ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.10, No.14, 1–6

Node pre-fetching architecture for real-time ray tracing

Jeong-soo Park[1)], **Woo-chan Park**[2)], **Jae-Ho Nah**[1)], **and Tack-don Han**[1a)]

[1] *Department of Computer Science, Yonsei University, 137 Shinchon-dong, Seodaemoon-gu, Seoul, South Korea*

[2] *Department of Computer Engineering, Sejong University, 98 Gunja-dong, Gwangjin-gu, Seoul, South Korea*

a) *hantack@msl.yonsei.ac.kr*

Abstract: The traversal process in accelerated ray tracing algorithms requires many memory transactions of an acceleration structure. We present a pre-fetching system to pre-load potential node data into a cache. Experimental results show that the proposed scheme increases the performance of a cache system by reducing the cache miss rate.

Keywords: ray tracing, computer graphics, acceleration structure, cache memory

Classification: Electron devices, circuits, and systems

References

[1] I. Wald, S. Boulos and P. Shirley: ACM TOG **26** (2007) 6.
[2] D. R. Horn, J. Sugerman, M. Houston and P. Hanrahan: Symp. I3D (2007) 167.
[3] J. Nah, J. Park, C. Park, J. Kim, Y. Jung, W. Park and T. Han: ACM TOG **30** (2011) 160.
[4] J. Bigler, A. Stephens and S. G. Parker: IEEE/EG Symp. IRT (2006) 187.
[5] J. Tse and A. J. Smith: IEEE Trans. Comput. **47** (1998) 509.
[6] H. Igehy, M. Eldridge and K. Proudfoot: ACM SIGGRAPH/EG HWWS (1998) 133.
[7] I. Wald and P. Slusallek: ACM SIGGRAPH/EG HWWS (2001) 21.
[8] W. Park, K. Lee, I. Kim, T. Han and S. Yang: IEEE Trans. Comput. **52** (2003) 1505.

1 Introduction

Ray tracing algorithms simulate the traversal of a ray through a three-dimensional scene, so that images can be synthesized with realistic global illumination effects such as shadows, reflection, and refraction. However, ray tracing algorithms consumes an enormous amount of computation and bandwidth when searching for the closest visible primitive and its hit point from a particular view point.

DOI: 10.1587/elex.10.20130468
Received June 17, 2013
Accepted June 28, 2013
Publicized July 09, 2013
Copyedited July 25, 2013

Previous researches have attempted to accelerate ray tracing algorithms by scene data pre-processing. Since it is expensive and wasteful to search for visible primitives by testing all of the primitives in a scene, hierarchical structures are used to prune unnecessary intersection tests. Though using an acceleration structure could reduce the burden of intersection testing, it still entails an enormous amount of randomized memory access to the acceleration structure itself. To remedy this problem, many researchers have leveraged parallel architectures and cache memory systems in SIMD-equipped CPUs [1], GPUs [2], and custom designed accelerators [3].

In this paper, we present a hardware based node pre-fetching technique for hierarchical acceleration structure to improve memory performance of hardware ray tracing accelerator. Since the traversal of a hierarchical structure involves storing some nodes in a stack to process them later, our proposed scheme generates additional requests to the cache for the nodes in the stack prior to use them. We simulated our scheme in the Manta ray tracer [4] and the DineroIV cache simulator. The result showed up to 19% improvement in cache performance without significant bandwidth overhead.

2 Background

In the last decades, many forms of acceleration structures have been proposed. Among them, kd-trees because of their good traversal performance [2] and bounding volume hierarchies (BVHs) because of their faster structure update [1] are primarily used. Since a kd-tree subdivides the scene spatially, its node traversing sequence is ordered front-to-back. Thus, whenever the intersected primitive is found, the kd-tree traversal process can be terminated even if the stack is not empty. In contrast, a BVH is an object subdivision tree. Due to overlapped nodes in the BVH, the BVH traversal process should be continued until the stack is empty to obtain the closest hit point.

The pre-fetching technique was originally developed for instruction cache architecture, allowing CPUs to retrieve instructions ahead of executions [5]. In the CPU design, the instructions are read-only and control flow, except branching, is quite predictable, that makes the pre-fetching technique very effective. In the field of computer graphics, pre-fetching is used in texture mapping to hide memory latency [6]. In this pre-fetch architecture, texels are immediately requested after a fragment is generated from the rasterization stage, before the texture mapping stage. Because the texture data is read-only and has high temporal locality, the texture pre-fetching technique effectively reduces the latency of texture accesses. There also exist prior research that utilizes pre-fetching instructions of commodity CPU for optimizing software ray tracer [7]. They pre-loads the scene data to hide network communication latency for distributed cluster rendering. As in their work, in order to use pre-fetching effectively, algorithms must be simple enough such that it can easily be predicted which data will be needed in the near future. They only addressed the expected potential of pre-fetching the scene data to be used in future frame without detailed explanation.

DOI: 10.1587/elex.10.20130468
Received June 17, 2013
Accepted June 28, 2013
Publicized July 09, 2013
Copyedited July 25, 2013

The proposed scheme predicts and pre-fetches the data to be used during single ray processing, while the method proposed by Wald et al. [7] pre-fetches the scene data for next frame. These make the proposed scheme more effective for real-time ray tracing architecture. Even though software ray tracer can take advantage of pre-fetching during single ray traversal, the pre-fetching thread could disrupt the execution of main rendering thread due to enormous thread switching. Because the independent hardware pre-fetching unit does not degrade the execution of main rendering process, the proposed scheme is much more suitable for ray tracing accelerator hardware.

3 Node pre-fetching architecture

Figure 1 illustrates an example of our proposed architecture. The traversal unit searches subsets of primitives in leaf nodes to check for intersections by searching the acceleration structure. The intersection test unit actually checks every primitive in the leaf node for its intersection by the ray. Since both units request data in external memory (Ext. Mem.), cache memories can be used to exploit the locality of the data. The proposed scheme adds a pre-fetching unit between an intersection unit and a traversal unit. The pre-fetching unit generates memory requests for the node data in the node stack. To achieve this, the stack requires an additional top pointer (pre-fetching pointer) to pass the next address on to the pre-fetching unit.

The proposed scheme works as follows: When the traversal unit inserts a node into the stack, it assigns the same position to the pre-fetching pointer with the conventional stack top pointer. After a ray meets a leaf node, the intersection test unit activates the pre-fetching unit while checking the intersection between the ray and the primitives in the leaf node. The pre-fetching unit pops the node address from the node stack using the pointer. Subsequently, the popped node address is sent to the node cache if the traversal unit does not execute any memory operations. The proposed scheme continues to demand the node addresses in the stack until every addresses in the

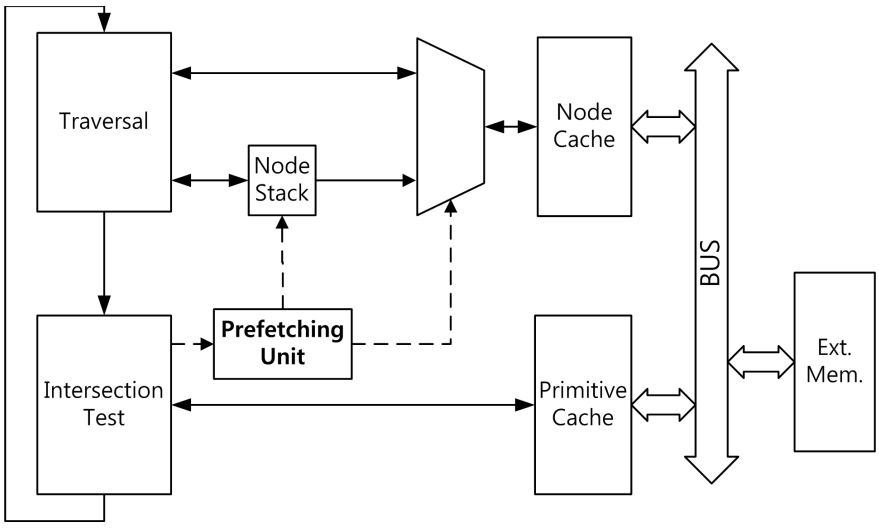

Fig. 1. Proposed node pre-fetching scheme.

DOI: 10.1587/elex.10.20130468
Received June 17, 2013
Accepted June 28, 2013
Publicized July 09, 2013
Copyedited July 25, 2013

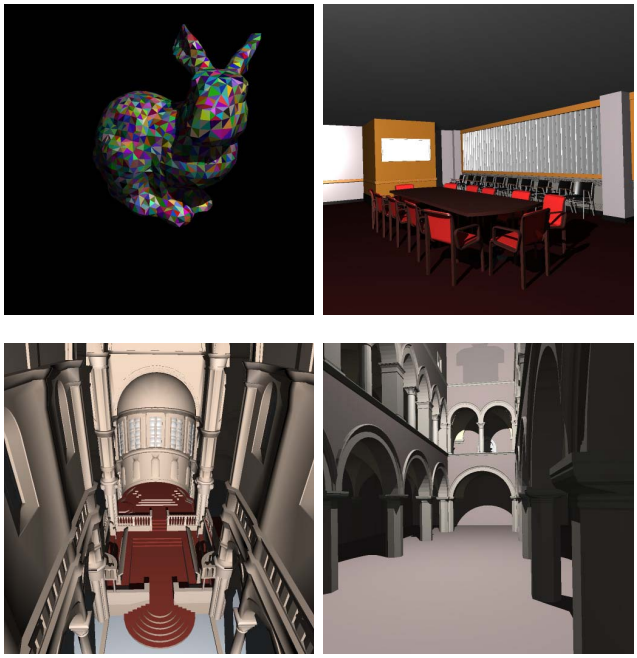

Fig. 2. Images from Test Scenes: Bunny (upper left), Conference (upper right), Sibenik (lower left), and Sponza (lower right).

stack is requested or the intersection test is ended. By doing so, the node data is loaded in the cache before we restart the traversal from the current node in the stack.

If the tracing of a ray finishes with a non-empty stack, the pre-fetched data in the stack can be useless and it could cause unnecessary bandwidth consumption. However, this bandwidth overhead is negligible. In the case of BVHs, tracing a radiance ray cannot be finished until the stack is empty. In the case of kd-trees, the number of used stack entries is usually not larger than three [2] because only intersected nodes are pushed to the stack. As the number of pre-fetched data in the stack is usually three or less, and it is not very burdensome.

4 Experimental results

To evaluate our proposed scheme, we performed the cache simulation with the DineroIV cache simulator over four well-known benchmarks (Figure 2): Bunny, Conference, Sibenik, and Sponza. The node memory request data were traced out using the Manta interactive ray tracer [4]. We shot a primary ray and a shadow ray per pixel. We performed cache simulation with 8 KB and 16 KB cache sizes with one-way (direct-mappped) and two-way set associativity configurations. The used block size used was 32 bytes. Table I shows the experimental results of the cache simulation and the ratios of the results with and without the pre-fetching scheme.

In addition to cache simulation, we also derived average traversal cycle per ray concept from performance measurement for traditional rasterization

DOI: 10.1587/elex.10.20130468
Received June 17, 2013
Accepted June 28, 2013
Publicized July 09, 2013
Copyedited July 25, 2013

Table I. Cache simulation results of the proposed scheme. A denotes results without the pre-fetching and B denotes results with the pre-fetching scheme. Assoc. stands for associativity.

Scene	Cache size (assoc.)	Miss rate(%) A	Miss rate(%) B	B/A (%)	Band-width(KB) A	Band-width(KB) B	B/A (%)	ACPR (Ratio(%))
KD-tree								
Bunny	8 K(1)	7.1	5.9	83.0	411	396	96.4	126.6/107.9(85)
	8 K(2)	3.6	2.9	80.7	208	196	93.9	72.0/61.0(85)
	16 K(1)	4.6	3.8	82.5	264	253	95.8	87.6/75.1(86)
	16 K(2)	2.3	1.9	80.3	135	126	93.2	51.7/45.4(88)
Conference	8 K(1)	14.3	12.2	84.9	2,526	2,535	100.3	239.0/206.2(86)
	8 K(2)	8.7	7.3	84.4	1,527	1,524	99.8	151.6/129.7(86)
	16 K(1)	9.7	8.2	84.8	1,703	1,708	100.3	167.2/143.8(86)
	16 K(2)	6.1	5.1	84.5	1,070	1,069	100.0	111.0/95.4(86)
Sibenik	8 K(1)	12.8	11.6	91.2	3,473	3,478	100.1	215.6/196.9(91)
	8 K(2)	5.5	5.0	90.9	1,496	1,494	99.9	101.6/93.8(92)
	16 K(1)	7.2	6.6	91.3	1,958	1,961	100.1	128.2/118.8(93)
	16 K(2)	2.9	2.6	91.0	789	788	100.0	61.0/56.4(92)
Sponza	8 K(1)	12.6	11.1	87.6	1,917	1,919	100.1	212.5/189.1(89)
	8 K(2)	7.5	6.5	87.4	1,133	1,131	99.8	132.9/117.2(88)
	16 K(1)	7.4	6.5	87.6	1,126	1,127	100.0	131.3/117.2(89)
	16 K(2)	4.8	4.2	87.5	731	731	100.0	90.7/81.3(90)
BVH								
Bunny	8 K(1)	12.0	10.5	87.4	4,902	4,916	100.3	347.8/307.7(88)
	8 K(2)	5.8	5.0	86.8	2,341	2,333	99.7	182.0/160.6(88)
	16 K(1)	5.7	4.9	87.1	2,305	2,306	100.0	179.4/158.0(88)
	16 K(2)	1.5	1.3	87.2	608	606	99.7	67.1/61.7(92)
Conference	8 K(1)	37.5	32.4	86.5	31,349	31,560	100.7	1029.4/893.0(87)
	8 K(2)	25.6	21.9	85.8	21,386	21,360	99.9	711.3/612.4(86)
	16 K(1)	17.6	15.3	86.6	14,748	14,863	100.8	497.4/436.0(88)
	16 K(2)	12.4	10.7	85.9	10,406	10,401	100.0	358.4/313.0 (87)
Sibenik	8 K(1)	29.8	27.9	93.6	31,599	31,622	100.1	823.5/772.8(94)
	8 K(2)	20.8	19.4	93.5	22,041	22,036	100.0	583.0/545.6(94)
	16 K(1)	14.5	13.6	93.5	15,401	15,405	100.0	414.6/390.5(94)
	16 K(2)	7.7	7.2	93.5	8,192	8,191	100.0	232.8/219.5(94)
Sponza	8 K(1)	26.5	24.3	91.7	15,717	15,751	100.2	735.3/676.5(92)
	8 K(2)	21.9	20.1	91.4	13,028	13,026	100.0	612.4/564.3(92)
	16 K(1)	13.1	12.0	91.5	7,752	7,761	100.1	377.2/347.8(92)
	16 K(2)	6.5	5.9	91.4	3,851	3,850	100.0	200.7/184.7(92)

DOI: 10.1587/elex.10.20130468
Received June 17, 2013
Accepted June 28, 2013
Publicized July 09, 2013
Copyedited July 25, 2013

hardware [8]. It can be calculated from the equation

$$ACPR_{TRV} = N_{avg} * ((1 - R_{miss}) * L_{hit} + R_{miss} * L_{miss}) \quad (1)$$

where $N_{avg}, L_{hit}, R_{miss}$ and L_{miss} denote average number of node traversal per ray, hit ratio, hit latency, miss ratio and miss penalty respectively. In our experiments, we assign 1 for L_{hit} and 100 for L_{miss}.

Simulation results show that our pre-fetching scheme is useful for both kd-trees and BVHs. The pre-fetching scheme reduces cache miss rates of kd-tree traversal and BVH traversal by 8.7–19.3% and 6.4–14.2%, respectively so that the average cycles for a ray traversal is also reduced. The results also show that our scheme is more suitable for set-associative caches rather than direct-mapped caches. This is because higher associativity reduces the chance that the pre-fetched node data will be evicted. In terms of the memory bandwidth required, the results with and without the pre-fetching scheme are very similar. On average, our proposed scheme decreases the required memory bandwidth for kd-tree traversal by 1.2% and increases that for the BVH traversal by 0.1%. As predicted in the previous section, the memory traffic overhead of the proposed scheme is very low.

ACPR is also decreased by 6–16%, which is actually tens of cycles per ray. This is quite promising for entire ray tracing performance because ten or hundred millions of cycles can be reduced while shooting millions of rays to render a frame. In contrast kd-tree, traversal of BVH structure can not early-terminated, so that ACPR value of BVH traversal is more than that of kd-tree.

5 Conclusion

In this paper, we proposed a simple and effective pre-fetching scheme that loads node address in the node stack into cache blocks prior to traversing the node. The experimental results show that the proposed pre-fetching scheme is effective for both kd-tree and BVH structures. The pre-fetching scheme decreases the cache miss rates by 6.4–19.3% and the average traversal cycles per ray by 6–16%, while the bandwidth overhead is negligible.

In future studies, we intend to exploit more accurate prediction rules before pre-fetching by considering additional information such as node level. We also would like to extend our scheme to BVH and other types of acceleration structures. Furthermore, as software render tries to leverage pre-fetch instruction of commodity CPU by pre-loading data as much as possible, we will extend the hardware pre-fetching unit to other types of data such as primitive list or textures.

DOI: 10.1587/elex.10.20130468
Received June 17, 2013
Accepted June 28, 2013
Publicized July 09, 2013
Copyedited July 25, 2013