IEIE Transactions on Smart Processing and Computing, vol. 4, no. 2, April 2015
http://dx.doi.org/10.5573/IEIESPC.2015.4.2.065

Real-time Ray-tracing Chip Architecture

Hyung-Min Yoon[1], Byoung-Ok Lee[1], Cheol-Ho Cheong[1], Jin-Suk Hur[1], Sang-Gon Kim[1], Woo-Nam Chung[1], Yong-Ho Lee[1], and Woo-Chan Park[2,*]

[1] Siliconarts, Inc., Seoul, South Korea {yoonhm, reeb, chjeong, gjwlstjr, sgkim, woonam, jason0720}@siliconarts.com
[2] College of Electronics and Information Engineering, Sejong University / Seoul, South Korea pwchan@sejong.ac.kr

***** Corresponding Author: Woo-Chan Park

Received November 20, 2014; Revised December 27, 2014; Accepted February 12, 2015; Published April 30, 2015

** Short Paper*

** Extended from a Conference: Preliminary results of this paper were presented at the IEEE Hot-Chips 2014 symposium. The present paper has been accepted by the editorial board through the regular review process that confirms the original contribution.*

***Abstract*:** In this paper, we describe the world's first real-time ray-tracing chip architecture. Ray-tracing technology generates high-quality 3D graphics images better than current rasterization technology by providing four essential light effects: shadow, reflection, refraction and transmission. The real-time ray-tracing chip named RayChip includes a real-time ray-tracing graphics processing unit and an accelerating tree-building unit. An ARM Ltd. central processing unit (CPU) and other peripherals are also included to support all processes of 3D graphics applications. Using the accelerating tree-building unit named RayTree to minimize the CPU load, the chip uses a low-end CPU and decreases both silicon area and power consumption. The evaluation results with RayChip show appropriate performance to support real-time ray tracing in high-definition (HD) resolution, while the rendered images are scaled to full HD resolution. The chip also integrates the Linux operating system and the familiar OpenGL for Embedded Systems application programming interface for easy application development.

***Keywords*:** Ray-tracing hardware, GPU, 3D graphics, KD-tree, Global illumination

1. Introduction

In the mid-1990s, a graphics processing unit (GPU) was integrated into computer systems for 3D gaming. The GPU was developed as hard-wired logic in order to render large amounts of 3D graphics data at real-time speeds. From the mid-1990s to the early 2000s, GPUs adapted rasterization algorithms, which present the surface color of objects simply. Those GPUs had critical limitations, such as no light effects. In the early 2000s, in order to overcome the limitations and to support simple light effects, programmable hardware called a shader was integrated into the GPUs. GPUs with a shader are currently used in smart phones, smart TVs, PCs and servers. The shader has two functions. One is change of vertex color, and the other is change of pixel color according to the results of shader programs. Despite integrating the programmable hardware, implementation of four essential light effects (shadow, reflection, refraction and transmission) is not easy. In current GPUs, shader programs use local illumination algorithms for light effects. For example, the shadows are developed with independent light coordination, and sometimes different shadow algorithms are individually applied for each object. In addition, shader performance in current GPUs is insufficient to implement real-time ray tracing.

In order to achieve real-time and realistic light effects, the support of a global illumination algorithm is required. There are many global illumination algorithms, such as ray tracing, path tracing, distribution ray tracing, and others. Those algorithms are tracing rays via reflections with physically accurate models of surfaces and real light sources. Among those global illumination algorithms, we selected the Whitted ray-tracing algorithm to implement real-time ray-tracing rendering.

We developed a real-time ray-tracing GPU semiconductor intellectual property (IP) called RayCore in 2011 and a real-time KD-tree–building semiconductor IP

called RayTree in 2013. We developed RayChip to support real-time ray tracing for embedded applications by using two semiconductor IPs [1]. The RayChip provides sufficient performance for real-time ray tracing and for use of the familiar OpenGL for Embedded Systems (OpenGL ES) application programming interface (API).

When using ray-tracing software on a conventional desktop computer, the NVIDIA GTX 285 provides 142M rays/s performance only for primary rays, but degrades to 61M rays/s when tracing diffuse rays [3]. RayChip provides about 80M to 108M rays/s performance while it traces primary rays, secondary rays, refraction rays and shadow rays.

2. Related Work

To implement a hard-wired ray-tracing architecture, SaarCOR was proposed in 2004. SaarCOR [4] includes a ray generation unit, a shading unit, a KD-tree traversal unit and an intersection test unit. RPU [5], based on the SaarCOR architecture, was developed with a programmable shading unit in 2005. D-RPU [6], which was extended from RPU, was developed for BKD-trees [7] in 2006. CausticOne, which includes a hard-wired intersection test unit as a form of field-programmable gate array (FPGA) board, was announced in 2009. Caustic Series2, based on the CausticOne architecture, which includes Caustic RT2 custom chips to accelerate ray-tracing rendering on desktop PCs, was announced in 2013. For mobile real-time ray tracing, the RayCore internal architecture was presented in 2014 [2].

3. RayChip Architecture

RayChip is designed to achieve real-time performance for ray-tracing rendering. The chip processes multiple ray bounces recursively to create realistic images, and integrates KD-tree generation hardware to accelerate ray-tracing rendering for dynamic scenes.

As shown in Fig. 1, several processes are required to acquire the final rendered image. First, the central processing unit (CPU) loads three-dimensional (3D) graphics data from storage into memory. Second, the CPU selects the graphics object data to be rendered within the screen. Typically, 3D graphics data for ray tracing are used as a form of tree structure to manipulate the graphics object. The chip adapts RayTree, which generates a tree structure for fast rendering and reducing the CPU process load. RayTree generates a KD-tree structure because the KD-tree structure is suitable for ray-tracing rendering [2]. After the KD-tree is generated by RayTree, it is transferred to RayCore. RayChip performs processes with the following steps.

(1) Generate primary rays per pixel and secondary rays (ray generation).

(2) Find the object reached from a ray (traversal), and then find the exact triangle of the object hit by the ray recursively (intersection test).

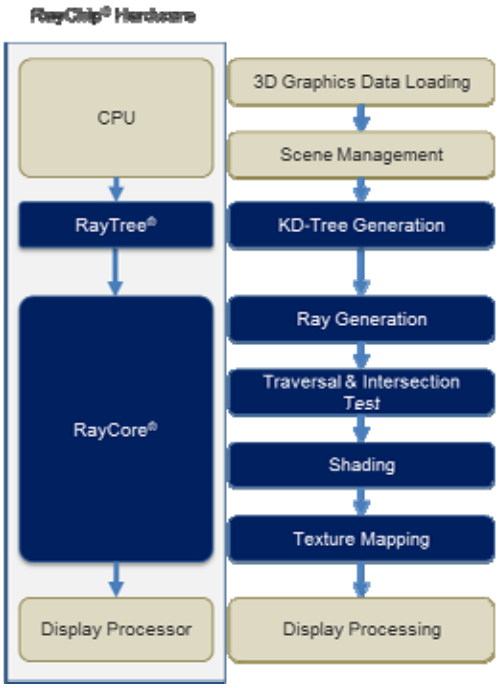

Fig. 1. RayChip processes.

(3) Determine the color value of the pixel according to Phong illumination (shading).

(4) Perform texture mapping with a mip-mapping algorithm (texture mapping).

Finally, the display processor transfers the final rendered image, which is stored in a frame buffer of the display device.

In order to operate standalone ray-tracing processing, the chip integrates the CPU, memory, storage, display processor, RayCore 1000 series, RayTree and other peripherals, as shown in Fig. 2. We used Fujitsu 55nm low-power technology to develop RayChip. An ARM11 CPU and dual 512MB double data rate 3 (DDR3) memory is used. The ARM11 CPU provides suitable performance to support ray-tracing rendering up to HD resolution and 60fps, while RayTree reduces the CPU load for tree-structure building. The chip integrates a 2D engine as the display processor, which supports a 2D BitBLT function and full HD resolution scaling with an HDMI 1.3 interface. As a general interface and a storage interface, the chip includes USB 2.0 and secure digital input/output (SDIO) as an SD memory interface. To provide HD resolution at minimum, six RayCores and RayTree are also integrated into the chip; six RayCores can generate HD images in real time, and full HD or above images for offline rendering.

The ARM11 CPU and DDR3 memory operate at 533MHz, whereas RayCore and RayTree operate at 266MHz. RayCore consumes 0.85W per core, providing 100M rays/s and 60fps performance. The total size of the six RayCores is 30M gate counts. RayTree builds 1M triangles into a KD-tree structure in a second. The size of RayTree is 3.5M gate counts. RayChip is designed for various operating systems (OSs) by a simple change of software in the SD memory card. After power-on,

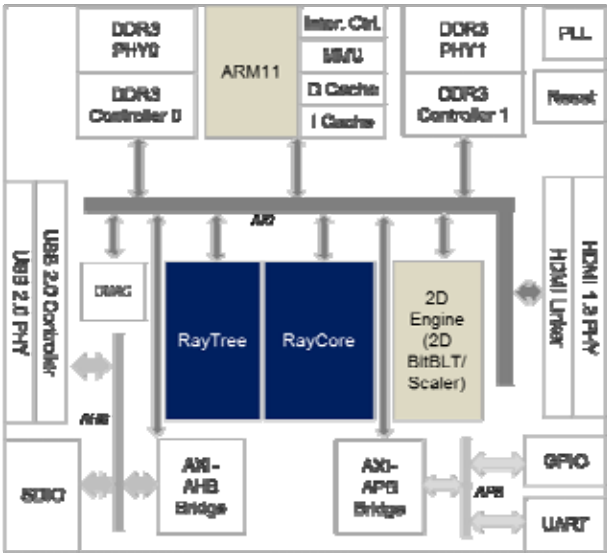

Fig. 2. RayChip architecture.

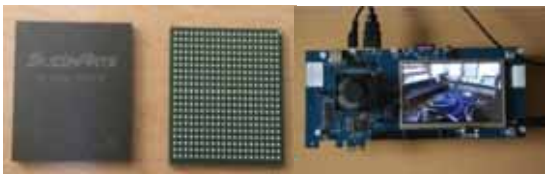

Fig. 3. RayChip & Test board.

RayChip loads the Linux kernel, libraries and other software into an SD memory card and starts the initial program immediately. RayChip, shown in Fig. 3, is packaged by FBGA type with 400 pins. The chip size is 9,630μm × 9,430μm with a scribe lane structure. The test board of RayChip is equipped with HDMI, SD, and USB interfaces, a 5-inch touch LCD display, etc., as shown in Fig. 3.

3.1 RayCore Architecture

In order to implement real-time ray tracing, we designed fully hard-wired logic called RayCore to minimize silicon area and power consumption.

RayCore adopts a fixed-pipeline architecture based on a multiple instruction, multiple data (MIMD) architecture, which has the best performance for ray tracing among single hardware architectures [2]. MIMD architecture for ray tracing shows better performance than single instruction, multiple data (SIMD) architectures, such as NVIDIA's approach and Imagination Technology's approach [2]. MIMD architecture processes independent rays immediately and allows high hardware utilization, whereas SIMD architecture waits for all reflections of similar rays because of the ray coherence problem [3]. In addition, the traversal unit and the intersection test unit are consolidated. The consolidated logic is called a unified traversal and intersection (T&I) test unit, which is designed as an MIMD architecture. These unified traversal and intersection test units perform the computations in a single pipeline and eliminate the load imbalance problem.

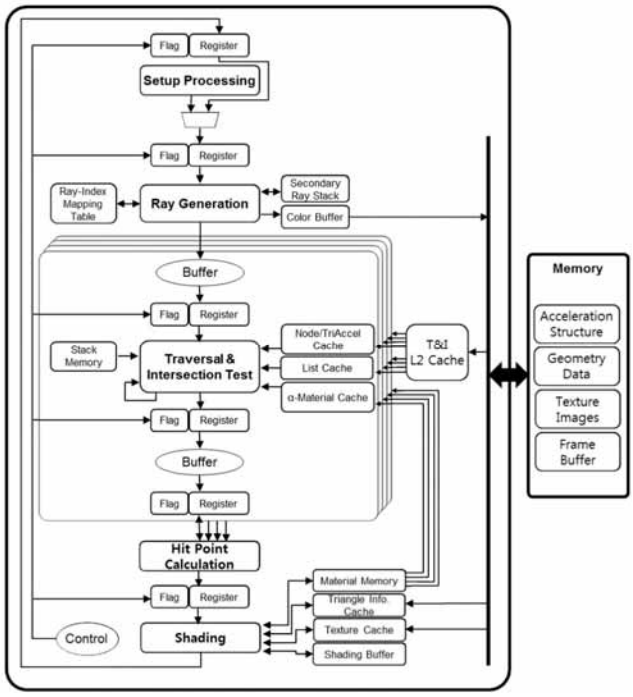

Fig. 4. RayCore architecture.

To process large 3D graphics data, multiple hierarchical caches are required to minimize memory bandwidth. RayCore uses a two-level cache hierarchy and integrates an efficient memory latency hide technique named "looping for the next chance" [2]. RayCore generates ray threads as simple multi-threading. The cache system sets the ray thread to idle mode when a cache miss occurs and pre-fetches data for the next loop. And the cache system sets the ray thread to active mode at the next loop to re-access the cache.

The detailed architecture of RayCore is shown in Fig. 4, and the processes are as follows.

(1) Setup-processing unit: passes ray information to ray generation.

(2) Ray-generation unit: primary/secondary ray generation.

(3) Unified traversal & intersection test units: tree node traversal and ray-triangle intersection test.

(4) Hit-point calculation unit: calculate the position (x,y,z) of the hit point.

(5) Shading unit: Phong illumination, texture mapping, and normal mapping.

RayCore supports selectively dithered rendering (SDR) and super sampling capabilities. SDR decreases the number of rays to be computed while preserving image quality. Super sampling increases the number of rays and generates high-quality images with less aliasing.

3.2 RayTree Architecture

Building an accelerated structure for real-time ray tracing has another bottleneck in dynamic scenes that include moving objects. A dynamic scene requires regeneration of the tree structure for each scene. RayTree

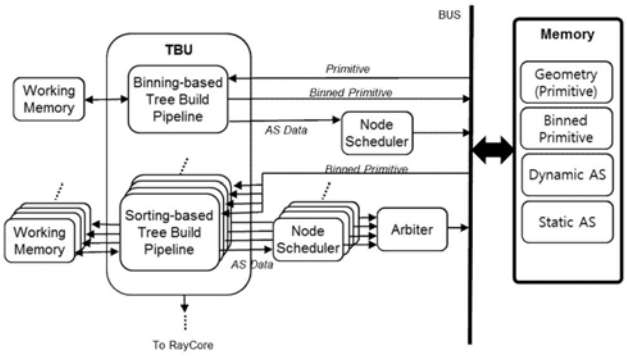

Fig. 5. RayTree architecture.

provides better performance for KD-tree generation than other approaches [2]. RayTree performs fast KD-tree building without tree-quality degradation. As shown in Fig. 5, RayTree consists of two main units. First, a binning-based tree build pipeline unit uses a binning method [8] to make upper-level nodes, called scan-trees. A sorting-based tree build pipeline unit adopts a sorting method [9] for making lower-level nodes, called KD-trees. To minimize off-chip memory access, RayTree utilizes internal static random access memory as working memory for sorting, and reorders node data for an efficient burst transfer in memory.

3.3 Software & API Integration

To operate the ARM CPU, we ported and installed the ARM Linux 3.15.4 kernel, libraries and necessary device drivers, such as HDMI, SD memory, and so on. The RayCore API and the RayTree API are installed for application of real-time ray tracing.

Since RayChip has dual DDR3 memory, we divided use of memory into two groups according to high frequency of memory usage. The Linux OS and applications are located in the first memory, and the framebuffer is located in the second memory to minimize bus bottleneck. The RayCore API is designed to co-operate OpenGL ES Version 1.1. New functions or parameters are added for ray tracing specific features. The RayCore API uses modified versions of the functions and parameters in OpenGL ES Version 1.1. In addition, only a few functions are added for the ray tracing–specific capabilities.

4. RayChip Evaluation

Prior to the performance test, we performed functional tests of RayChip to confirm our synthesis and timing criteria. The criteria are clock and power. We used test firmware on the test board to identify the correct operations of individual IPs. The clock for the ARM 11 CPU and DDR memory was correctly supplied at 533MHz, and the 266MHz clock for RayCore and RayTree was properly generated via phase-locked loop (PLL). As core voltage, 1.2V power input is used. The chip also uses 1.8V and 3.3V power inputs to support SD and USB. These power inputs is sufficient to operate all IPs in RayChip, including DDR memory, concurrently.

In order to estimate the performance of RayChip, we used two performance metrics: rays/s and frames/s. In order to evaluate the performance of ray tracing, we measured how many rays are processed in a second. In addition, to get overall RayChip performance, we measured the processing time from scene management to displaying the final image on screen.

As shown in Fig. 6, there are several 3D graphic images acquired by RayChip. These images show realistic global shadow and reflection, and depict exquisite refraction and transmission on transparent objects. The frame rates of these images from RayChip vary from 10fps to 45fps as announced by Park et al. [1].

We used two sets of 3D graphics content to evaluate the performance of RayChip. The first is high-quality content named Times Square to measure maximum performance, and the other is interactive user interface (UI) content named Water Drop UI, which is controllable by finger-touch, as shown in Fig. 7. Times Square is heavy content to perform real-time ray tracing, which contains 97,638 triangles and 159MB textures with a lot of reflections, refractions and transmissions. Water drop UI

Fig. 6. Ray tracing images generated from RayChip.

Fig. 7. Benchmarked 3D graphics contents (Left: Times Square, Right: Water Drop UI).

Table 1. RayCore performance in HD resolution.

Scene	Rendering Mode		Performance		
	SDR	Super Sampling	M rays/s		
			Min	Max	Avr.
Time Square	off	off	88.99	117.47	105.57
	on	off	70.26	114.33	96.03
	off	**on**	**92.87**	**118.45**	**107.75**
	on	on	86.64	115.77	103.99
Water Drop UI	off	off	72.09	86.27	80.93
	on	**off**	**73.43**	**84.90**	**80.02**
	off	on	68.93	86.72	81.64
	on	on	71.56	88.89	83.78

Table 2. RayChip performance in HD resolution.

Scene	Rendering Mode		Performance		
	SDR	Super Sampling	Frames/s		
			Min	Max	Avr.
Time Square	off	off	11.58	32.57	17.70
	on	**off**	**18.27**	**65.16**	**31.82**
	off	on	2.98	8.18	4.50
	on	on	6.11	17.99	9.27
Water Drop UI	off	off	29.74	34.68	32.76
	on	**off**	**46.50**	**53.45**	**50.65**
	off	on	7.15	8.67	8.29
	on	on	14.44	21.03	16.56

requires a lot of computation on refractions and transmissions on the water drops, with tracing of 15 ray bounces. Water drop UI consists of 21,220 triangles and 12MB textures.

We measured RayCore performance with SDR and super sampling rendering modes. In Table 1, the RayCore of the RayChip shows the best performance when super sampling is on, since cache hit ratio increases due to increasing coherence of texture map and polygon. When SDR is on, the performance slightly decreases because cache hit ratio decreases due to decreasing coherence of texture map and polygon. With two sets of heavy content, performance varies from 80M rays/s to 108M rays/s.

Overall performance of RayChip is shown in Table 2. The measured frame rates are over 30 frames/s, which meets real-time constraints. When super sampling is on, the frame rate decreased significantly, requiring four times more rays to be computed than the super-sampling-off mode.

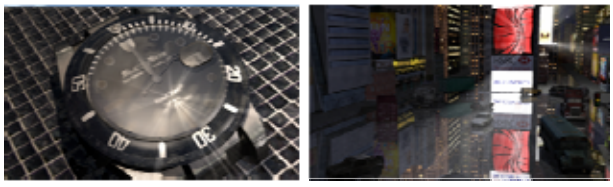

Fig. 8. Luminescence effect on ray-traced images using a shader.

5. Conclusion

In this paper, we introduced a new GPU chip architecture for real-time ray tracing. RayChip is the world's first real-time ray-tracing chip targeted at embedded applications, such as TV, media boxes and game consoles. Images better than quadruple HD can be rendered with multiple RayChips.

Despite supporting real-time ray tracing, the weakness of RayChip is the absence of collaboration with current shader GPUs. Incorporating current GPUs and RayCore, many special effects are enabled with real-time ray-tracing functionalities. Fig. 8 shows the luminescence effect on ray-traced images using a current GPU. Fig. 10's images are generated by an FPGA board and a PC desktop.

Real-time ray-tracing chip architecture will evolve to provide high performance when implementing a realistic graphical user interface and virtual reality in the near future.

References

[1] Woo-Chan Park et al., "RayChip®: Real-time Ray-tracing Chip for Embedded Applications," *in A Symposium on High-Performance Chips, HotChips Presentation*, Aug. 2014. Article (CrossRef Link)

[2] Jae-Ho Nah et al., "RayCore: A Ray-Tracing Hardware Architecture for Mobile Devices," *in ACM Transactions on Graphics*, Vol. 33, No. 5, Article 162, Aug. 2014. Article (CrossRef Link)

[3] D. Kopta, et al., "Efficient MIMD Architectures for High-Performance Ray Tracing," *in IEEE International Conference on Computer Design*, pp. 9-16, Oct. 2010. Article (CrossRef Link)

[4] Jorg Schmittler et al., "Realtime ray tracing of dynamic scenes on an FPGA chip," *in Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pp. 95–106. 2004. Article (CrossRef Link)

[5] Sven Woop et al. "RPU: A programmable ray processing unit for realtime ray tracing." *ACM Trans. Graph*, pp. 24, 3, 434–444. 2005. Article (CrossRef Link)

[6] Sven Woop et al. "Estimating performance of a ray-tracing ASIC design." *in Proceedings of the IEEE/EG Symposium on Interactive Ray Tracing*, pp.

7–14. 2006. Article (CrossRef Link)
[7] Sven Woop et al. "B-KD trees for hardware accelerated ray tracing of dynamic scenes." *in Proceedings of the 21 ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware (GH'06)*, pp. 67–77. 2006. Article (CrossRef Link)
[8] M. Shevtsov, A. Soupikov, and A. Kapustin, "Highly parallel fast *kd*-tree construction for interactive ray tracing of dynamic scenes," *Computer Graphics Forum*, Vol. 26, No. 3, pp. 395-404, Sept. 2007. Article (CrossRef Link)
[9] I. Wald and V. Havran, "On Building Fast kd-trees for Ray Tracing, and on Doing That in O(N log N)," *IEEE Symposium on Interactive Ray Tracing*, pp. 61-69, 2006. Article (CrossRef Link)

Hyung-Min Yoon is CEO, Siliconarts, Inc. He received his BS in Geology, and his MS and PhD in Computer Science from Yonsei University, Korea, in 1998 and 2006, respectively. He worked as a Project Leader at Samsung Advanced Institute of Technology, Korea.
His current research interests include real-time ray tracing GPUs, 3D graphics algorithms and applications.

Byoung-Ok Lee is VP, Research Lab, Siliconarts, Inc. He received his BS in Electronic Communication Engineering from Hanyang University, Korea, in 1991. He worked as Head of the SoC center at MtekVision, Ltd., Korea, and as a research engineer at LG Electronics, Ltd., Korea. His current research interests include real-time ray tracing GPUs, 3D graphics algorithms and SoC applications.

Cheol-Ho Jeong is Principal Research Engineer, Research Lab, Siliconarts, Inc. He received his BS in Software Engineering from Soongsil University, Korea, in 1996, and his MS and PhD in Computer Science from Yonsei University, Korea, in 1998 and 2004, respectively. He worked as a project leader at the Samsung Electronics Digital Media R&D center. His current research interests include real-time ray tracing GPUs, 3D graphics algorithms and SoC architecture

Jin-Seok Hur is Senior Research Engineer, Research Lab, Siliconarts, Inc. He received his BS in Electrical Engineering from Yonsei University, Korea, in 2002, and his MS in Digital Entertainment from Korea Polytechnic University, Korea, in 2013. He worked as a Project Leader at Qosba, Korea. His current research interests include real-time ray tracing GPUs, 3D graphics algorithms and 3D H/W architecture.

Sang-Gon Kim is Principal Research Engineer, Research Lab, Siliconarts, Inc. He received his BS and MS in Physics from Yonsei University, Korea, in 1997 and 1999, respectively. He worked as VP, Research Lab, Geon-Tech Korea, Ltd., Korea, and as R&D Team Leader, ARmax, Ltd., Korea. His current research interests include real-time ray tracing GPUs, 3D graphics algorithms and applications.

Woo-Nam Chung is Senior Engineer, Siliconarts, Inc. He received his BS, MS and PhD in Computer Science from Yonsei University, Korea. His current research interests include real-time ray tracing GPUs, 3D graphics algorithms and applications.

Yong-Ho Lee is a researcher, Research Lab, Siliconarts, Inc. He received his BS and MS in Computer Science from Yonsei University, Korea, in 2011 and 2014, respectively. His current research interests include real-time ray tracing GPUs, 3D graphics algorithms and VR applications.

Woo-Chan Park was born on 1 May, 1970, in Korea. He received his BS, MS, and PhD in Computer Science from Yonsei University, Seoul, Korea, in 1993, 1995, and 2000, respectively. He is currently a Professor, Sejong University. His research interests include 3D rendering processor architecture, ray-tracing accelerators, parallel rendering, high-performance computer architecture, computer arithmetic, and ASIC design.