Eurographics Conference on Visualization (EuroVis), Posters Track (2017)
A. Puig Puig and T. Isenberg (Editors)

Memory Efficient Parallel Ray-casting Algorithm for Unstructured Grid Volume Rendering

Duksu Kim †

KISTI (Korea Institute of Science and Technology Information)

Abstract

We propose a novel memory efficient parallel ray casting algorithm for unstructured grid. To reduce the high memory consumption of a previous work (i.e. Bunyk), we use a small size of local buffer for each thread to keep view dependent information for most recently visited faces. To improve the utilization efficiency of a local buffer, we propose an index-based hash function and a novel group traversal scheme. With our method, a small size buffer achieves a high hit ratio (i.e. utilization efficiency). As a result, we achieved a compatible performance with Bunyk while using less than 1% of memory space for view dependent face information. Also, our method shows even better performance than Bunyk for a large dataset since the buffer size is small enough to utilize CPU caches and our traversal scheme maximally takes advantage of the early ray termination optimization.

1. Introduction

Direct volume rendering (DVR) is one of the most fundamental visualization methods and widely used in various fields including medical image, scientific simulations, and so on. Ray casting is the most popular algorithm for DVR due to its generality and accuracy. Ray casting for uniform grid having regular structure is relatively easy to implement and various acceleration techniques are well studied including parallel algorithms using multi-core CPUs and GPUs [BW01].

For unstructured grid, it is rather complex to perform ray casing and requires much computation. Bunyk et al. [BKS97] proposed a fast ray casting algorithm for unstructured grid. This *Bunyk* algorithm computes view dependent face information (VDFI) for all faces as a pre-processing. With the full VDFI list, they could perform ray traversal without duplicated computations and achieved high performance improvement. However, it requires much memory space and hard to apply to a large dataset. Riberio et al. [RMB*07] solved the memory overhead by using a small size buffer instead of a full VDFI list. Their Face Driven Ray-Casting (VF-Ray) method achieved a close performance with Bunyk by exploiting ray coherency that means nearby rays tend to visit similar faces. Maximo et al. [MRB*08] extended VF-Ray to a GPU parallel algorithm (GPU VF-Ray) by allocating a buffer for each thread, and it achieved up to five times faster performance than a serial algorithm. However, it is still hard to apply to a large dataset with a limited GPU memory since a massively parallel algorithm needs to launch tens of thousands of threads and it consumes much memory (e.g., a few gigabytes) even though each local buffer is small. Also, it does not guarantee accurate results for non-convex meshes.

Advantage of Our Approach: We propose a novel memory efficient parallel ray casting algorithm for unstructured grid. Our method has following benefits.

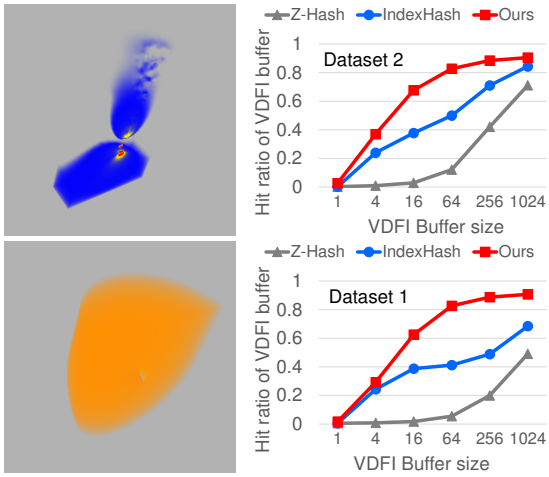

Figure 1: *The left images show rendering results of two benchmark datasets. The right graphs show the VDFI buffer hit ratio of three different algorithms. Z-Hash is results of using the hash function proposed by Maximo et al. [MRB*08]. Index Hash is results with our index-based hash function. Ours is results when we use Index Hash with our group traversal algorithm.*

1. **Memory efficient:** With our index-based hash function and a novel group traversal algorithm, a small size buffer shows a high utilization efficiency (Fig. 1). As a result, we can achieve a compatible performance with the original Bunyk algorithm while using less than 1% memory for VDFI and about half memory for whole process.
2. **High performance for a large dataset:** In some cases, our method achieved even higher performance than Bunyk for a large dataset (Fig. 3). It is because the size of buffer is small enough to fit to CPU caches while showing a high hit ratio. Also, our group traversal scheme is good for exploiting the early ray

† bluekdct@gmail.com

DOI: 10.2312/eurp.20171157

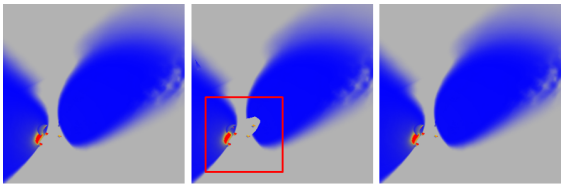

Figure 2: *These figures show rendering results of (a) the bunyk algorithm [BKS97], (b) GPU VF-Ray [MRB*08], and (c) our method. We found that the result of GPU VF-Ray has artifacts for the non-convex mesh.*

termination optimization and it leads further performance improvement.

3. **Accurate for non-convex mesh:** With our memory efficient algorithm, we can employ a ray-face intersection list like Bunyk. This leads accurate results for non-convex meshes different with GPU VF-Ray that does not maintain the intersection information to save the limited memory of GPUs (Fig 2).

2. Our Approach

Our method is based on Bunyk algorithm and we allocate a fixed size of local buffer for each thread similar with GPU VF-Ray [MRB*08]. Local buffers maintain VDFIs for most recently visited faces and reuse them when the next ray meets the faces (buffer hit). If the buffer does not have the VDFI for currently visited faces (buffer miss), it is computed and stored in the buffer. Different with GPU VF-Ray, we simply groups rays starting from nearby pixels in the image plane. We observed that this simple approach shows high ray coherency while avoiding expensive projection operations and point-in-face tests in VF-Ray. Each ray groups is processed by a thread while using its local buffer.

To improve the utilization efficiency (i.e. hit ratio) of a small size buffer, we present a novel hash function. We observed that depth based hash function (Z-hash) used in VF-Ray causes congestion to a specific region of a buffer when faces are closely residing in a region. This drops the hit ratio of a buffer and it gets worse as the size of buffer decreased. To avoid much conflicts in a small buffer, we use the index of faces. We found that our *index-based hash function* makes slots of a buffer are more evenly utilized and we achieved up to 38% higher hit ratio than Z-hash when the buffer size is small (e.g., 16) (Fig. 1).

To further improve efficiency of a small buffer, we propose a novel traversal scheme, *group traversal*. In a ray-by-ray processing, the hit ratio of a small buffer inevitably low since the buffer can keep the VDFIs only for faces in far away after processing a ray. To maximally exploiting ray coherency, we make rays in a group traverse together step-by-step like a breath first traversal. In our group traversal, a ray is handled right after a nearby ray visit the nearest face and it may meet the same face with high probability. In our benchmarks, it improves the hit ratio up to 41% compared with ray-by-ray approach (Fig. 1).

3. Results

We have implemented our algorithm based on the Bunyk ray casting implementation of VTK [SML06]. We have optimized the VTK's implementation with an array-based data structure (*VTKopt*)

		Vertex	Faces	Tetra	VDFI
Dataset 1	#	647K	7,583K	3,773K	7,583K
	Mem.	5 MB	364 MB	121 MB	546 MB
Dataset 2	#	7,659K	85,051K	41,939K	85,051K
	Mem.	61 MB	4,082 MB	1,342 MB	6,124 MB

Table 1: *This table shows the number of vertexes, faces, and tetrahedrons, and required memory space for the benchmark datasets. The right most column shows the required memory space for the full VDFI list.*

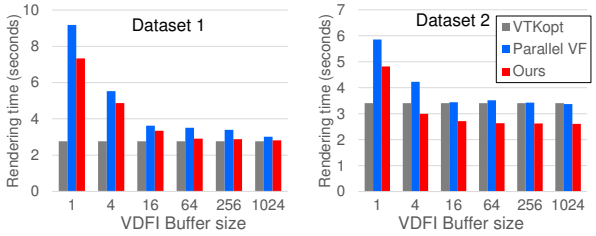

Figure 3: *These graphs show the performance of three different ray casting algorithms. For this experiment, we used eight CPU threads on a machine consisting of two Intel hexa-core CPUs (2.4 GHz) with 128GB memory.*

and we also have implemented Maximo. et al.'s method as a CPU version (*Parallel VF*). Then we applied these algorithms to two different benchmark datasets (Table 1).

Fig. 3 shows the performance of previous works and our method. Our method generally shows higher performance than *Parallel VF* and achieves a compatible performance with *VTKopt* by using a small size (e.g., 16-64 slots) buffer that takes less than 1% of memory compared with the full VDFI list. These results validate the benefit of our memory efficient algorithm.

In some cases, our method outperformed than *VTKopt* for a large data as shown in the right graph of Fig. 3. This is mainly because buffers in our method are small enough to utilizing CPU caches (e.g., L1 and L2) different with a huge full VDFI list. We also found that our method is good for taking advantage of the early ray termination optimization [Lev90]. Ray casting algorithms usually takes several steps before updating the color of a pixel to minimize context switch overhead. On the other hand, our step-by-step group traversal has proper workload for color integration after few steps (e.g., one or two) and it can terminate traversal for a ray as soon as the opacity of the pixel meets a threshold. We expect this performance inversion becomes more clear as the size of data increased and it also demonstrates that our method suitable for handling a large dataset.

4. Future Work

Although we currently apply our approach only to multi-core CPUs, our small and high efficient local buffer may fit to GPU architectures having limited memory space. We would like to extend our method to massively parallel algorithm on GPUs.

Acknowledgment

We would like to thank anonymous reviewers for their constructive feedbacks. This work was supported by the National Research Council of Science & Technology (NST) grant by the Korea government (MSIP) (CMP-16-03-KISTI).

References

[BKS97] BUNYK P., KAUFMAN A., SILVA C. T.: Simple, fast, and robust ray casting of irregular grids. In *Scientific Visualization Conference, 1997* (1997), IEEE, pp. 30–30. 1, 2

[BW01] BRODLIE K., WOOD J.: Recent advances in volume visualization. In *Computer Graphics Forum* (2001), vol. 20, Wiley Online Library, pp. 125–148. 1

[Lev90] LEVOY M.: Efficient ray tracing of volume data. *ACM Transactions on Graphics (TOG) 9*, 3 (1990), 245–261. 2

[MRB*08] MAXIMO A., RIBEIRO S., BENTES C., OLIVEIRA A. A., FARIAS R. C.: Memory efficient gpu-based ray casting for unstructured volume rendering. In *Volume Graphics* (2008), pp. 155–162. 1, 2

[RMB*07] RIBEIRO S., MAXIMO A., BENTES C., OLIVEIRA A., FARIAS R.: Memory-aware and efficient ray-casting algorithm. In *Computer Graphics and Image Processing, 2007. SIBGRAPI 2007. XX Brazilian Symposium on* (2007), IEEE, pp. 147–154. 1

[SML06] SCHROEDER W., MARTIN K. M., LORENSEN W. E.: *The Visualization Toolkit (4th ed.): An Object-Oriented Approach to 3D Graphics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. 2