Evaluation and Improvement of GPU Ray Tracing with a Thread Migration Technique

Xingxing Zhu and Yangdong Deng
Institute of Microelectronics, Tsinghua University, Beijing, China
Email: zhuxingxing0107@163.com, dengyd@tsinghua.eud.cn

***Abstract*—Ray tracing is a computer graphics rendering technique. Different from the traditional rasterization algorithm, the ray tracing algorithm simulate the real vision process. Being able to deliver highly realistic graphics effects, it has been considered as the fundamental graphics rendering mechanism for high-end applications and is also likely to be adopted as the work-horse of future graphics hardware. However, the high computational effort is the major stumbling block for ray tracing to be deployed in real-time applications. In this paper we present a detailed characterization of the ray tracing algorithm by focusing on ray tracing algorithm based on BVH acceleration structure. Using a state of art graphics processor unit (GPU) simulator, we are able to provide key insight on improving the performance of ray tracing on GPUs. We propose thread migration technique to reduce the cache miss rate and enhance the processing throughput of primary and secondary rays.**

***Index Terms*—ray tracing, graphic processing unit, thread migration**

I. Introduction

Ray tracing is a computer graphics rendering technique [1]. Different from the traditional rasterization algorithm [2], it is based on the vision formation process. Since it is able to deliver highly realistic graphics effects, it has been considered as the technology of choice for high-end graphics. In addition, the ray tracing algorithm has a strong potential to be adopted as the work-horse of future consumer level graphics hardware. So far, it has been widely used in such fields as engineering design, film industry, entertainment, gaming, computer aided design(CAD) and movie production. Especially, ray tracing based graphics have appeared in many Hollywood movies like “Happy Feet”, “The Lord of the Rings” and “Avatar”.

However, the high computational demand is the major stumbling block for ray tracing to be adopted in real-time applications. Therefore, now the ray tracing technology can only be used in off-line mode. In this paper we present a detailed characterization of the ray tracing algorithm based on BVH acceleration structure. Using a state of art Graphics Processor Unit (GPU) simulator, we are able to provide key insight on improving the performance of ray tracing on GPUs. We also propose a thread migration technique. The thread migration reduced the cache miss rate and improved the speed of the ray traversal.

The contribution of the paper is that the proposed techniques significantly reduces cache miss rate. The miss rate of the primary ray was reduced by 34% to 51%, and the rate of the secondary ray was decreased by 62% to 76%. The resultant speed of the primary ray traversal is improved by 1.29 to 1.74 fold, while the speed of the secondary is improved by 1.85 to 2.59 X.

To explore the performance of our algorithms, we implemented our algorithms in CUDA(Compute Unified Device Architecture) and benchmarked their performance on an NVIDIA GeForce GTX 560Ti GPU. We also used GPGPU-Simversion 3.x as our GPU emulator. The emulator configuration is based on the Fermi architecture with 15 SM(Streaming Multiprocessor, SM) cluster, each cluster contains 10 SM cores.

II. Background

A. Ray Tracing Algorithm

The basic idea of ray tracing is to trace the light from the observation point through each pixel on the view plane to the objects in the scene. We compute the color of the pixel according to the reflection and refraction characteristics of the surface and the light source. The light from the observation point to each pixel of the view plane is called the primary ray. The light generated at the intersection point of the scene according to the laws of reflection and refraction of the light is called the secondary ray.

The essential stage of ray tracing algorithm is to search the objects that each ray intersects. The scene is composed of a variety of different elements such as triangles, polygons, polyhedrons, sphere, etc.. Among those elements triangle is the most commonly used, which is also compatible with the rasterization algorithm primitive. A moderate size scene contains 50K to 10M triangles. In order to improve space traversal performance, several acceleration data structures are widely used. It can reduce the complexity of algorithms to $O\log(N)$ [3], while N is the number of triangles in the scene. The commonly used acceleration structures in ray tracing are Bounding Volume Hierarchy (BVH)[4]

Manuscript received May 8, 2013; revised August 5, 2013

doi: 10.12720/ijsps.1.1.111-115

and kd-Tree [5]. BVH is a tree structure to organize the Axis-Aligned Bounding Box (AABB) of special geometric objects hierarchically. Kd-Tree is a space–partitioning data structure for organizing points in a k-dimensional space. Generally the performance of ray traversal on kd-Tree is faster than on BVH [6], but BVH is more suitable for dynamic scenes for the coordinates of the AABB can be updated without reconstructing the whole tree. Therefore, we use BVH as the acceleration structure in this paper. General-purpose

B. （General-purpose）Graphic Processing Unit

GPU(Graphics Processing Unit, GPU) is a graphic processing hardwarewhich was designed for rasterization algorithm. Modern GPU are mostly programmable so that is can also be used for other parallel computing applicationsGPU is a highly parallel, multithreaded, manycore processor with tremendous computational power and memory bandwidth. The NVIDIA Fermi architecture has 512 CUDA(Compute Unified Device Architecture) core, each CUDA core can operate one integer or floating-point instruction per clock cycle. There are 16 SM coreson one chip.Each of them contains 32 CUDA cores. All threads in GPU are organized into blocks, and a certain number of threads in the block are organized into a warp. The warp can be mapped to the SM.

III. QUANTITATIVE ANALYSISOF THE RAY TRACING ALGORITHM

A. Test Scene

In the paper we use our implementation of TimoAila's open source GPU ray tracer [7] to analyze some detailed characterization of ray tracing algorithm.In order to get the implementation details of the underlying hardware, we used the traditional GPU emulator GPGPU-Sim. The detail information is shown in Fig. 1 and Table I.

B. The Cache Miss Rate

In the many-core architecture, the global memory access latency will increase greatly with the increase of the numbers of SM cores, which is influenced by on-chip interconnection network and the limitation of the global memory port, and thus affect the scalability of the many-core architecture.

For many-core architecture, the memory access latency is (described) in equation (1).

$$T = t_{L1} + \alpha_{L1}\left[t_{traf} + t_{L2} - t_{L1} + +\alpha_{L2}(t_{dram} - t_{L2})\right] \quad (1)$$

t_{L1} is the access latency to the cache in the core, t_{L2} is the access latency to the global memory-side secondary cache, t_{dram} is the access latency to the global memory, t_{traf} is the network latency from the SM core to the global memory, α_{L1} and α_{L2} are the cache miss rate.

Fig. 2, Fig. 3 and Fig. 4 show the simulation results of the ray tracing program on the GPGPU-Sim emulator about the *conference room* scene. Each cluster contains 15 SM cores.

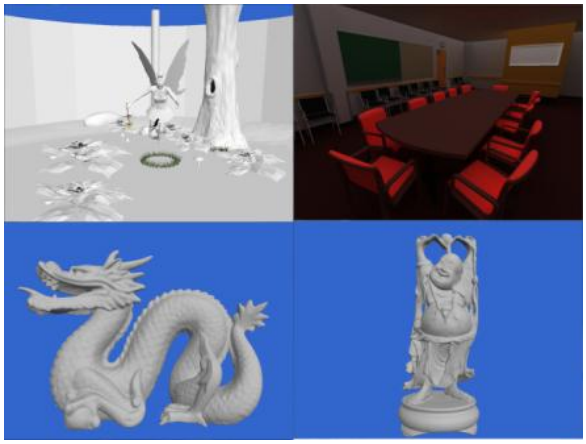

Figure 1. Test scenarios(top-left: fairy forest, top-right: conference room, bottom-left: Stanforddragon, bottom-right: happy Buddha), the render resolution is 1024*768.

TABLE I. TEST SCENARIOS

Scene	Vertex	Triangle	BVH	
			Node	Leaf
Fairy Forest	100737	174117	57519	57519
Conference Room	166907	282759	103801	103801
Stanford Dragon	435545	871414	299504	299504
Happy Buddha	543775	1087425	385901	385901

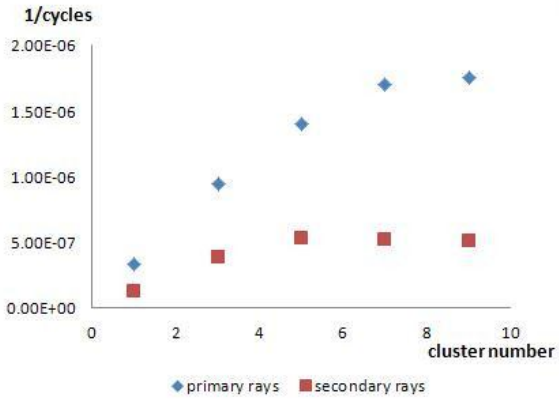

Figure 2. Execution speed of the primary and secondary rays on the conference room scene

Fig. 2 shows the total processing speed significantly deviates from the linear. The secondary rays even start to decline, when the number of the core cluster is more than 5. Fig. 3 shows that the number of the global memory access increaseswith the increase number of the SM cores, and thus lead to the increase of the network latency (t_{traf}). Due to t_{L1}, t_{L2}, t_{dram} are fixed memory latency, we can only optimize the cache miss rate. Consider the size of the cache is limited by the chip area, we try to change the way of access to reduce the cache miss rate.

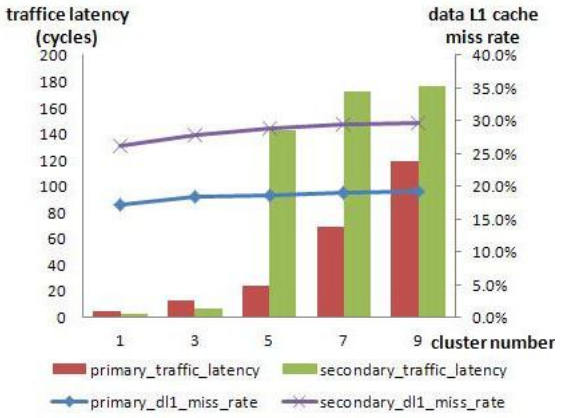

Figure 3. Network latency and the cache miss rate

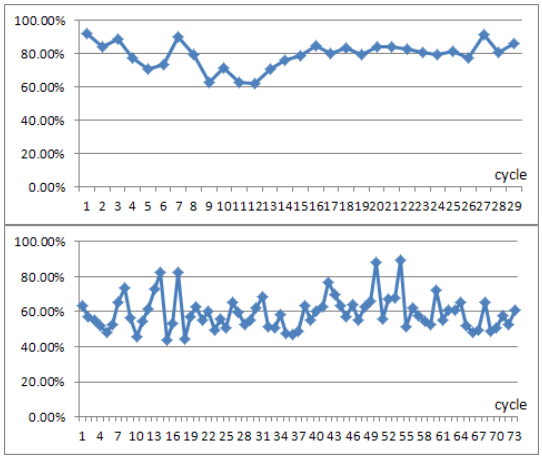

Figure 4. The cache miss rate about the conference room scene(top: the primary ray cache miss rate, bottom : the secondary ray cache miss rate)

Fig.4 shows that the cache miss rate of the primary ray is about 20%, and the cache miss rate of the secondary ray is about 40%. Such high degree of cache miss rate will result in frequent replacement (thrashing) of the data in the cache, affecting the performance of the cache.

IV. Results and Analysis of the Ray Tracing Algorithm Based on Thread Migration

A. Thread Migration

Traditionally, a ray will be bound to a thread in the ray tracing program running on GPU. All the rays will be packaged and mapped to the SM cores, then do parallel computing on GPUs. That means we bind the threads with the SM core and the data, for each SM core to access is random. The random access to data will cause high degree of the cache miss rate and will greatly restrict the program's performance. Otherwise， the information of the ray tracing program for each thread only includes the light source, direction and other relevant information, which is very simple. So we solve the problem from another standpoint. We bind data with the SM core, and the thread executes through the scheduling assignment. This is the thread migration [8].

In the thread migration, the memory address is divided into a few section, each section corresponds to a SM core. Only the corresponding SM core can access the data in the section. In the thread migration, the thread will migrate between the SM cores according to the accessed data. For example, if the thread(i)running on the SM core(t) needs to access the data address corresponding to the SM core(j)($i \neq j$), the information of thread(i) will be migrated to SM core(j) .

We design a thread migration program which takes the advantage of the characteristics of BVH traversal. We bind the sub-tree with the SM core and divide the traversal process into scheduling distribute stage and sub-tree traversal stage. During the scheduling stage, the thread will traversal the upper BVH tree, and will stop after it reaches the certain level of the BVH tree. Then the accessed sub-tree will be inserted into the waiting queue. During the sub-tree traversal stage, they will traversal the sub-tree using the BVH traversal algorithm. Fig. 5 shows the detailed information.

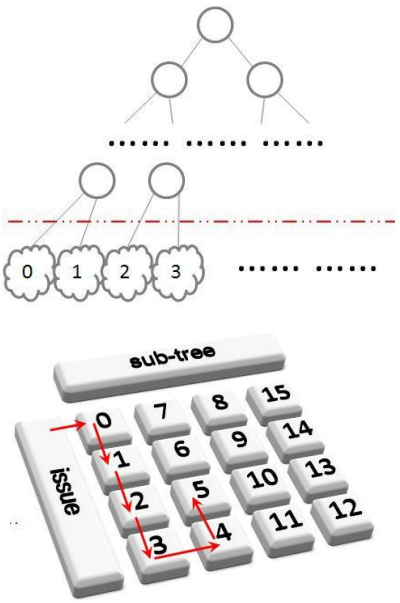

Figure 5. Scheduling distribute stage and sub-tree traversal stage(top: Scheduling distribute stage, bottom: sub-tree traversal stage)

B. Results and Analysis

We use the GPGPU-Sim to approximately simulate the ray tracing algorithm based on the thread migration, In our design, each sub-tree traversal program running on the SM core only can traversal the sub-tree. This reduces the competition of the different threads of the data in cache and alleviates the cache miss rate. However, we cannot directly specify the thread on different SM cores in GPGPU-Sim emulator, but can only through the automatic allocation of hardware and software. Therefore, it is difficult to achieve the separation of different sub-tree traversal thread.

We allocate a certain virtual cache to each sub-tree bounding on the SM core to achieve our goal. There are

m virtual caches, wherein m is the number of the sub-trees. Each virtual cache is as large as the real one andcorresponds to a curtain sub-tree. To run a sub-tree thread, only the corresponding virtual cache data can be accessed while the others cannot. But the data in the other virtual caches is still stored without being replaced. Thus the data is still available when we switch to another sub-tree thread. Due to the cache miss rate are the main problem the thread migration technique proposed to solve, the virtual cache can approximately simulate its behavior.

TABLE II. THE MISS RATE IN THE TRAVERSAL STAGE

Ray	Scene	Cache miss rate		
		Standard	Improved	Improved/ standard (%)
Primary ray	Fairy	0.2872	0.1409	49
	Conference	0.2875	0.1888	66
	Dragon	0.3558	0.1736	49
	Buddha	0.3884	0.2300	59
Secondary ray	Fairy	0.3891	0.0945	24
	Conference	0.3558	0.0979	28
	Dragon	0.4818	0.1827	38
	Buddha	0.5007	0.1598	32

TABLE III. THE SIMULATION TIME IN THE TRAVERSAL STAGE

Ray	Scene	Simulation cycle		
		Standard	Improved	Improved /standard (%)
Primary ray	Fairy	517168	399729	77
	Conference	297066	223943	75
	Dragon	386685	221619	57
	Buddha	291061	186805	64
Secondary ray	Fairy	1748566	693091	40
	Conference	1081839	583214	54
	Dragon	1581138	665433	42
	Buddha	3009938	1161525	39

Table II and Table III show the results simulated on the GPGPU-Sim. From the Table II we can see that the miss rate of the primary ray was reduced by 34% to 51%, and the rate of the secondary ray was decreased by 62% to 76%. From Table III we can see that the resultant speed of the primary ray traversal is improved by 1.29 to 1.74 fold, while the speed of the secondary is improved by 1.85 to 2.59 X. In addition, we can see that the accelerator ratio of the complex scenes is better than the simple scenes. This indicates that the thread migration technology has advantage in processing the complex scenes. The reason is that the random memory access in such cases is likely to cause the problem of clogging of memory port and interconnect network which can be solved by the thread migration technology.

V. CONCLUSION

In this paper we present a detailed characterization of the ray tracing algorithm by focusing on ray tracing algorithm based on BVH acceleration structure. Using a state of art graphics processor unit (GPU) simulator, we are able to provide key insight on improving the performance of ray tracing on GPUs. We propose thread migration technique to reduce the cache miss rate and enhance the processing throughput of primary and secondary rays.

ACKNOWLEDGMENT

The traversal part of our code is the implementation of TimoAila's downloadable ray tracing. And the author would like to explicitly thank the anonymous ICSAP reviewers.

REFERENCES

[1] A. Appel, "Some techniques for shading machine rendering of solids," in *Proc. AFIPS Conf.*, Washington DC, 1968, pp. 37-45.
[2] L. Szirmay-Kalos, T. Umenhoffer, B Toth, *et al*. "Volumetric ambient occlusion for real-time rendering and games," *IEEE Computer Graphics and Applications*, vol. 30, pp. 70-79, Jan. 2010.
[3] J. Pantaleoni and D. Luebke, "HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry," in *Proc. of the Conf. on High Performance Graphics*, 2010, pp. 87-95.
[4] H. Weghorst, G. Hooper, and D. P. Greenberg, "Improved computational methods for ray tracing," *ACM Transactions on Graphics*, vol. 3, pp. 52-69, Jan 1984.
[5] I. Wald, "Fast construction of SAH BVHs on the Intel many integrated core (MIC) architecture," *IEEE Trans. on Visualization and Computer Graphics*, vol. 18, pp. 47-57, Jan. 2012.
[6] A. Santos, J. M. Teixeira, T. Farias, *et al*., "Understanding the efficiency of KD-tree ray-traversal techniques over a GPGPU architecture," *International Journal of Parallel Programming*, vol. 40, pp. 331-352, June 2012.
[7] T. Aila and S. Laine, "Understanding the efficiency of ray traversal on GPUs," in *Proc. of the High Performance Graphics*, New York, 2009, pp. 145-149.
[8] M. Lis, S. K. Shim, O. Khan, and S. Devadas, "Shared memory via execution migration," presented at International Conference on Architectural Support for Programming Languages and Operating Systems, New York, 2011.

XingxingZhu received her B.S. degree in In School of Information Science and Engineering from the Southeast University, Nanjing, China in 2009. She is currently pursuing the M.S. degree in Institute of Microelectronics in the University of Tsinghua, Beijing, China. Her research interests mainly focus on ray tracing algorithm and general purpose computing on graphics processing hardware (GPGPU).

Yangdong Deng received his Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University, Pittsburgh, PA, in 2006. He received his ME and BE degrees in Electrical and ElectronicsDepartment from Tsinghua University, Beijing, in 1998 and 1995, respectively. His research interests include parallel electronic design automation (EDA) algorithms, parallel program optimization and general purpose computing on graphics processing hardware.