Accurate and Memory-Efficient GPU Ray-Casting Algorithm for Volume Rendering Unstructured Grid Data

Gibeom Gu[1] Duksu Kim[2]

[1]KISTI (Korea Institute of Science & Technology Information) [2]KOREATECH (Korea University of Technology and Education)

Abstract

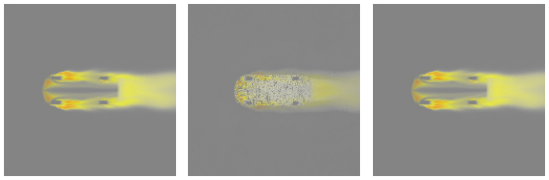
We present a novel GPU ray-casting algorithm for volume rendering unstructured grid data. We employ the per-pixel intersection list to guarantee accurate results for non-convex meshes. For efficient memory access for the lists on the GPU, we represent the intersection lists for all faces as an array. To increase ray-coherency in a thread block and improve memory access efficiency, we propose an image-tile based ray distribution method. Also, we found that a prior approach using a per-thread local buffer to reduce redundant computation is not proper for recent GPUs. Instead, we use an on-demand calculation strategy that achieves much better performance even when it allows duplicate computation. With a GPU, our method achieved up to 36.5 times higher performance for the ray-casting process, and 18.1 times higher performance for the entire volume rendering, compared with the Bunyk algorithm using a CPU core. Also, our approach showed up to 8.2 times higher performance than a GPU-based cell projection method while generating more accurate rendering results.

1. Introduction

Direct volume rendering (DVR) is widely used in various fields including medical imaging, scientific simulations, and so on. One of the most popular algorithms for DVR (hereafter called 'volume rendering') is ray-casting due to its generality and accuracy. For a uniform grid, ray-casting is relatively easy to implement, and various acceleration techniques have been well studied including parallel algorithms using multi-core CPUs and GPUs [BW01].

Unlike the uniform grid, an unstructured grid has an irregular structure, which makes it more complicated to perform ray-casting, requiring a significant amount of computation. To handle this high computational overhead, projection-based methods (e.g., cell projection) have been proposed [SLSM06, MMFE06, MA04, CICS05]. Cell projection methods can directly utilize the high polygon rasterization performance of the GPU, and shows much higher performance than CPU-based ray-casting methods. However, the rendering results can include artifacts depending on the accuracy of visibility ordering. There have been attempts to implement the ray-casting algorithm on the GPU to perform accurate volume rendering [WKME03, BPCS06, MHDH07, MHDG11]. One recent work [MRB*08] proposed a GPU ray-casting algorithm (i.e., VF-GPU) for a modern GPU architecture designed for GPGPU. Although VF-GPU showed up to five times higher performance improvement than using a CPU, their approach does not guarantee accurate results for non-convex meshes [Kim17]. Also, we found that VF-GPU achieves a limited performance improvement on current GPUs having different characteristics (e.g., L1/L2 caches and much more cores) with the GPUs used when VF-GPU developed.

We present a novel GPU ray-casting method for volume rendering unstructured grid data. Our method has the following benefits:



(a) Bunyk [BKS97] (b) HAVS [CICS05] (c) Ours

Figure 1: *These figures compare the rendering results of three algorithms for Dataset 3, CFD simulation data for a car.*

- **Accurate:** Unlike cell projection methods, our method generates accurate rendering results (Fig. 1). Also, our method garantees accurate results for non-convex meshes unlike VF-GPU.
- **Memory and cache efficient:** Our on-demand VDFI (view-dependent face information) calculation strategy decreases both memory and cache transactions with a higher cache hit ratio while taking less memory space than VF-GPU using thread-local VDFI buffers. Furthermore, our image-tile based ray distribution method further improves cache utilization efficiency.
- **High ray-casting performance:** Our methods showed up to 8.2 times higher volume rendering performance than a GPU-based cell projection method. Also, it achieved up to 36.5 times higher ray-casting performance than a CPU-based method.

2. Our Approach

To guarantee accurate rendering results for non-convex meshes, we employ the per-pixel intersection list used in *Bunyk algorithm* [BKS97]. A straightforward approach for building the lists is to use a linked-list for each pixel since it is easy to add a new projected face to the lists with insertion sort [YHGT10]. However,

DOI: 10.2312/eurp.20191150

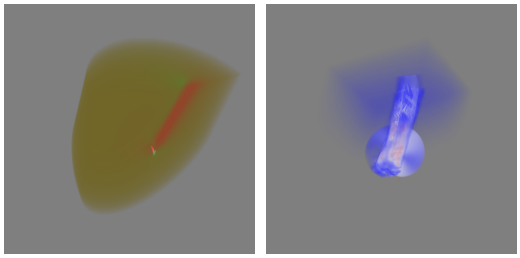


Figure 2: *These figures show the volume rendering results of our method for Dataset 1 (left) and 2 (right).*

	Vertex	Faces	Tetra	Boundary
Dataset 1	647,073	7,583,488	3,773,943	71,614
Dataset 2	7,320,994	83,385,432	41,257,368	1,741,392
Dataset 3	7,659,517	85,051,683	41,939,085	2,347,026

Table 1: *This table shows the different statics of each benchmark.*

the linked-list is not appropriate for the GPU since it leads to irregular memory access. Therefore, we represent the intersection lists of all pixels as an array.

In our method, a thread block processes given ray groups. To improve ray-coherency in a ray group, we propose a novel ray distribution method that groups rays from an image-tile, which is a square image having a certain size. We use a 2D thread block having the same size as an image-tile, and each thread handles a ray. With the Night Profiler, we found that ray-casting with our distribution method took less L2 cache transactions (e.g., 21%) than using a naive pixel-order ray distribution approach.

During the ray-casting process, we take an on-demand VDFI computation strategy unlike a prior GPU ray-casting algorithm, VF-GPU [MRB*08]. Bunyk algorithm computes and stores VDFIs for all faces as a preprocessing to remove redundant calculation. To take the benefit of the Bunyk algorithm while reducing the memory overhead, VF-GPU used a thread-local VDFI buffer having a specific size. It maintains recently computed VDFIs, and a thread reuses them if the required VDFI is in the buffer. However, we found that it is hard to take advantage of the buffers unless the buffer size is large enough to guarantee a high hit ratio for all threads. Since GPU threads run in a unit of warp (e.g., 32 threads), the work of threads in the warp serialized if one or more threads in a warp fail to find the required VDFI in the buffer. This warp divergence invalidates the benefit of buffer hits for other threads, and the cost of buffer handling becomes overhead. Unfortunately, it is hard to reserve enough memory space for all buffers because we need to launch hundreds of thousands of threads to fully utilize massive parallelism of recent GPUs. In our experiments, we also found that calculating VDFI whenever a ray meets a face achieves much better performance than using thread-local buffers even though it can result in duplicate computation. We observed that, compared with the on-demand calculation, using buffers has up to 1.5 times more memory access and up to 4.5 times more L2 cache transactions, while decreasing L1 cache hit ratio by about 18%. As a result, our method shows up to 3.9 times higher ray-casting performance compared with using VDFI buffers (e.g., 256 slots) for Dataset 1, which is a small dataset and we can reserve relatively large space for a buffer. Based on these theoretical and experimental observations, our method takes the on-demand VDFI calculation strategy.

(seconds)	Bunyk	Bunyk-Parallel	HAVS	Ours
Dataset 1	10.31	2.93	1.04	0.57
Dataset 2	50.65	15.52	12.44	4.83
Dataset 3	5.31	2.53	10.62	1.29

Table 2: *This table shows the average rendering time (in seconds) per frame for three benchmark datasets (Table 1).*

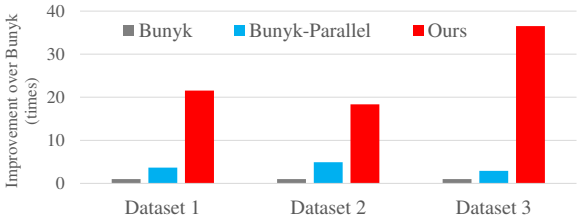


Figure 3: *This figure shows the performance improvements for ray-casting process of two algorithms over Bunyk (baseline).*

3. Results

We implemented our method (*Ours*) in a system with a GPU (Nvidia GTX 1080) and a quad-core CPU by using CUDA. To compare the efficiency and accuracy of our approach, we also implemented a Bunyk algorithm that uses a CPU core (*Bunyk*), a CPU parallel Bunyk algorithm that uses four CPU cores (*Bunyk-Parallel*) based on Kim [Kim17] by using OpenMP, and one of the most well-known GPU-based cell projection method (*HAVS*) proposed by Callahan et al. [CICS05].

Table 2 shows the average volume rendering time per frame of the prior works and our method. Our method shows up to 18.1 and 5.1 times higher performance than *Bunyk* and *Bunyk-Parallel*, respectively. *HAVS*, which uses the same GPU as ours, also exhibited 2.8 and 1.2 times higher performance than *Bunyk-Parallel* for Dataset 1 and 2, respectively. However, our method generally achieved better performance (e.g., up to 8.2) than *HAVS*. In Dataset 3, cells are crowded in specific regions, and we can avoid unnecessary ray traversal with the early ray termination method [Lev88]. Therefore, for Dataset 3, even *Bunyk-Parallel* showed better performance than *HAVS*, which needed to consider all faces. These results validate the benefits of our approach.

Ray-casting performance: To check the benefit of our GPU ray-casting algorithm, we measured the processing time just for the ray-casting process, while excluding common processing time on the CPU, like the intersection list construction time. As shown in Fig. 3, our method achieved up to 36.5 times (25.5 times on average) higher ray-casting performance than *Bunyk*. More specifically, ours showed much higher performance improvement for Dataset 3. This is because rays visit a small ratio of faces during the actual ray-casting process, and the preprocessing time for computing VDFIs for all faces becomes overhead rather. These results demonstrate the efficiency of our GPU ray-casting algorithm.

Acknowledgments

We would like to thank anonymous reviewers for their constructive feedbacks. Duksu Kim is a corresponding author, and this work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2018R1C1B5045551 and No.CMP-16-03-KISTI), and the Korea Institute of Science and Technology Information (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

[BPCS06] BERNARDON F. F., PAGOT C. A., COMBA J. L., SILVA C. T.: GPU-based tiled ray casting using depth peeling. *Journal of Graphics tools 11*, 4 (2006), 1–16. 1

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

[CICS05] CALLAHAN S. P., IKITS M., COMBA J. L. D., SILVA C. T.: Hardware-assisted visibility sorting for unstructured volume rendering. *IEEE Transactions on Visualization and Computer Graphics 11*, 3 (2005), 285–295. 1, 2

[Kim17] KIM D.: Memory efficient parallel ray-casting algorithm for unstructured grid volume rendering. In *Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Posters* (2017), Eurographics Association, pp. 13–15. 1, 2

[Lev88] LEVOY M.: Display of surfaces from volume data. *IEEE Computer graphics and Applications 8*, 3 (1988), 29–37. 2

[MA04] MORELAND K., ANGEL E.: A fast high accuracy volume renderer for unstructured data. In *2004 IEEE Symposium on Volume Visualization and Graphics* (2004), IEEE, pp. 9–16. 1

[MHDG11] MUIGG P., HADWIGER M., DOLEISCH H., GROLLER E.: Interactive volume visualization of general polyhedral grids. *IEEE transactions on visualization and computer graphics 17*, 12 (2011), 2115–2124. 1

[MHDH07] MUIGG P., HADWIGER M., DOLEISCH H., HAUSER H.: Scalable hybrid unstructured and structured grid raycasting. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1592–1599. 1

[MMFE06] MARROQUIM R., MAXIMO A., FARIAS R., ESPERANCA C.: Gpu-based cell projection for interactive volume rendering. In *2006 19th Brazilian Symposium on Computer Graphics and Image Processing* (2006), IEEE, pp. 147–154. 1

[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

[SLSM06] SHAREEF N., LEE T.-Y., SHEN H.-W., MUELLER K.: An image-based modeling approach to gpu-based unstructured grid volume rendering. *In Proc. of Volume Graphics* (2006), 31–38. 1

[WKME03] WEILER M., KRAUS M., MERZ M., ERTL T.: Hardware-based ray casting for tetrahedral meshes. In *IEEE Visualization, 2003. VIS 2003.* (2003), IEEE, pp. 333–340. 1

[YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. *Computer Graphics Forum 29*, 4 (2010), 1297–1304. 1