Multimed Tools Appl
DOI 10.1007/s11042-015-2534-4

Geometry transition method to improve ray-tracing precision

Dongseok Kim · Jae-Ho Nah · Woo-Chan Park

Received: 30 September 2014 / Revised: 12 February 2015 / Accepted: 24 February 2015

Abstract We propose a method for moving the view position to the origin and moving the coordinates of primitives so that they are at the same distance in order to improve ray-tracing precision. This approach exploits the principle that a floating-point number provides higher precision near zero. In this way, we can significantly reduce the number of self-intersections occurring in ray tracing that are caused by limited floating-point precision. The experimental results show that the number of self-intersections is reduced by up to 84.6 %. We also propose a hardware approach to resolve the computational overhead in the proposed algorithm. Its contribution to the hardware size is very small in comparison with the size of the entire ray-tracing hardware.

Keywords 3D graphics · Ray tracing · Rendering artifact · Floating-point arithmetic

1 Introduction

Ray tracing is a global illumination algorithm that is able to create very realistic images [15]. Because of the vast amount of calculation needed compared to local illumination models, global illumination based on ray tracing has not yet been used in real-time graphics processing but has been widely used for high-quality offline rendering. With advances in semiconductor technology, recent research has focused on developing real-time ray-tracing algorithms [10].

D. Kim · W.-C. Park (✉)
Department of Computer Engineering, Sejong University, Seoul, Republic of Korea
e-mail: pwchan@sejong.ac.kr

D. Kim
e-mail: dskim@rayman.sejong.ac.kr

J.-H. Nah
LG electronics, Seoul, Republic of Korea
e-mail: nahjaeho@gmail.com

Published online: 06 March 2015

A dedicated hardware architecture for mobile ray tracing, called RayCore, has recently been issued [10]. The RayCore architecture is based on a multiple instruction, multiple data (MIMD)-based ray-tracing architecture [11] and has been developed as a hardware architecture that can be integrated into mobile application processors (APs). Mobile APs have many limitations compared to desktop processors, including a smaller die area, lower power resources, and lower memory bandwidth. For these limitations, a 24-bit floating-point format was chosen for RayCore [10] instead of the 32-bit format, but this 24-bit precision can result in rendering more artifacts than the 32-bit precision.

The core operation of a ray-tracing system is to find the first intersection between a ray and a primitive. Due to the limitations of the floating-point format, intersection algorithms have intrinsic numerical problems that can result in rendering artifacts. To resolve these problems, it is important to choose a good epsilon [1]. Usually, a small epsilon value [13], called RAY_EPSILON [12], is used to check if t is within some tolerance [3]. However, finding a perfect epsilon is difficult. For example, an epsilon that is too small may result in a self-intersection problem in which the hit point of a secondary ray is located in the same object in which it originated. In contrast, an epsilon that is too large may result in overlaps from the extended geometry. Both examples can make incorrect images, such as holes.

In an attempt to avoid the self-intersection problem, several related studies have been carried out using a fixed-point format [5] or an integer format [6], object-space intersection computation [1], and by adding a small amount of padding to bounding boxes to avoid holes [8, 9]. Hanika [5] and Heinly et al.'s [6] methods are very effective for fixed-point and integer ray tracers, respectively, but neither method can be applied to floating-point formats that have been widely used in most graphic applications. Dammertz and Kellers method [1] offers higher numerical precision by avoiding some of the quantization and the avoidance of self-intersections of secondary rays. However, this method has some potential problems with the subdivision accuracy at irregular vertices and the performance degradation when subdividing surfaces on the fly. Ize [8] proposed a numerically robust bounding volume hierarchy (BVH) traversal algorithm, and Keely [9] proposed a BVH-based hardware architecture with reduced precision arithmetic. However, these approaches cannot be used with spatial subdivision structures, such as kd-trees and grids.

In contrast to the previous work, the aim of this paper is to improve the floating-point precision in ray tracing without modifying the acceleration data structures, traversal algorithms, or data formats. The proposed algorithm consists of two simple steps. First, we move the camera position to the origin of the coordinate. After that, the absolute value of the coordinate is changed to a smaller absolute value than that of the original coordinate. This method exploits a principle of floating-point arithmetic that a floating-point number with a small absolute value has higher precision than a floating-point number with a large absolute value. For this reason, we can reduce the inaccuracies in the intersection test results and self-intersection problems. This concept is known as an origin offset; however, to the best of our knowledge, our approach is the first application of an origin offset to improve ray-tracing precision.

Additionally, we implement a hardware unit in our approach. This hardware implementation can significantly reduce the overhead of software implementation. To evaluate the effect of our origin-offset approach in ray-tracing pipelines, we also integrate the hardware unit with the RayCore system [10]. Thanks to the simplicity of our approach, we think it is applicable not only to RayCore but also to other ray tracers. Additionally, our approach can be used with other approaches, such as epsilon-based approaches [3, 12, 13] or padding-based approaches [8, 9], for higher image quality.

We tested the proposed algorithm on the following benchmarks: the Game Institute's [2] Spatial Partitioning II, Sponza, and Sibenik. The results showed that the rendering artifacts due to low precision were reduced by 65.4 %, 84.6 %, and 24.8 %, respectively, on each benchmark.

This paper is organized as follows: We first describe the background and related work in Section 2. We next describe the proposed algorithm and its hardware architecture in Section 3 and the experimental results in Section 4. Finally, we describe the conclusions and limitations in Section 5.

2 Background and related work

2.1 Rendering artifacts

The typical number that can be represented exactly is of the form (significant digits)×(base)$^{(exponent)}$. The IEEE 754 standard [7] defines single-precision normalized values as follows:

$$f = (-1)^s \cdot 1.m_{22}m_{21}\cdots m_0 \cdot 2^{e-127} \tag{1}$$

where the 32 bits can be interpreted as one sign bit, an 8-bit exponent, and a 23-bit mantissa. A real number cannot be represented precisely in the floating-point format, and arithmetic operations cannot be represented precisely in floating-point arithmetic [4]. To reduce this error of the floating-point format, we can generally use 64-bit double precision, improve computational algorithms, or use origin offsets.

Due to the floating-point precision, a primitive that should be missed by a ray can be hit and vice versa. These types of errors make rendering artifacts, such as dark spots, on the rendered image. There are two specific cases: a) floating-point calculation errors in intersection tests between the ray and primitives and b) a self-intersection problem of secondary rays. The probability of these problems occurring increases as the absolute value of a floating-point number increases. This is because the resolution of floating-point numbers becomes coarser as the distance from the origin increases [4]. Our goal is to reduce the probability of rendering artifacts using origin offsets during ray tracing.

2.2 Fixed-point ray tracing

Fixed-point ray tracing [5] is a simple way to maintain high computational accuracy and to avoid the self-intersection problem. Figure 1 shows how the self-intersection problem for fixed-point ray tracing can be avoided. This method is as follows: Distance t measured along ray direction w uses the same resolution as the vertex data. Therefore, it can only be wrong by one unit of the integer grid. The calculation results might have a rounding error as high as 0.5. The worst-case error along each axis is thus $1 + 0.5 < 2$, and consequently the ray origin needs to be shifted by only two units. This is effective for fixed-point ray tracing; however, it has limited use for the floating-point ray tracing used by most graphic applications.

2.3 Object space intersection computation

Dammertz and Keller [1] proposed a method of determining the point of intersection in object space instead of computing an intersection as the distance along the ray. They directly

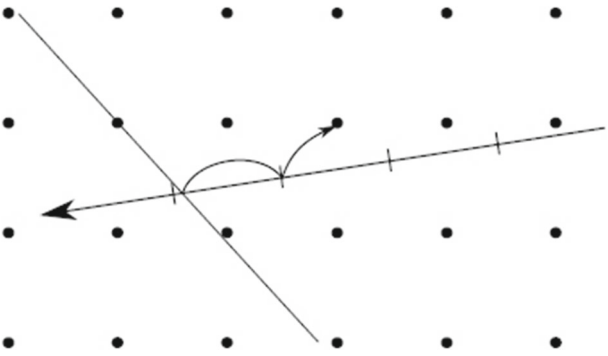


Fig. 1 The self-intersection problem for fixed-point ray tracing [5]

computed a three-dimensional interval in object space that contains the true point of intersection; it can also be interpreted as an axis-aligned bounding box. It is found by hierarchical subdivision (see also Fig. 2).

Dammertz and Keller used the subdivision algorithm that is subdivided until the subdivision no longer changes in the floating-point representation. If the neighboring bounding boxes resulting from the subdivision overlap or at least touch in the floating-point representation, no cracks can occur. In addition, they used a simple way to avoid the self-intersection problem in which the starting point of the secondary ray is selected as the corner point of the farthest intersection interval in the direction of the normal. However, there is a performance problem when subdividing subdivision surfaces and calculating the axis-aligned bounding box. For example, the render time increased about 2.5 times from 1.318 sec to 3.283 sec for the Dog scene [1].

3 Proposed algorithm and architecture

3.1 Viewport transfer algorithm

As mentioned in Section 2.1, rendering artifacts can exist in floating-point ray tracers due to precision problems. The probability of these rendering artifacts is proportional to an absolute value of coordinates; thus, when a camera and an object are located far from zero, rendering artifacts are more likely. This is because the addition and subtraction of floating-point arithmetic are computed on the exponent of the larger number.

One goal of the proposed algorithm is to reduce these rendering artifacts caused by floating-point precision. The key point of the proposed algorithm is the movement of the camera position to the origin of coordinates and of objects and light sources to the origin

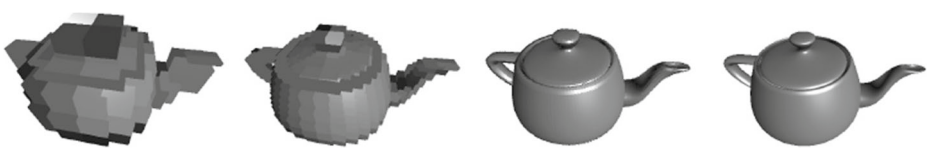


Fig. 2 The hierarchical refinement of the bounding boxes [1]

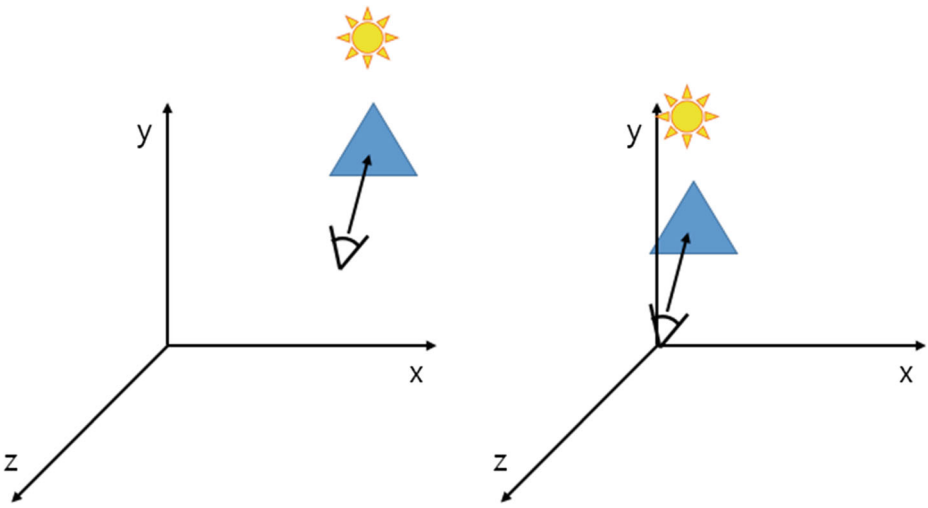


Fig. 3 An overview of the proposed algorithm (left, original; right, transferred)

by the same amount. Figure 3 shows our idea of viewport transfer to decrease the absolute values of coordinates.

The proposed algorithm operates as follows: First, the camera position used to generate primary rays is changed to (0, 0, 0), as shown Fig. 4. Second, we subtract the camera position from the coordinates of objects and light sources. As a result, the origin of the primary rays is zero and the coordinates of objects and light sources have smaller absolute values than the original values.

We describe a specific example to compare the original coordinates and the transferred coordinates with our approach in Table 1. In this table, we include the camera position and data of a particular primitive (id: 0x35c6) that generates a rendering artifact in the Game Institutes Spatial Partitioning II scene. After applying the proposed algorithm, the camera position is changed to (0, 0, 0) and each coordinate value of the moved triangle has smaller absolute values than the original values. For a particular primitive in Table 1, the average absolute values of the transferred coordinates with the proposed algorithm are 80.1 % less than those of the original coordinates.

These numbers with smaller absolute values indicate the increased accuracy of intersection tests. To be more concrete, we can ensure high precision when we compute the t (the distance from the ray origin to the hit point) value in a ray-triangle intersection test and run barycentric coordinate tests in an intersection test [14].

As a result, our approach has two advantages. First, the number of cases in which primitives that should be missed by a ray are hit and vice versa decreases, so the accuracy of the

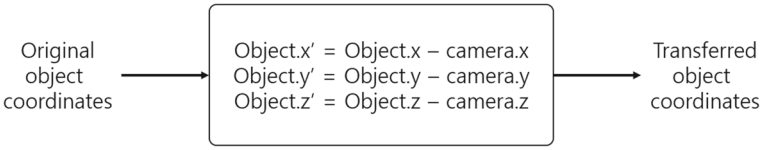


Fig. 4 Proposed algorithm's object coordinate transfer

Table 1 An example result of our approach in the Spatial Partitioning II scene

		Original coordinates	Transfeered coordinates	Reduction rate of the absolute value(%)
Camera	x	780.937500	0	–
	y	141.634766	0	–
	z	73.593750	0	–
Vertex 0	x	941.414063	160.476563	83.0
	y	196.027344	54.392578	72.2
	z	113.305664	39.711914	65.0
Vertex 1	x	950.984375	170.046875	82.1
	y	146.320313	4.685547	96.8
	z	107.782227	34.188477	68.3
Vertex 2	x	949.000000	168.062500	82.3
	y	139.466797	−2.167969	98.4
	z	108.924805	35.331055	67.6
Barycentric coordinate	nd	655.835938	132.193359	79.8
	bd	−333.277344	−57.7812500	82.7
	cd	296.359375	51.7421875	82.5

intersection tests is increased. Second, the probability of the self-intersection problem also decreases.

3.2 Hardware architecture

As described above, the proposed algorithm performs the viewport transfer at each primitive. Thus, its computational overheads can cause performance degradation. According to our experiments, the performance degradation of the proposed algorithm averages 18 % on our 24-bit software ray tracer. To resolve this problem, we implement a hardware viewport transfer unit (VTU) and integrate the VTU with RayCore [10]. The existence of the VTU has been noted in [10] as a geometric transition. A VTU does not significantly affect the entire hardware size because it is very small; a VTU only requires a few floating-point adders and multipliers and does not require any on-chip memory or complex control logics.

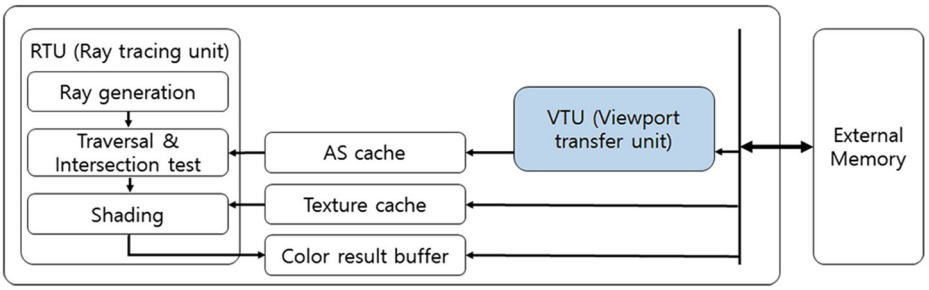


Fig. 5 Proposed hardware architecture

Fig. 6 Benchmarks: Spatial partitioning II(left), Sponza(*center*), Sibenik(*right*)

Figure 5 shows the proposed hardware architecture. The data from the external memory through a bus are converted by a VTU. The converted data are then stored to the acceleration structure (AS) cache. The VTU is fully pipelined for high performance and does not require off-chip memory accesses. As a result, the insertion of the VTU into RayCore does not cause performance degradation.

4 Experimental results

4.1 Experimental environment and benchmark

To measure the effect of the proposed method, we used a 24-bit software ray tracer on a desktop with Intel i5-2500 3.3 GHz CPU and 8 GB RAM. The screen resolution of rendering images was 800x480. We used a single-point light source. As a result, 384,000 primary rays and 384,000 shadow rays were generated per frame.

For the performance evaluation, we selected three scenes: the Game Institutes lab project 16.1 Spatial Partitioning II, Sponza, and Sibenik (see Fig. 6). The first benchmark is comprised of several rooms and it is connected by a long passage between each room. This benchmark serves as the momentum to start this study because this scene showed many rendering artifacts with the 24-bit ray tracer. The Sponza and Sibenik scenes have been widely used in ray tracing, and the original geometry data of each scene are close to zero.

Table 2 Benchmark details

			x	y	z
Spatial Partitioning II	Camera position		780.937500	141.634766	73.5937500
	Bounding box	Min	−885.726563	−278.933594	−886.484375
		Max	1241.906250	278.933594	1003.179688
Sponza	Camera position		−1187.687500	700.382813	149.572266
	Bounding box	Min	−1743.747559	−92, 323967	−781.708435
		Max	1745.200684	1566.984131	781.403748
Sibenik	Camera position		−1633.300780	643.249512	−666.711365
	Bounding box	Min	−2777.996830	−3.05996895	−851.363403
		Max	1258.215700	3063.028810	851.363403

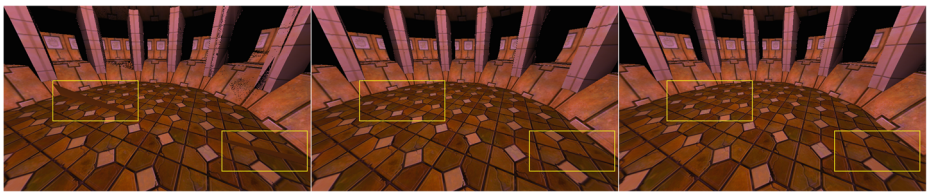


Fig. 7 Spatial partitioning II scene(left, original 24bit; center, proposed 24bit; right, original 32bit)

These scenes have been modified to make them 100 times larger so that they have large coordinates, like the Green Bunny scene in [5].

Table 2 shows each benchmarks details. Bounding boxes show the minimum and maximum coordinates of objects in each scene, and the maximum absolute values of the scenes can be obtained from the bounding boxes. For the Spatial Partitioning II scene, we chose a camera position that shows objects that should be missed by rays. For the other two scenes, the camera positions were selected in the outskirts and have large absolute values.

4.2 Intersection test error

Figure 7 shows one particular room in the first benchmark. Two primitives (id: 0x7aab [left], 0x7b13 [right]) that should be missed are shown in the yellow rectangles of the left image. A ray hit primitive (id: 0x7646 [left], 0x7b22 [right]) that should be hit, and therefore two primitives (id: 0x7aab [left], 0x7b13 [right]) that should be missed, are not shown in the viewport transferred image (center) and the 32-bit floating-point format image (right).

4.3 Self-intersection problem

We compared our approach to the original in terms of the number and the probability of the self-intersection problem (see also Table 3). According to the results, our approach reduced the probability of the self-intersection problem in both 24-bit and 32-bit floating-point formats. The number of self-intersection problems was also reduced up to 84.6 % and 99.1 % on the 24-bit and 32-bit floating point formats, respectively.

Table 3 The number and the reduction of self-intersection problems

	24-bit floating-point format					32-bit floating-point format				
	Original		Proposed		Reduction	Original		Proposed		Reduction
	Count	%	Count	%	%	Count	%	Count	%	%
Spatial partitioning	13,091	3.4	4,524	1.2	65.4	212	0.1	2	0.0	99.1
Sponza	3,172	0.8	487	0.1	84.6	0	0.0	0	0.0	–
Sibenik	22,310	5.8	16,767	4.4	24.8	208	0.1	117	0.1	43.7

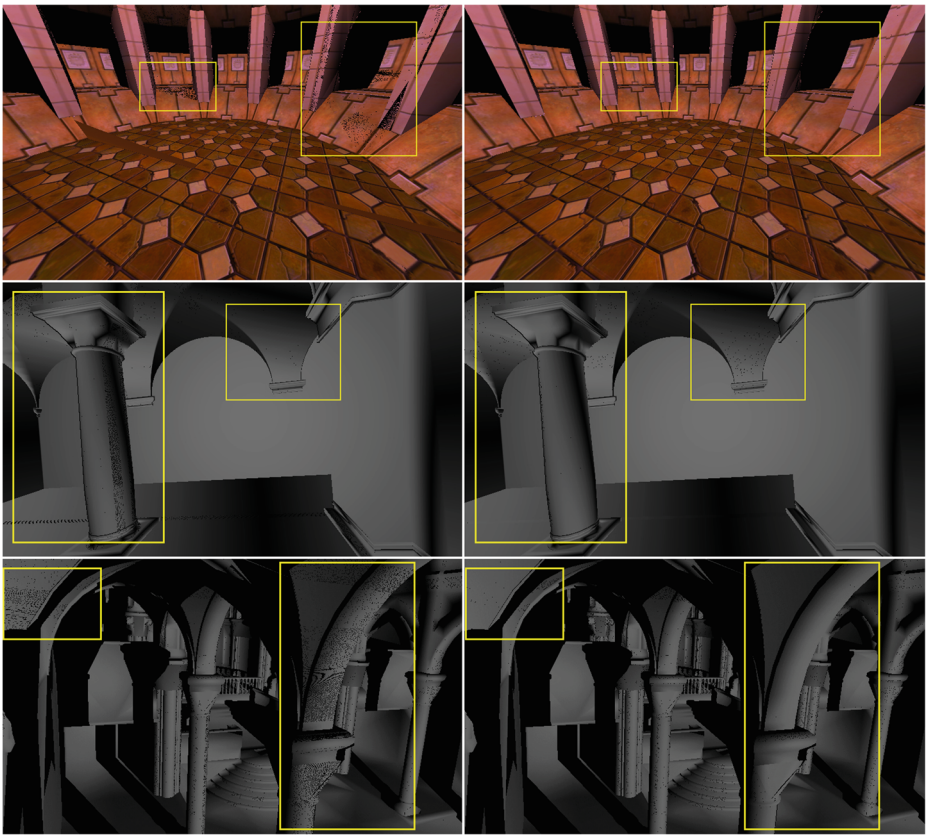


Fig. 8 (*left*) Original images, (*right*) rendered images with our approach

Figure 8 shows the comparison of the three cases in our benchmarks. The left images of each figure are the original images using the 24-bit floating-point format, and the right images are the improved images by the proposed algorithm. It is easy to find many rendering artifacts in the yellow rectangles in the original images. In contrast, our approach significantly reduces the rendering artifacts at the same pixels.

5 Conclusion and future work

In this paper, we have proposed a method to improve image quality by reducing rendered artifacts. Our method involves moving the camera position to the origin and the coordinates of primitives the same amount. As a result, the absolute value of the computational number becomes less, so the floating-point precision is improved in ray tracing and rendered artifacts are reduced an average of 58.3 %.

The computational overhead required for the proposed algorithm can cause performance degradation. The proposed algorithm is implemented using hardware and integrated with RayCore [10]. This additional hardware unit is much smaller than the whole RayCore

system, so it does not significantly affect either the performance or the hardware size of the ray-tracing system.

The proposed algorithm has two limitations. First, if a camera position is close to zero, the proposed algorithm has no advantage because of the following reason. The absolute value of the coordinates of the translated object will still be large because the reduced amount of the absolute value of the coordinates will be very small. Second, if the position of an object is close to zero and the camera position is far from zero, the proposed algorithm will not be very advantageous because of the following reason. The absolute value of the coordinates of the translated object will still be large because a number with a small absolute value is changed to a number with a large absolute value. In these cases, one of the two numbers has a large absolute value. For this reason, an intersection test error or a self-intersection problem can occur.

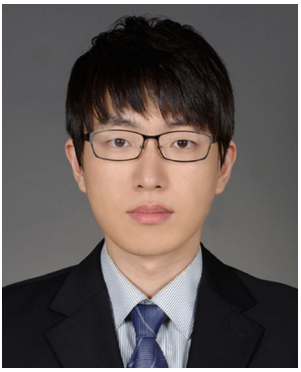
In the future, we will address these limitations by improving the proposed algorithm. To accomplish this, we will reduce the precision loss in the addition or subtraction in the floating-point arithmetic and use variable epsilon values to ensure fewer losses. In addition, we believe that the combination of our approach and a method using a small epsilon [13] called RAY_EPSILON [12], robust BVH algorithms [8], or BVH hardware architectures [9] can result in more accurate ray-traced images.

Acknowledgments This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2012R1A1A2004624).

References

1. Dammertz H, Keller A (2006) Improving ray tracing precision by object space intersection computation. In: Proceedings of the 2006 IEEE symposium on interactive ray tracing, pp 25–32
2. Game institute. http://www.gameinstitute.com/game-development/. Accessed 1 April 2014
3. Glassner A (ed) (1989) An introduction to ray tracing. Academic Press Ltd
4. Goldberg D (1991) What every computer scientist should know about floating-point arithmetic. ACM Comput Surv (CSUR) 23:5–48
5. Hanika J (2007) Fixed point hardware ray tracing. dissertation, Ulm University
6. Heinly J, Recher S, Bensema K, Porch J, Gribble C (2009) Integer ray tracing. Journal of Graphics, GPU, and Game Tools 14:31–56
7. IEEE standard for binary floating-point arithmetic for microprocessor systmes ANSI/IEEE Std. 754 (1985)
8. Ize T (2013) Robust BVH ray traversal. Journal of Computer Graphics and Techniques 2(2):12–27
9. Keely S (2014) Reduced precision hardware for ray tracing. High Performance Graphics, pp 29–40
10. Nah JH, Kwon HJ, Kim DS, Jeong CH, Park J, Han TD, Manocha D, Park WC (2014) RayCore: A ray-tracing hardware architecture for mobile devices. ACM Trans Graph 33(5):162
11. Park WC, Nah JH, Park JS, Lee KH, Kim DS, Kim SD, Park JH, Kim CG, Kang YS, Yang SB, Han TD (2008) An FPGA implementation of whitted-style ray tracing accelerator. IEEE Symposium on Interactive Ray Tracing:187–187
12. Pharr M, Humphreys G (2004) Physically based rendering from theory to implementation. Morgan Kaufmann Publishers
13. Suffern K (2007) Ray tracing from the ground up. A K Peters
14. Wald I (2004) Realtime ray tracing and interactive global illumination. Dissertation, Sarrland University
15. Whitted T (1980) An improved illumination model for shaded display. Commun ACM 23(6):343–349

Dongseok Kim received the B.S. and M.S. degrees from the Department of Computer Engineering, Sejong University in 2006 and 2008, respectively. Currently, he is a Ph.D. student at Department of Computer Engineering, Sejong University. His research interests include ray tracing, rendering algorithms, advanced shading model, acceleration structures, and graphics hardware.



Jae-Ho Nah received the B.S., M.S., and Ph.D. degrees from the Department of Computer Science, Yonsei University in 2005, 2007, and 2012, respectively. Currently, he is a senior research engineer at LG Electronics. His research interests include ray tracing, rendering algorithms, and graphics hardware.

Woo-Chan Park was born on 1 May, 1970, in Korea. He received the BS, MS, and PhD degree in computer science from Yonsei University, Seoul, Korea, in 1993, 1995, and 2000 respectively. He is currently associate professor at Sejong University. His research interests include 3D rendering processor architecture, ray tracing accelerator, parallel rendering, high performance computer architecture, computer arithmetic, and ASIC design.