Cache-Aware Iso-Surface Volume Rendering with CUDA

Junpeng Wang*
Virginia Tech

Fei Yang†
Chinese Academy of Sciences

Yong Cao‡
Virginia Tech

Abstract

Most of the GPU-based ray casting algorithms map volume data to 3D texture of the GPU, so that the hardware-accelerated tri-linear interpolation could be taken advantage of. However, in hardware, texture cache is implemented in a 2D fashion. As a result, the viewing direction has a significant effect on the texture cache hit rate. This paper presents a new sampling strategy, i.e. warp marching, for the ray casting algorithm. The new strategy samples volume data in a cache-friendly manner and displays a novel computation-to-core mapping. We apply it to the existing iso-surface volume rendering algorithm and demonstrate significant performance enhancements in certain viewing directions.

1 Introduction and Related Work

Ray casting algorithm is inherently a parallel algorithm, where each casted ray can be calculated in parallel without computational dependency. The algorithm, therefore, is well adopted to graphics processing units (GPUs).

Due to the fast hardware-accelerated tri-linear interpolation, 3D texture becomes the best choice to store volume data. However, the optimization of texture cache is limited in 2D space [2]. As a result, cache performance in some viewing directions is poor. Existing algorithms try to hide the cache penalty by reorganizing volume data into small bricks with varying orientations [5] or by dynamically adjusting thread block shape according to viewing direction [4]. These solutions, however, did not noticeably improve cache performance when facing certain directions. Here, we address this problem by introducing a new cache-friendly sampling strategy.

2 Motivation

There are different variations of the GPU-based volume ray casting where, for most cases, one ray is mapped to one GPU thread. Such basic strategies have been taken for granted, and can be seen in recent literature [1]. We call this mapping of rays to threads the **Standard** method. At any given instance, the samples of all rays are fitted into a plane, *sampling plane*. The algorithm accesses all these samples on the plane in parallel. It then marches this plane from front to back along a given camera orientation.

In CUDA, threads are organized as grids of blocks. 32 threads form an atomic group, i.e. warp, when mapped to GPU hardware. To achieve a good cache performance, memory stride inside such an atomic group of threads should be minimized [2,4]. Volume data mapped to 3D texture is organized as many 2D slices, and z-curve is used to optimize the spatial locality inside each slice. When the sampling plane is aligned with these 2D slices (Figure 1, *facing XY*), the cache hit rate is high. But, when the plane is perpendicular to these 2D slices (Figure 1, *facing ZY*), the cache performance is poor, because samples cover different slices and are far from each other in the memory space.

*e-mail: junpeng@vt.edu
†e-mail: yangfei09@mails.gucas.ac.cn
‡e-mail: yongcao@vt.edu

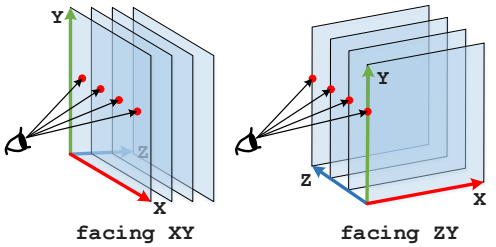


Figure 1: Effects of different viewing directions.

3 Warp Marching

We propose a new sampling strategy that demonstrates a different computation-to-core mapping. Compared to the standard method (Figure 2 *a*) which assigns one thread to a ray, this strategy assigns one warp of threads to a ray (Figure 2 *b*). To simplify the illustration, we assume one warp has four threads. At any given time, the thread warp samples multiple points along the ray in parallel. Then, it marches along the ray direction in this manner from front to back until it finds the iso-surface. Because the algorithm marches a warp size of samples at a time, we call it *warp marching*.

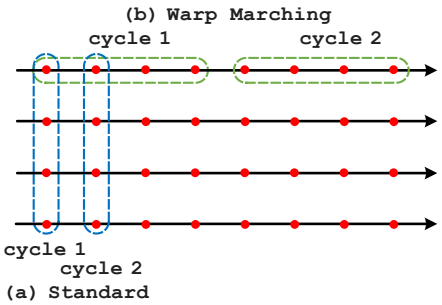


Figure 2: Different sampling strategies (four threads per warp).

The iso-surface volume rendering algorithm can be broken down into three steps: (1) ray bounding-box intersection tests; (2) casting rays through the volume to find iso-value positions; (3) shading on these positions. In our implementation, each step is a CUDA kernel. We only focus on the second kernel to apply warp marching, since it is the performance bottleneck. To handle the new computation-to-core mapping strategy, three cases need to be considered according to three possible relationships between the densities of a warp of samples and the iso-value:

a) density values are all lesser than the iso-value;
b) density values are all larger than the iso-value;
c) some density values are lesser than the iso-value, while others are larger than the iso-value.

We give each thread of the warp a flag bit. If the sample's density is lesser than the iso-value, the bit is set to 0, otherwise it is set to

1. Then the warp voting function, i.e. `__ballot()`, can be used to quickly get the state of the whole warp. For case *a*, the warp voting function returns $0x0$ (four threads per warp). For case *b*, it returns $0xf$. In both cases, the warp of samples are simply skipped. Finally rays can march through spaces that do not contain the iso-value. For case *c*, the warp voting function returns a value between $0x0$ and $0xf$. This is the case where the iso-value is detected. Here, we need to identify the thread that takes the sample around the iso-value position (apply `__clz()` to the value returned from `__ballot()`), and have it do a linear interpolation to find the iso-surface.

Two special cases which do not occur in any of the situations described so far need to be considered carefully. The first one is when the iso-surface is located in the gap between two cycles of a warp (Figure 3 *left*). The first cycle of the warp gets state $0x0$ and skips the warp of samples. The second cycle of the warp gets state $0xf$ where it again skips all samples. Thus warp marching has missed the iso-value completely. Such a condition can be managed by having the warp walk back one sample, so that it becomes the case *c* we discussed before. The second special case involves the boundary condition. When $density \geq iso\text{–}value$, we set the flag bit to 1. However warp marching cannot detect the iso-value if it is on the first sample of the warp (Figure 3 *right*). To address this, warp marching tests whether the first sample's density is equal to the iso-value, in the first cycle. This problem will not happen in the second cycle and cycles after, since warps go back one sample in every marching step.

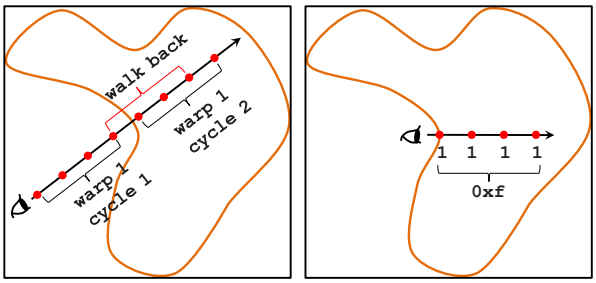
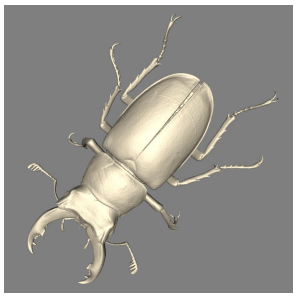


Figure 3: Special cases of iso-surface warp marching.

4 Result

beetle (16-bit, 0.65 GB)
832×832×494



bat (8-bit, 1.12 GB)
906×911×1466

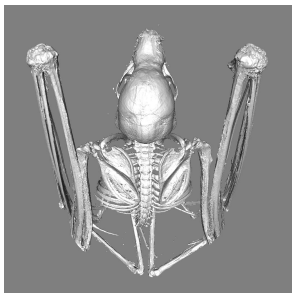


Table 1: Experiment data sets and dimensions.

Results in this section are collected from an NVIDIA GeForce GTX TITAN GPU. Texture cache hit rates are collected by CUPTI [3]. Two experiment data sets (Table 1) are rendered in a resolution of 1024×1024 respectively. To reflect the distance between rays, we record the projected resolution of volumes (listed under the name of data sets in Table 2). This resolution and the volume dimension together decide the distance between samples on the sampling plane. For example, the *X* dimension of *beetle* is 832, and the projected length of this dimension is 875, so distance between rays is 0.95 voxel length. For *bat*, this distance is 1.57. To keep the distance constant during marching, an orthographic projection is applied. Sampling distance along rays is fixed at 0.3 voxel length.

When facing *XY* plane of the volume, the standard method is always better than the warp marching. A warp of threads in this mapping strategy take samples on the same 2D slice, and these samples are close to each other in the memory space. Warp marching, on the contrary, takes samples across slices. However, since the distance along rays is fixed at a small constant value, it also achieves real-time rendering performance (though not as good as the standard method). When facing *ZY* plane, warp marching is significantly faster than the standard method. In this direction, texture cache hit rate of the standard method is very low, because threads of the same warp take samples from different slices of the volume. The memory gap goes beyond the cache size, consequently resulting in poor rendering performance. In the same orientation, samples of the same warp in warp marching are taken from the same slice and the distance between them is fixed at 0.3 voxel length. So, warp marching significantly outperforms the standard method.

The viewing direction effect is extraordinarily significant in the demonstrated results. One reason is the large size of the experiment data sets. Also, the results are collected without any optimizations of the ray casting algorithm. Some optimizations, such as empty space leaping, might be able to mitigate the viewing direction effect, since they reduce texture fetching operations.

	Facing *XY* Plane		Facing *ZY* Plane	
	beetle 875×875	bat 576×579	beetle 520×875	bat 932×579
Standard	94.67%	94.13%	56.32%	33.13%
Warp Marching	78.58%	75.33%	95.83%	97.38%
Speedup	0.83	0.80	1.70	2.94
Standard	39.62	30.13	9.80	2.10
Warp Marching	31.92	25.81	35.99	29.49
Speedup	0.81	0.86	3.67	14.04

Table 2: Texture cache hit rate (top) and fps (bottom).

5 Conclusion and Future Work

We introduce a new sampling strategy for the texture-based iso-surface volume rendering. The strategy takes advantage of cache coherence and increases rendering performance at certain camera orientations.

Applying warp marching to other types of GPUs, even those with varying warp sizes, is a promising direction for future exploration. Also, a hybrid approach that adjusts sampling strategy based on viewing direction is worth trying.

References

[1] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. R. Salama, and D. Weiskopf. Real-time volume graphics. In *ACM Siggraph 2004 Course Notes*, page 29. ACM, 2004.

[2] NVIDIA. Cuda c programming guide. *NVIDIA Corporation*, 2013.

[3] NVIDIA. Cupti user's guide. *NVIDIA Corporation, July*, 2013.

[4] Y. Sugimoto, F. Ino, and K. Hagihara. Improving cache locality for ray casting with cuda. In *ARCS Workshops*, pages 339–350, 2012.

[5] D. Weiskopf, M. Weiler, and T. Ertl. Maintaining constant frame rates in 3D texture-based volume rendering. *Proceedings Computer Graphics International, 2004.*, pages 604–607.