IEICE Electronics Express, Vol.9, No.9, 849–854

Efficient ray sorting for the tracing of incoherent rays

Jae-Ho Nah[1a)], **Yun-Hye Jung**[1b)], **Woo-Chan Park**[2c)], **and Tack-Don Han**[1d)]

[1] *Department of Computer Science, Yonsei University*
134 Shinchon-dong, Seodeaman-gu, Seoul 120–749, Korea
[2] *Department of Computer Engineering, Sejong University*
98 Gunja-dong, Gwangjin-gu, Seoul, Korea
a) *jhnah@msl.yonsei.ac.kr*
b) *yhjung@msl.yonsei.ac.kr*
c) *pwchan@sejong.ac.kr*
d) *hantack@msl.yonsei.ac.kr*

Abstract: High-quality rendering via ray tracing requires the tracing of many incoherent secondary rays. In order to accelerate the tracing of incoherent rays, we propose a simple sorting method to increase ray coherence. In this method, we use ray origin buckets and ray direction grids to reorder rays quickly. We implemented our approach on the Manta interactive ray tracer and achieved up to a 1.48× speedup.

Keywords: ray tracing, path tracing, ray sorting, ray reordering
Classification: Electron devices, circuits, and systems

References

[1] I. Wald, P. Slusallek, C. Benthin, and M. Wagner, "Interactive rendering with coherent ray tracing," *Computer Graphics Forum*, vol. 20, no. 3, pp. 153–164, Sept. 2001.
[2] I. Wald, S. Boulos, and P. Shirley, "Ray tracing deformable scenes using dynamic bounding volume hierarchies," *ACM Transactions on Graphics*, vol. 26, no. 1, Jan. 2007.
[3] S. Boulos, I. Wald, and C. Benthin, "Adaptive ray packet reordering," *Proc. 2008 IEEE/EG Symp. Interactive Ray Tracing*, pp. 131–138, Aug. 2008.
[4] J. Bigler, A. Stephens, and S. G. Parker, "Design for parallel interactive ray tracing systems," *Proc. 2006 IEEE/EG Symp. Interactive Ray Tracing*, pp. 187–196, Sept. 2006.
[5] E. Mansson, J. Munkberg, and T. Akenine-Moller, "Deep coherent ray tracing," *Proc. 2007 IEEE/EG Symp. Interactive Ray Tracing*, pp. 79–85, Sept. 2007.
[6] B. Moon, Y. Byun, T.-J. Kim, P. Claudio, H. Kim, Y. Ban, S. W. Nam, and S.-E. Yoon, "Cache-oblivious ray reordering," *ACM Transactions on Graphics*, vol. 29, no. 3, pp. 28:1–28:10, July 2010.
[7] R. Overbeck, R. Ramamoorthi, and W. R. Mark, "Large ray packets for real-time Whitted ray tracing," *Proc. 2008 IEEE/EG Symp. Interactive Ray Tracing*, pp. 41–48, Aug. 2008.

DOI: 10.1587/elex.9.849
Received March 05, 2012
Accepted April 16, 2012
Published May 14, 2012

1 Introduction

Ray tracing has been widely used for high-quality rendering. Ray tracing algorithms trace the path of light to simulate global illumination effects such as reflection, refraction, and shadows. However, while doing this, ray tracing incurs considerable computational cost because dozens of traversal and intersection operations per ray are performed to find the hit point.

In recent years, many studies of real-time ray tracing have tried to accelerate ray tracing by means of packet tracing. Packet tracing [1] reorganizes each of the rays into packets of rays in order to amortize the cost across sets of rays and to exploit the CPU's SIMD (single instruction, multiple data) units. This algorithm was extended to a packet-frustum traversal scheme on dynamic bounding volume hierarchies [2]. This scheme achieved an order-of-magnitude speed-up in terms of primary visibility because it substitutes per-ray-packet operations for expensive per-ray operations. However, the global illumination effects caused by ray tracing necessitate many incoherent secondary rays. Because packet-based approaches exploit ray coherence, the gains made by packet tracing dwindle when incoherent rays are traced [3].

In this paper, we introduce an acceleration technique for incoherent rays. Our method sorts the rays using their origins and directions with linear time complexity. We use bucket sorting to achieve this time complexity. Completion of this sorting yields rays that are now more coherent. Existing coherent ray tracing algorithms, such as packet tracing [1, 2], can now be more effective, even in the case of incoherent rays. This is possible because higher ray coherence increases both the SIMD efficiency [1, 2] and the odds of passing the early hit/miss test in [2]. Experimental results show that our method increases the performance of path tracing with a packet-based ray tracer [4].

2 Related work

Considerable effort has been directed toward accelerating incoherent ray tracing. In order to exploit ray coherence, a few studies aimed to reorder rays before ray traversal. Masson et al. [5] grouped coherent rays together to increase coherence, but they failed to increase the overall rendering performance due to the high cost of sorting. Cache-oblivious ray reordering [6] is a ray reordering technique using a hit point heuristic, which uses an approximate hit point between the rays and a simplified representation of the original models. This approach effectively reduces I/O thrashing during out-of-core ray tracing. However, the advantages of this approach in packet tracing have not been presented.

On-the-fly ray reordering and novel traversal methods have also been studied. Partition traversal [7] filters active SIMD rays for each traversal step. This method is more robust with incoherent rays and large packets compared to the masked traversal [1] and ranged traversal [2] methods. Adaptive ray packet reordering [3] reorders rays when the packet utilization rate drops below a certain threshold. As described in [6], however, these on-the-fly approaches complicate the traversal stage and have low modularity.

DOI: 10.1587/elex.9.849
Received March 05, 2012
Accepted April 16, 2012
Published May 14, 2012

3 Proposed algorithm

The main goal of our approach is to increase ray coherence for packet tracing before traversal. For higher ray coherence, we reorder rays in a ray packet using their origin and direction. Fig. 1 depicts this process. First, we generate a large ray packet (e.g., 1024 rays per packet). Large ray packets are selected because the ray packets generated after ray reordering are much smaller than the original ray packet. Next, we classify rays using 24 ray direction buckets. The criteria of this classification are the octants and major axes of each ray's direction. Afterwards, we construct a local origin grid and classify rays onto the grid. The criterion of this classification is the position of each ray's origin. Finally, we trace a sub-packet of coherent rays through an acceleration structure such as a bounding volume hierarchy (BVH).

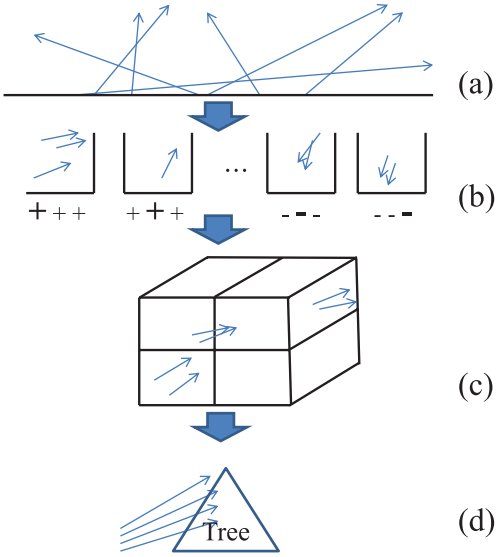

Fig. 1. The process of the proposed method: (a) incoherent rays, (b) ray classification sorting using 24 ray buckets, (c) ray classification using a local origin grid, and (d) traversal of the coherent ray packet

The details of our approach are described below. The first stage is ray classification using the rays' direction in the ray packet. Because there are eight octants and three major axes, we can assign 24 buckets. When we classify rays, we do not copy all of the data of the rays because doing so would be very expensive relative to the structure of arrays (SoA) for packet tracing. In other words, if we try to access all of the data for a single ray, a non-sequential data access method would be necessary. Hardware scatter/gather support can solve this problem, but current SIMD operations (e.g., Intel SSE) do not support scatter/gather routines. Therefore, we only store the rays' index in direction buckets and postpone the execution of ray reordering. Also, we read the rays' origin values and calculate their min/max values in each direction bucket. These min/max values represent the axis-aligned bounding boxes (AABBs) of the rays in each direction bucket.

With the completion of the direction-based ray classification procedure, we visit each ray direction bucket and construct a local origin grid. The size

DOI: 10.1587/elex.9.849
Received March 05, 2012
Accepted April 16, 2012
Published May 14, 2012

of an origin grid can range from one to eight. In order to set this size, we compare the root node's AABB and the bucket's AABB on three axes. In order to prevent grid cells that are too small, we divide the grid cell for each axis only if the axis range of the bucket's AABB is greater than a quarter of that of the root node's AABB. If all three axes are divided, the size of the origin grid is eight. If none of the axes are divided, the size of the origin grid is one. After the grid size is set, we put ray indices onto the proper grid cells.

We then make a coherent sub-packet in a SIMD-friendly manner for low sorting overhead and efficient packet traversal. First, we exploit SIMD store instructions when we copy the origin and direction from the original ray packet data to the new reordered ray packet data. This process is performed using the indices in the grid cells. When we read the original ray, we initially put this ray into a temporary ray packet until four (= SIMD width) rays are stored. Temporary ray packets include only the rays' origins and direction member variables. The sign of the direction vector, the inverse direction vector, and the initial t distance are computed on the fly. T distance means distance to the nearest object pierced by the ray. The initial t distance is set to the maximum number representable in type float. When four rays are stored or when the final ray in a ray packet is fetched, we copy this data by means of SIMD operations. Because this process requires a sequential memory access, the sorting overhead of a SOA form can be decreased. Second, we maintain 16-byte alignment for SIMD traversal when we create a sub-packet. This alignment can increase SIMD efficiency during the traversal step.

Upon the completion of ray reordering, we perform a ray traversal procedure with this new sub-packet. Because the proposed method is performed during the ray generation stage, there are no restrictions on the types of acceleration structures and traversal algorithms. After traversal, we copy the hit results from the new ray packet data to the original ray packet data.

Our ray reordering method has no disadvantages in terms of the quality of the output images. Because a ray packet consists of rays which are created at the same depth, the rays do not have dependencies between them. Therefore, our reordering method does not change the shading result of the ray packet.

4 Experimental results

In order to evaluate our approach, we used the Manta [4] interactive ray tracer. For incoherent ray tracing, we set the renderer to the path tracer. Because diffuse inter-reflection rays are created on the reflective surfaces, we used Lambertian surfaces for all benchmarks. We made 64 samples per pixel by means of jittered sampling. The ray recursion depth was two. We did not shoot shadow rays for a simple experiment. Rays were traversed using both ranged [2] and partition [7] traversals. For these experiments, an Intel Core i7 980X (3.3 GHz six-core processor) with 6 GB of memory was used.

We chose the Conference (282 K triangles), Fairy (174 K triangles), and Sibenik (80 K triangles) scenes for experiments. All images were rendered at a resolution of 1024×1024 (Fig. 2). For each scene, we compared the traversal

DOI: 10.1587/elex.9.849
Received March 05, 2012
Accepted April 16, 2012
Published May 14, 2012

results with and without the proposed ray sorting method. We compared the traversal results for ranged traversal without ray sorting, partition traversal without ray sorting, and ranged traversal with ray sorting. We did not include the results of partition traversal with ray sorting because the performance advantages from ray sorting were lower than the sorting overhead in this case. In other words, partition traversal did not efficiently use the advantage of increased coherence. With our sorting method disabled, we used 8×8 (the default in Manta) and 32×32 ray packets for ranged traversal and partition traversal, respectively. With the proposed sorting method enabled, we used 32×32 ray packets to obtain sufficient input rays for sorting.

Fig. 2 and Table I depict the experimental results. The graph in Fig. 2 shows the performance of each case. This graph illustrates measurements of the total rendering time for one picture. Per-frame statistics are also shown in Table I. According to the results, our approach was 1.18–1.46 times faster than the original BVH packet traversal (ranged traversal) method without ray sorting. As described in [3], a "speculative" first hit and a "conservative" interval arithmetic test in ranged traversal would be ineffectual when tracing incoherent rays. This was avoided by applying our reordering scheme such that the number of interval arithmetic (IA) tests and ray-triangle intersection tests decreased.

In addition, our approach achieved higher rendering performance than partition traversal by an average of 15 percent. Partition traversal always uses SIMD ray packets, but ranged traversal adaptively selects either scalar processing or SIMD processing according to SIMD efficiency. Of course, scalar processing for a single ray is simpler than SIMD processing for four SIMD rays. Due to this feature, our approach is faster than partition traversal [7], although the total numbers of N_B and N_T for partition traversal [7] and that of ranged traversal with sorting (our method) in Table I are similar.

The performance improvement of our approach is proportional to the number of ray-triangle intersection tests for each scene. This is because

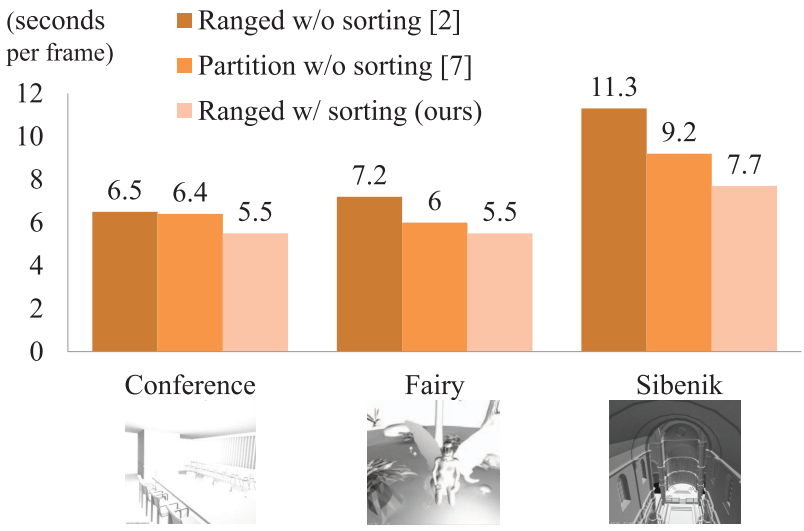

Fig. 2. Benchmark results (1): Seconds per frame (a lower value is better)

DOI: 10.1587/elex.9.849
Received March 05, 2012
Accepted April 16, 2012
Published May 14, 2012

Table I. Benchmark results (2): Average per-frame statistics. Notations: N_{IA}, N_B, and N_T are the number of interval arithmetic tests per ray, ray-box intersection tests per ray, and ray-triangle intersection tests per ray, respectively. A lower value is better.

Scene	Item	Ranged w/o sorting [2]	Partition w/o sorting [7]	Ranged w/ sorting (ours)
Conference	N_{IA}	3.53	2.88	2.04
	Scalar N_B	3.50	0.00	2.98
	SIMD N_B	3.57	6.93	4.24
	Scalar N_T	13.99	0.00	12.82
	SIMD N_T	28.55	22.35	10.68
Fairy	N_{IA}	5.09	4.84	2.39
	Scalar N_B	3.77	0.00	3.07
	SIMD N_B	3.00	7.08	4.79
	Scalar N_T	19.74	0.00	14.85
	SIMD N_T	35.35	30.01	16.58
Sibenik	N_{IA}	7.72	5.96	3.63
	Scalar N_B	3.48	0.00	2.69
	SIMD N_B	2.22	6.95	3.60
	Scalar N_T	30.28	0.00	23.60
	SIMD N_T	60.75	38.04	20.22

the main effect of our approach is a reduction in unnecessary ray-triangle intersection tests. As a result, our approach showed the highest performance improvement in the Sibenik Scene.

5 Conclusions and future work

In this study, we presented a simple sorting method to increase ray coherence for packet tracing. Because the performance advantage of packet tracing [2] depends on ray coherence, the weak point of packet tracing is the associated degradation in the performance of the incoherent rays [3]. Our approach compensates for the weak points of the packet tracing of incoherent rays. According to experimental results in Manta, the use of our ray sorting method led to an increase in speed by 1.18–1.46 times compared to the ranged traversal method in dynamic bounding volume hierarchies [2].

In future studies, we would like to extend our approach to GPU-based ray tracers. We hold that bucket sorting can be performed quickly on current GPUs. In addition, we would like to apply our approach to other acceleration structures and traversal algorithms, such as kd-tree packet traversal [1].

Acknowledgments

We would like to thank Xin Sun and Xin Tong for fruitful discussions and feedback. This research was supported by the MKE (The Ministry of Knowledge Economy), Korea and Microsoft Research, under IT/SW Creative research program supervised by the NIPA (National IT Industry Promotion Agency). The Conference and Sibenik scenes are courtesy of Greg Ward. The Fairy scene is courtesy of Utah 3D animation repository.

DOI: 10.1587/elex.9.849
Received March 05, 2012
Accepted April 16, 2012
Published May 14, 2012