Xiaoye Zhu, Ke Lu *

University of Chinese Academy of Sciences
Beijing 100049, China

Yutong Kou

Huazhong University of Science & Technology
Wuhan 430074, China

ABSTRACT

The rapid development of graphics hardware has spawned a large number of GPU-based direct volume rendering methods. However, fast volume rendering for out-of-core datasets faces specific challenges, and many in-core-based methods cannot be applied to the volume rendering of out-of-core datasets. Although volume rendering algorithms based on partitioning strategies can effectively overcome these problems, the current partition-based techniques do not achieve a good balance between the partitioning method and the sub-block drawing method. In this paper, we propose an efficient out-of-core volume rendering method for datasets consisting of non-uniform size sub-blocks. The proposed method examines the parallel rendering possibilities between sub-blocks to accelerate the rendering process. Experimental comparisons with several volume rendering algorithms show that our method is faster than the current state-of-the-art approaches.

Index Terms— out-of-core datasets, parallel volume rendering, partitioning

1. INTRODUCTION

In scientific computing, visualization techniques use computer graphics and image processing strategies to convert computational data into graphics and images that are displayed on the screen and processed interactively. Volume rendering is widely used in the field of medical and industrial imaging because it offers the efficient rendering of high-quality images and convenient parallel processing. The rapid development and wide application of graphics hardware has brought about explosive growth in the amount of image data, leading to new developments in visualization technologies for 3D data. However, the traditional in-core methods are limited by their internal and external memory access speed, resulting in fairly slow rendering speeds. Over the past two decades, many algorithms have been proposed to overcome this problem. One popular trend in volume rendering is based on a partition strategy, which divides the original data into sub-blocks before synthesizing the rendering result. This method can be used alongside hardware acceleration technology to achieve a better rendering speed. Inspired by this, we propose a volume rendering method based on partitioning by non-uniform size sub-blocks. When applied to ray-casting-based hardware-accelerated volume rendering and compared with state-of-the-art methods on various datasets, our method exhibits good performance.

2. RELATED WORK

Research on 3D visualization began in the mid-1970s, and Levoy et al. [1] proposed the first volume rendering method in 1988, known as the ray casting algorithm. Many volume rendering algorithms that directly process the spatial domain appeared shortly after. In recent years, research on volume rendering methods for large-scale data has continued. Okuyan [2] proposed a GPU-based volume rendering algorithm that uses a cell projection-based ray-casting algorithm designed for CPU implementations. This enables the direct volume rendering of unstructured out-of-core volume data. The novel SparseLeap method [3] moves the major cost of empty space skipping out of the ray-casting stage by passing over such regions in very large volumes efficiently. Maloca et al. [4] developed an advanced high-end VR image display method to provide new views and interactions in ultra-high-speed projected digital scenery.

In terms of 3D scalar data fields, the development of volume rendering methods faces two main challenges. First, many volume rendering applications for massive data are still severely constrained by the contradiction between huge data volumes and limited computing resources. With the concept of general-purpose GPUs (GPGPUs), heterogeneous parallel computing platforms based on CPU–GPU combinations have become feasible in PC systems, with Gelder [5] and Engel [6] describing classic methods of using GPUs to accelerate rendering. Second, traditional methods do not perform well

*This work was supported in part by the National Key R&D Program of China (2017YFB1002203), the National Natural Science Foundation of China (61671426, 61731022, 61871258, 61471150, 61572077), the Beijing Natural Science Foundation (4182071), the Instrument Developing Project of the Chinese Academy of Sciences (YZ201670), and the Scientific Research Program of Beijing Municipal Education Commission (KZ201911417048).
(*Corresponding auther: Ke Lu; email: luk@ucas.ac.cn*)

978-1-5386-9214-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICMEW.2019.00109

with large datasets, because they were not designed for scenarios in which the amount of data exceeds the size of the internal memory. To improve the efficiency, many methods have been applied at various stages of the volume rendering process. For example, the method of Novins [7] calculates the rendering result for each slice, and can draw the ray projection without importing all the slices into the internal memory. Lim [8] applied K-means clustering to transfer function design, resulting in the volume data boundaries being extracted according to the cluster centroids. Volume rendering for out-of-core datasets based on a partitioning strategy has also been developed. Xue [9] designed an out-of-core visualization framework to process out-of-core datasets, and proposed a fast volume rendering method based on a semi-adaptive partitioning strategy.

Based on the above studies, and assuming a uniform-size partitioning strategy, Yao [10] grouped sub-blocks according to the parallel relationship between them, and achieved a good acceleration effect by drawing groups in parallel. Comparing the algorithms of Yao [10] with those described above, we can draw the conclusion that it is feasible to deal with massive data in a block manner, so that data access is indexed by the block number. However, the equal-sized partitioning strategy used by Yao cannot effectively distinguish the background area. Naturally, we consider the use of a fast volume rendering method for datasets of non-uniform size sub-blocks. This method could be used to process non-uniform size sub-block datasets, such as in the semi-adaptive partitioning of volume data processed by Xue [9]'s method.

3. PARALLEL RENDERING METHOD FOR PARTITIONED DATASETS

Based on the above work, we propose a new fast volume rendering method for datasets consisting of non-uniform size sub-blocks. This method obtains suitable non-uniform size sub-blocks so as to control the number of partitions while maximizing the background area and reducing the internal and external data exchange overhead. In the proposed method, the input volume data has been divided into non-uniform size sub-blocks and stored in an array (i.e. the input data of the proposed method is a partitioned dataset). The array elements contain size and spatial coordinates of each sub-block.

Firstly, we design the sub-block data structure to store the original information and additional information such as adjacent sub-block's index obtained through calculations. Secondly, we traverse the sub-blocks and divide them into empty blocks and non-empty blocks according to the transfer function so that only the non-empty sub-blocks are drawn. We then start from an unvisited sub-block at random, and conduct a breadth-first search (BFS) to generate a global occlusion relationship graph according to the occlusion relationship between the sub-blocks. Topological sorting is then applied to identify sub-blocks with parallel relationships and place them into one group. Finally, all the groups of sub-blocks are rendered from front to back using a GPU-implemented volume ray caster. The details of each step are described in the following sub-sections. The overall process of our method is illustrated in Fig.1.

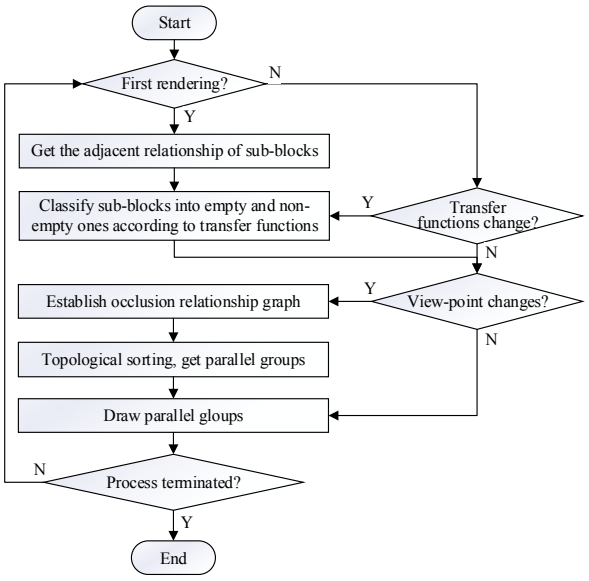


Fig. 1: Flowchart for the new volume rendering algorithm.

3.1. Adjacent relationship of sub-blocks

Initially, the volume data are divided into non-uniform size sub-blocks, and we store the size and spatial coordinates of the sub-blocks. However, to obtain the occlusion relationship between sub-blocks, we need to collect some additional information for each sub-block first. This includes: (1) $fidx$, which is used to locate the position of data in the external memory; (2) $bbox[6]$, which stores the spatial coordinates of the bounding box; (3) $adj[6]$, which stores the "$fidx$" of the adjacent sub-blocks in contact with each of the current block's six boundary faces.

Because of non-uniform partitioning, there may be an uncertain number of adjacent sub-blocks in contact with each of the six contacted surfaces. To obtain the necessary information, we traverse all of the sub-blocks, and for each sub-block, search for the remaining sub-blocks that satisfy certain conditions and add their information into $adj[i]$ for its ith boundary face. For example, for sub-block $B1$, if sub-block $B2$ satisfies the following three conditions: (1) $|B1.bbox[3] - B2.bbox[2]| \leq 1$; (2) $B1.bbox[0] < B2.bbox[1]$ or $B1.bbox[1] > B2.bbox[0]$; and (3) $B1.bbox[4] < B2.bbox[5]$ or $B1.bbox[5] > B2.bbox[4]$. Then $B2$ is a sub-block that is in contact with the face (upper face) of $B1$. We add the "$fidx$" of $B2$ into $B1.adj[3]$.

3.2. Occlusion relationship graph

After classify the sub-blocks into empty and non-empty ones according to the transfer function, we start from a random non-empty sub-block B_0 to establish the occlusion relationship graph for all non-empty sub-blocks. The occlusion relationship of the current sub-block and its adjacent sub-blocks on each contact face are determined in turn according to the judgment criterion represented by Fig.2. The occlusion relationship between two adjacent sub-blocks can be determined by the angle between view direction and the normal of the contact face.

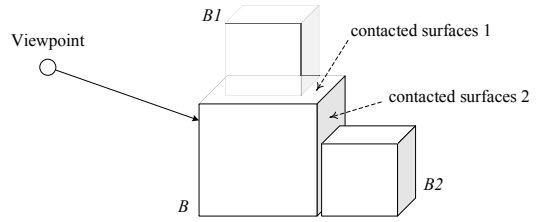


Fig. 2: Judgment of occlusion relationship between sub-blocks, where B does not occludes $B1$, B occludes $B2$.

The process of establishing a occlusion relationship graph is based on the BFS (Breadth First Search) algorithm. Initially, there are no edge connections between any sub-blocks, and the in-degree of all sub-blocks is 0. We then search from a random sub-block that is currently unvisited and set up directed edges between sub-blocks which have occlusion relationship according to the adjacent relationship and the occlusion judgment criterion during the search procedure. After BFS process finished, the occlusion relationship graph is established. Obviously, this graph should be a DAG (Directed Acyclic Graph). Otherwise, the sub-blocks cannot be rendered correctly because of the existence of cyclic occlusion relationship among sub-blocks.

3.3. Parallel rendering for groups of sub-blocks

We use the method from Yao [10] to perform a topological sort on the occlusion relationship graph and obtain multiple parallel groups. Then the sub-blocks in each group are drawn in parallel and the drawing between groups is conducted in series. For a set of grouped sub-blocks, using GPUs to draw them in parallel would result in faster rendering speed. We improved the original GPU ray caster using Krüger's [11] method, and implemented the ray casting method on the GPU to render a group of sub-blocks in parallel.

Firstly, we construct a large 3D texture containing sub-textures, where the large texture is the delivery medium for passing sub-block data into the fragment shader. Secondly, depth maps of the front and back faces of all sub-blocks in a parallel group are passed into the fragment shader, along with the index of the sub-blocks within the parallel group. This information is used to calculate the direction and starting position of the ray, respectively. We then add the sub-block data to the large texture, and pass the three-dimensional texture into the fragment shader. Finally, we push the ray in the fragment shader and accumulate color and opacity.

There are three points to note in the above process: First, the large 3D texture has a fixed size and is only created once, because this process is very time consuming. Subsequently, this structure is overwritten with new sub-block data. Second, the large 3D texture is divided into uniform-sized sub-textures, with the maximum width, height, and depth of the sub-blocks in the current group used to define the size of the sub-textures. Thus, when we draw each group of sub-blocks, we pass the size of each sub-texture into the fragment shader and recalculate the coordinates of the sub-textures. Third, when drawing the large texture, the position of the ray needs to be offset, and so we need to shift the ray rendering the different sub-blocks to the corresponding position. Thus, an additional 2D texture with the same size as the frame buffer is used to store the index of sub-textures.

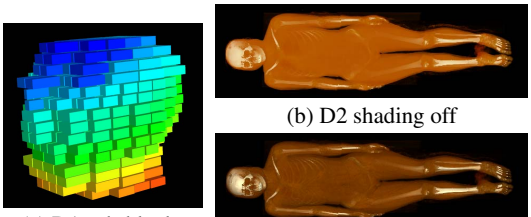
4. EXPERIMENTAL RESULTS AND ANALYSIS

We implemented the method proposed in this paper and compared its performance with that of other algorithms. The algorithms used in the experiments are named A1–A5. A1 is an algorithm similar to that in [7], but with parallel acceleration via OpenMP. A2 is an algorithm based on semi-adaptive partitioning and serial rendering, i.e. the sub-blocks are rendered one by one. A3 is based on uniform-sized partitioning and serial rendering. A4 is Yao [10]'s algorithm based on uniform-sized partitioning and parallel drawing of sub-blocks. A5 is the proposed method of this paper.

Three real datasets of different sizes are used to test the algorithms. The first dataset (denoted as D1) is taken from www.psychology.nottingham.ac.uk, and has a size of $208 \times 256 \times 225 \times 8$bit (11.43MB). The second dataset (denoted as D2) is the CT dataset of the Chinese Visible Human from Southern Medical University $512 \times 512 \times 1714 \times 16$bit (857MB). The third dataset(denoted as D3) is the result of scanning chips for flaw detection, which is collected from an ultrasonic non-destructive testing (NDT) system. D3 has a size of $3000 \times 2400 \times 1900 \times 16$bit (25.48GB).

All the tests are run on a PC equipped with Intel Core i5-6500 CPU, 8GB physical memory and a GeForce GTX 1060 graphics card. Some of the rendering results are presented in Fig.3, where Fig.3a illustrates the sorting result of the sub-blocks of D1, where the colors (blue to red) indicate the distances (near to far) from the viewpoint to the the sub-blocks, and the same color means the same parallel group.

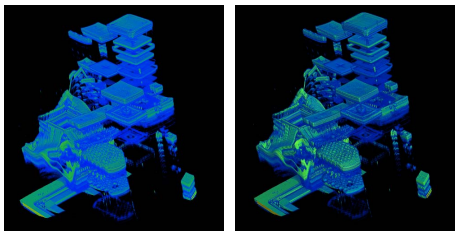
Table 1 lists the rendering speed of the five algorithms, which indicate that the proposed algorithm achieves good performance in terms of rendering speed. Compared to serial



(a) D1 sub-blocks

(b) D2 shading off

(c) D2 shading on



(d) D3 shading off (e) D3 shading on

Fig. 3: Some rendering results of the new algorithm.

Table 1: Comparison of the rendering speed

	Shading off			Shading on		
	D1	D2	D3	D1	D2	D3
A1	0.25s	1.81s	364s	0.26s	2.23s	825s
A2	0.08s	1.78s	21.22s	0.12s	1.85s	22.89s
A3	0.04s	1.82s	45.66s	0.10s	2.55s	71.71s
A4	0.02s	1.37s	17.65s	0.04s	1.41s	17.78s
A5	**0.01s**	**1.08s**	**15.36s**	**0.02s**	**1.10s**	**15.60s**

rendering (A2 and A3), our algorithm is much faster regardless of the partitioning strategy. The proposed algorithm is always faster than A4, with the speed advantage of the new algorithm becoming more obvious as the amount of data increases.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a parallel volume rendering algorithm for out-of-core data that can obtain high-resolution rendering results. This algorithm has been designed to handle out-of-core datasets partitioned by non-uniform size sub-blocks, and can separate the empty parts of the original dataset as far as possible and reduce the data transferred from the external memory to the internal memory. After discarding unnecessary sub-blocks, the remaining sub-blocks are processed in parallel on the GPU to improve the rendering speed. Experimental results show that the new method is effective and efficient for visualizing out-of-core datasets.

Our algorithm works well on a PC, and it is possible to draw separate sub-blocks on distributed computing nodes, which will further improve the rendering efficiency. Future work will focus on this direction of research.

6. REFERENCES

[1] Marc Levoy, *Display of Surfaces from Volume Data*, IEEE Computer Society Press, 1988.

[2] Erhan Okuyan and Ugur Gudukbay, "Direct volume rendering of unstructured tetrahedral meshes using cuda and openmp," *The Journal of Supercomputing*, vol. 67, no. 2, pp. 324–344, 2014.

[3] Markus Hadwiger, Ali K Alawami, Johanna Beyer, Marco Agus, and Hanspeter Pfister, "Sparseleap : Efficient empty space skipping for large-scale volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 974–983, 2018.

[4] Peter Maloca, J Ramos de Carvalho, Tjebo Heeren, Pascal Hasler, Faisal Mushtaq, Mark Mon-Williams, Hendrik P N Scholl, Konstantinos Balaskas, Catherine Egan, Adnan Tufail, Lilian Witthauer, and Philippe C Cattin, "High-performance virtual reality volume rendering of original optical coherence tomography point-cloud data enhanced with real-time ray casting," *Translational Vision Science & Technology*, vol. 7, no. 4, 2018.

[5] Allen Van Gelder and Kwansik Kim, "Direct volume rendering with shading via three-dimensional textures," in *Symposium on Volume Visualization*, 1996.

[6] Klaus Engel, Martin Kraus, and Thomas Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," in *Proc. Eurographics/siggraph Workshop on Graphics Hardware*, 2001, pp. 9–16.

[7] Kevin L Novins, Francois X Sillion, and Donald P Greenberg, "An efficient method for volume rendering using perspective projection," *Acm Siggraph Computer Graphics*, vol. 24, no. 5, pp. 95–102, 1990.

[8] Jae Hwan Lim and Young Hwan Kim, "Transfer function design for volume rendering using k-means clustering," pp. 48–49, 2014.

[9] Jian Xue, Jun Yao, Ke Lu, Ling Shao, and Mohammad Muntasir Rahman, "Efficient volume rendering methods for out-of-core datasets by semi-adaptive partitioning," *Information Sciences*, vol. 370-371, pp. 463–475, 2016.

[10] Jun Yao, Jian Xue, Ke Lv, and Qinghai Miao, "A parallel volume rendering method for massive data," in *IEEE International Conference on Multimedia & Expo Workshops*, 2016, pp. 1–6.

[11] J. Krüger and R. Westermann, "Acceleration techniques for gpu-based volume rendering," *Proc.ieee Visualization Oct*, pp. 287–292, 2003.