The definitive version is available at the ACM Digital Library.

Adaptive Multi-Rate Ray Sampling on Mobile Ray Tracing GPU

Won-Jong Lee[†*] Seok Joong Hwang[‡] Youngsam Shin[†] Soojung Ryu[†] Insung Ihm[§]

[†] SAMSUNG Advanced Institute of Technology [‡] now at SKT [§] Sogang University

Figure 1: *Adaptive ray sampling on mobile ray-tracing GPUs. Similarity test classifies the samples into similar (yellow) and dissimilar (blue) (left). The similar samples are shaded by linear interpolation instead of full ray tracing. Compared with reference rendering (FSAA2×2) (middle), our implementation achieves 1.83 times better performance-power efficiency without compromising quality (PSNR 30.5dB) (right).*

Abstract

We present an adaptive multi-rate ray sampling algorithm targeting mobile ray-tracing GPUs. We efficiently combine two existing algorithms, adaptive supersampling and undersampling, into a single framework targeting ray-tracing GPUs and extend it to a new multi-rate sampling scheme by utilizing tile-based rendering and frame-to-frame coherency. The experimental results show that our implementation is a versatile solution for future ray-tracing GPUs as it provides up to 2.98 times better efficiency in terms of performance per Watt by reducing the number of rays to be fed into the dedicated hardware and minimizing the memory operations.

Keywords: ray tracing, GPU, mobile, low-power, resampling

Concepts: •**Computing methodologies** → **Ray tracing;** *Graphics processors;*

1 Introduction

As the recent mobile market is growing continuously, real-time ray tracing has attracted considerable attention for future graphics applications, such as UX/UI, high-quality AAA games, and virtual reality. However, real-time ray tracing on the current mobile CPUs and GPUs continues to be a challenge because of their limited computing power, memory bandwidth, and energy budget. Therefore, recently, various hardware-based ray tracing solutions such as fully dedicated hardware [Nah et al. 2014], hardware-software hybrids [Lee et al. 2013], and GPU IPs [McCombe 2014], has been proposed to solve these problems in mobile devices.

Today's ray-tracing hardware IPs for mobile devices exhibit reasonable performance (∼300 Mrays/s) and achieve real-time rendering of basic ray-tracing effects, such as shadows, reflections, and occlusions, at FHD resolution (1920×1080) in cooperation with rasterization GPUs, which is called hybrid rendering. However, it still cannot support the requirements of production-level applications, such as game engines with full ray tracing (1−3 Grays/s) [Johnstone et al. 2015]. Furthermore, these computing requirements can be increased because future graphics applications might require extremely high resolution (>4K).

In order to narrow this gap, we can consider the *adaptive ray sampling* approach, which can reduce the computing cost while maintaining the image quality. The key idea behind this approach is utilizing the pixel-to-pixel coherence contained in graphics applications and reusing the results of the ray-tracing of neighboring pixels for current pixels. That is, once it has traced rays for sparse samples and classified the remaining samples into two types – similar and dissimilar (to the sparse samples) – then it replaces the ray-tracing operation with inexpensive linear interpolation for similar samples.

In this paper, we present an adaptive multi-rate ray sampling algorithm for improving the performance-power efficiency of the mobile ray-tracing GPU, which is called SGRT (Samsung GPU based on Ray Tracing) [Lee et al. 2012; Lee et al. 2013]. We combine the existing two algorithms targeting commodity GPUs, *adaptive ray supersampling* [Jin et al. 2009] and *adaptive ray undersampling* [Kim et al. 2016], into a single framework and efficiently map it to the programmable shader core, which can significantly reduce the computing cost in fixed-function hardware. Unlike the previous works, our implementation aggressively utilizes *tile-based rendering*, which allows the operation to be performed using the on-chip internal memory without having to access the external memory. In addition, we extend this algorithm to a new *multi-rate sampling* scheme to apply the variable sampling rates according to the ratio of similarity in a tile, which can provide additional performance improvements. The experimental results show that our implementation is a versatile solution for future ray-tracing GPUs, as it achieves up to 2.98 times better efficiency in terms of performance per Watt by substantially reducing the computing cost of the dedicated hardware and minimizing the memory operations. To summarize, the main contributions of this paper are:

- Integration of adaptive ray supersampling and subsampling algorithms into a single framework,
- Proposing a new multi-rate sampling scheme to control sampling rates according to the similarity ratio in the tile, and
- Implementation of optimized algorithms to improve the performance-power efficiency for mobile ray-tracing GPUs.

*e-mail:joe.w.lee@samsung.com

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s). © 2016 Copyright held by the owner/author(s).

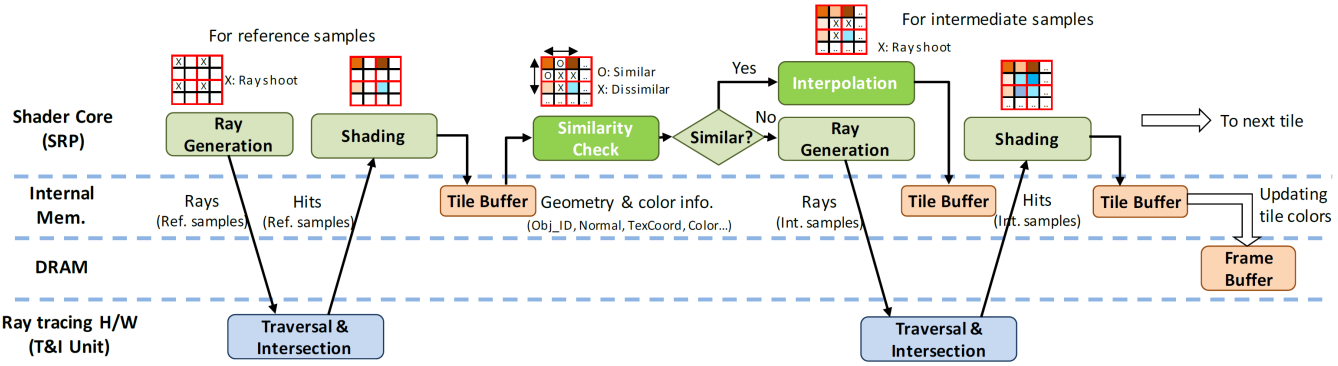


Figure 2: *Overview of adaptive ray sampling on ray-tracing GPUs. Additional kernels for similarity check and linear interpolation are easily implemented because the SRP supports full programmability. Our tile-based approach allows these operations to be performed using on-chip internal memory without having to access external memory*

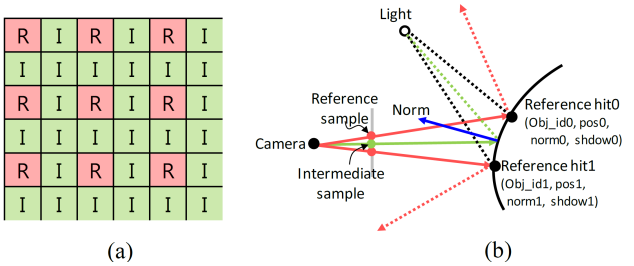


Figure 3: *(a) Reference (red) and intermediate samples (green). (b) Rays shot to the reference and the intermediate samples.*

2 Related Work

Despite the recent advance in mobile computing, the need for developing dedicated hardware has been great concern to allow high-quality graphics based on high performance and low power solutions. Spjut et al. [2012] proposed a version of the MIMD TM [Kopta et al. 2010] in which the number of thread multiprocessors and active threads was reduced, making it suitable for mobile devices. Kim et al. [2012] also proposed a dedicated multi-core processor for mobile ray tracing, which can be configured to both SIMT and MIMD modes. However, the above architectures could not provide a sufficiently high performance (less than 30 Mrays/s) for real-time ray tracing of real-world applications.

Lee et al. [2012] firstly presented a mobile GPU based on ray tracing called SGRT which combines the advanteges of programmable DSP cores called SRP (Samsung Reconfigurable Processor) [Lee et al. 2011] and a dedicated hardware, called T&I unit. This architecture has been revised with advanced features such as 2-AABB traversal units [Lee et al. 2014], Reorder Buffer [Lee et al. 2015b], and hybrid number system [Hwang et al. 2015]. Recently, a commercial mobile GPU featuring ray-tracing hardware blocks, called PowerVR GR6500 [McCombe 2014], has been announced. A key feature of this GPU is to adopt the hybrid rendering to combine the advantages of ray tracing and rasterization. Lee et al [2015a] also proposed a hybrid rendering architecture based on tile-based mobile GPUs. The above architectures realize real-time rendering of basic raytracing effects but it is still not enough to support AAA games and high-resolution VR applications.

Adaptive sampling in ray-tracing domain has been an important topic and was extensively utilized in past research such as hierarchical adaptive supersampling [Whitted 1980], optimal stochastic sampling [Dippe and Wold 1985], adaptive hierarchical sampling [Kajiya 1986], and two-level sampling method [Mitchell 1987]. These algorithms utilized the geometry information in image and object space for adaptive sampling. Recently, these algorithms evolved to be more advantageous to commodity PC and mobile GPUs. Jin et al. [2009] presented a selective and adaptive supersampling method, optimized for massively parallel modern GPUs. Kim et al. [2016] adopted this algorithm to adaptive undersampling method with novel post-correction filters, optimized for mobile GPUs.

Rasterization domain has also considered adaptive sampling approaches [Vaidyanathan et al. 2014; He et al. 2014; Clarberg et al. 2014] as the complexity of the fragment shading was sharply increased in commercial graphics applications such as AAA games. According to the various factors such as importance, interest, level of detail (LoD) and camera effects, these algorithms adaptively control the shading rates. However, these rasterization-based approaches are not extendable for ray-tracing GPUs though they are similar to the method presented in this paper.

3 Adaptive Multi-Rate Ray Samping on SGRT

In this paper, we present an adaptive multi-rate ray sampling algorithm targeting mobile ray-tracing GPUs. As a baseline algorithm, we utilize two adaptive methods, supersampling [Jin et al. 2009] and undersampling [Kim et al. 2016], developed for a commodity GPU ray tracer. Though these were invented for a different purpose, the key idea behind them is the same: to reuse the results of the ray tracing of the neighboring pixels for the current pixel. The brief operation flow is as follows. Initially, the screen area is partitioned into two types. Figure 3 shows an example of partitioning through 2×2 blocks, where regularly distributed set, marked as R, forms a group of reference samples (R-samples). The other samples, marked as I, are called intermediate samples (I-samples). This example shows 2×2 pixel partitioning, but this can be changed (e.g 3×3, 4×4) according to the sampling ratio. We integrate the super and undersampling algorithms into a single framework. A sample can be mapped to a sub-pixel in the supersampling mode, and it can be mapped to a pixel in the undersampling mode. The rendering consists of three steps.

- Ray tracing for R-samples are carried out. In this step, the sample color is computed and the additional geometry information to be used in the next step is collected (Figure 3b). The geometry information includes the hit object id, normal vector, position vector, and shadow bits of the hit point. The

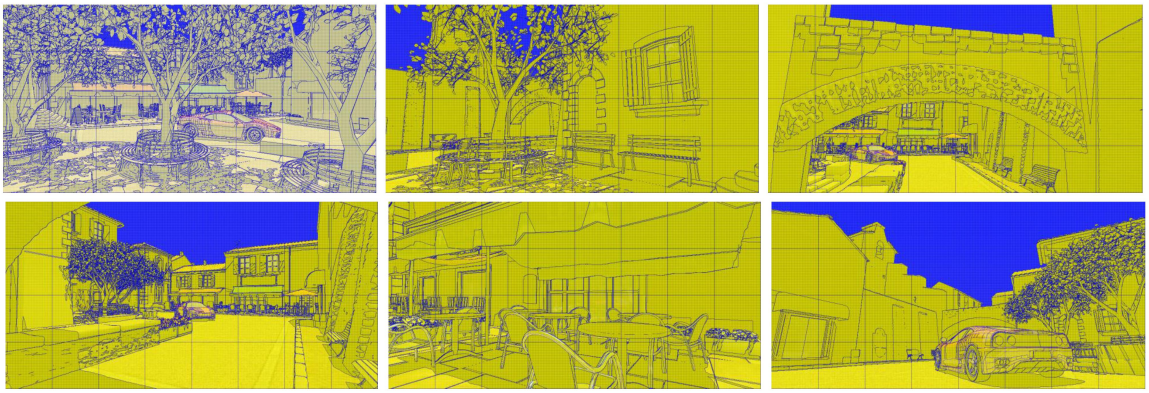


Figure 4: *Visualization of the similarity check for the six frames extracted from the animation sequence of Provence scene (yellow: similar, blue: dissimilar). Frame numbers are 1, 150, 340, 400, 600, and 700. Similarity ratio can be variant across the tiles in the same frame, and also variant across the frame for the same tiles.*

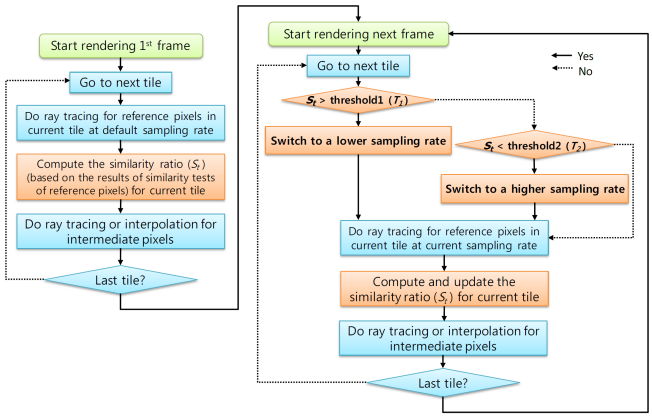


Figure 5: *Operational flow of the proposed adaptive multi-rate ray sampling using frame coherency.*

shadow bits are set if and only if a shadow is cast at the surface hit point with respect to the corresponding light source.

- Given this geometry information, the similarity check is conducted between the adjacent R-samples. This can be conducted with several tests if the object ids are the same, if the directions of normal vectors are similar, and if the shadow bits are the same. This test consists of two sub-steps for the X- and the Y-direction.
- Operations for I-samples are carried out. If the current I-sample passes the similarity check, the color can be calculated by linear interpolation of the neighboring R-samples. Otherwise, the normal ray-tracing for the current I-sample is conducted. For the secondarys, the above procedures can be applied with the same manner.

We extend this algorithm and efficiently map it onto the ray-tracing GPU. As a target GPU, we utilize SGRT [Lee et al. 2012; Lee et al. 2013], which consists of two components: dedicated hardware called the T&I unit and a programmable shader called SRP [Lee et al. 2011]. The T&I unit accelerates the traversal and intersection, which is the dominant operation in ray tracing. The other operations are implemented by software and can be executed in SRP. The communication between these two components is efficiently performed via a direct interface [Shin et al. 2013] without any external memory overhead. Figure 2 shows the overview of ray tracing based on adaptive ray sampling on SGRT. For a simple explanation, the procedures for secondary rays are omitted. Additional kernels for adaptive ray sampling, such as the similarity check, interpolation, and color update, are easily implemented by the SRPs full programmability. Conversely, the architecture using fully dedicated hardware [Nah et al. 2014] has to add additional logic to support these new features due to lack of flexibility, eventually increasing area costs. In addition, unlike the previous works [Jin et al. 2009; Kim et al. 2016], we adopt the tile-based approach to avoid expensive memory operations. In other words, we conducted ray tracing on a per tile (a subdivided area of the screen) basis, which can localize the buffer-storing geometry data, called the *tile-buffer*, into the internal memory in the SRP. Thus, the access to the external memory can be minimized during the rendering time.

The operational flow of our implementation is as follows. The rays for R-samples generated from the kernel in the SRP are transmitted to the T&I unit through the direct interface. The T&I unit performs the traversal and intersection operation and sends hit points back to the SRP. The shading kernel in the SRP is executed with these hit points. Additionally, the shading kernel stores the geometry and the color information for each hit point of the tile buffer in the internal memory. For the I-samples, the similarity check kernel is executed with the geometry data in the tile buffer and stores the binary results, test pass or fail, in the tile buffer again. This kernel produces two data streams based on the results of the similarity test. The I-samples that have passed the similarity test are fed into the interpolation kernel, and the I-sample that have failed the test are fed into the ray-generation kernel to restart normal ray tracing. Each kernel is implemented with the inspector-executor model [Hwang et al. 2014] to reduce the side-effects of branch divergence and maximize the efficiency of stream processing. If the color computation for the current tile is finished, the color data in the tile buffer are flushed into the external frame buffer by the DMA (Direct Memory Access) function. In the supersampling mode, the color update includes more steps to average the sample colors into a pixel color; thus, this mode allocates more memory with proportion to the sampling rate compared with the undersampling mode.

In addition, we extend the baseline algorithm to a more aggressive multi-rate sampling method by utilizing the tile-based approach and frame-to-frame coherence. Figure 4 is the visualization of the similarity check for the six frames extracted from the animation sequence of the Provence scene (yellow: similar, blue: dissimilar). the similarity ratio can be variant across the tiles in the same frame and across the frame for the same tiles. Thus, the tiles with a rel-

Figure 6: *Test scenes. Teapot (15K triangles), Chess (42K), BMW (55K), Chemical Lab.(98K), and Provence (600K).*

atively high similarity ratio can use the lower sampling ratio in the similarity check step (e.g. one R-sample per 2×2 block → one R-sample per 4×4 block), which can reduce the additional computing cost without significantly damaging the image quality. This can enable the multi-rate sampling to apply different sampling rates to the tiles according to the similarity ratio. The per-tile similarity ratio can be referenced from the previous frame, and this value is compared with the predefined thresholds. Based on this result, we can dynamically control the sampling rates on a per-tile basis, as shown in Figure 5. Alternately, we can switch the threshold values in order to avoid temporal aliasing caused by discretization jump when the sampling rates are changed. Finally, our adaptive ray sampling with multi-rate feature can significantly improve the performance-power efficiency by reducing the number of rays to be traced.

4 Experimental Results

To verify and evaluate how our framework can reduce the computing cost of the T&I unit, we utilized the cycle accurate simulator of the SGRT integrated with the energy model. This simulator provides rendered images, total execution cycles, hardware utilization, cache statistics, and expected performance. We used the energy and power model of [Lee et al. 2015b] which utilized a custom model based on the database built with the power values per component from Synopsys PrimTime PX [SYNOPSYS 2016] with SAMSUNG 14nm LPP process technology [Samsung 2016]. Cycle accurate simulator produced the activation counters for each hardware blocks and the aggregated power value can be obtained from this energy and power model. The hardware configuration of the T&I is the same as the T&I unit 2.0 [Lee et al. 2014]. As the test scenes, we used five datasets (Figure 6), Teapot (15K triangles), Chess (42K), BMW (55K), Chemical Lab.(98K), and Provence (600K). These scenes have enough secondary rays including shadows, refractions, and reflections. Test scenes were all rendered at 2048×1024 resolution. To make a comparion of the performance and quality, we choosed two reference conditions:

- FSAA (Full Scene AntiAliasing) 2×2: four rays are shot to each pixel of the target resolution.
- Standard 1×1: one ray is shot to each pixel of the target resolution.

These two references are compared with adaptive supersampling and adaptive undersampling mode, respectively.

Figure 7 shows the relative performance, power consumption, and their ratio (performance per Watt), which are measured for T&I unit applied by two adaptive ray-sampling approaches. Overall, the adaptive methods outperforms the reference methods up to 3× because the ray tracing for the similar I-samples can be replaced with interpolation operation in the SRP and the corresponding rays are not transmitted to the T&I unit. The performance improvements are proportional to the number of rays cut because the access to the external memory was minimized by tile-based rendering. In terms of power consumptions, the adaptive methods consumed slightly more powers (9%), this is because of the faster rendering speed and the higher pipeline utilization (but, the trends can be reversed if this values are translated into energy). As a result, the adaptive methods are advantageous as it achieves up to 2.98 times better efficiency in terms of performance per Watt. There are variants among the test scenes. The scenes with a higher similarity and lower frequency (Teapot, BMW) exhibit the better gains (2.43−2.98×). Even in the case of the scenes with a relatively higher frequency (Chess, Chemical Lab., Provence), they exhibit at least 1.53× gains. Lastly, we have to consider the overheads for additional kernels in the SRP. In our profiling results, the decomposition of the execution cycles in the SRP, we found that the overhead was 11% (5% for similarity check and 6% for interpolation) and it cannot hinder the execution of the T&I unit.

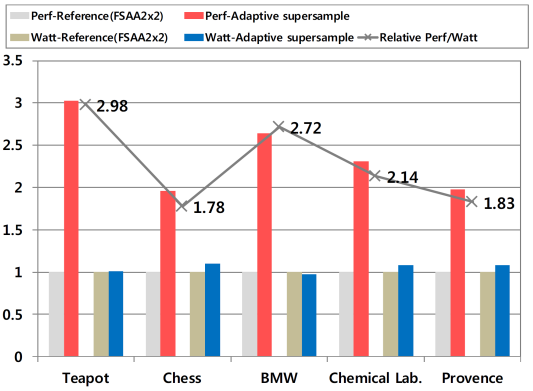
Table 1: *PSNR of the rendered scenes by adaptive ray sampling*

Sampling mode	Teapot	Chess	BMW	Chemical Lab.	Provence	**Average**
Adaptive supersampling	48.99	28.53	34.11	36.28	30.50	**35.68**
Adaptive undersampling	42.13	34.73	41.68	42.60	37.71	**39.77**

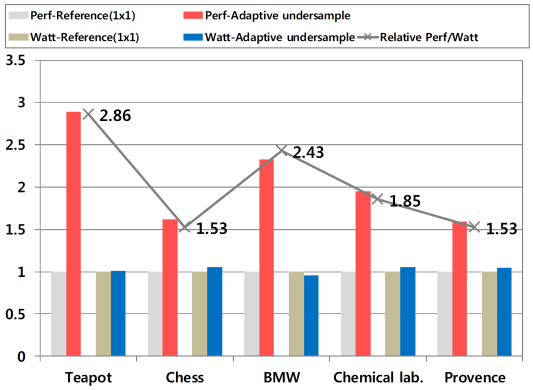
To evaluate the image quality, we computed the PSNR values to the references for each test scene (Table 1). In average, we obtained the values of 35.68 and 39.77 dB which are visually tolerable levels as shown in Figure 1. We believe that the image quality can be better if we will add more test conditions for similarity check in image space (e.g. texture and color difference) and apply post-correction method [Kim et al. 2016] to adjust the errors.

5 Conclusion and Future Works

In this paper, we presented an adaptive multi-rate ray sampling algorithm targeting mobile ray-tracing GPUs. We efficiently combined two existing algorithms into a single framework and extended it to a new multi-rate sampling scheme by utilizing tile-based rendering and frame-to-frame coherency. Our adaptive ray sampling



(a) Adaptive Supersampling



(b) Adaptive Undersampling

Figure 7: *Relative performance, power consumption, and the ratio of performance per Watt (to reference), for the execution of T&I unit applied by two adaptive sampling approaches.*

could achieve up to 2.98 times better efficiency in terms of performance per Watt by reducing the number of rays to be fed into the dedicated hardware and minimizing the memory operations.

In our current framework, the multi-rate sampling has not yet been applied. In near future, we will add this feature for further performance improvements and equip the more accurate similarity tests and post-correction filter to make the better quality by covering the corner cases like thin objects [Kim et al. 2016].

References

CLARBERG, P., TOTH, R., HASSELGREN, J., NILSSON, J., AND AKENINE-MÖLLER, T. 2014. AMFS: Adaptive multi-frequency shading for future graphics processors. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2014) 33*, 4, Article No. 141.

DIPPE, M., AND WOLD, E. 1985. Antialiasing through stochastic sampling. In *Proceedings of ACM SIGGRAPH 1985*, 69–78.

HE, Y., GU, Y., AND FATAHALIAN, K. 2014. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2014) 33*, 4, Article No. 142.

HWANG, S. J., DESHWAL, A., LEE, W.-J., SHIN, Y., LEE, J., YOO, D., AND RYU, S. 2014. Shading language compiler implementation for a mobile ray tracing accelerator. In *Proceedings of ACM SIGGRAPH Asia 2014, Symposium on Mobile Graphics and Interactive Applications (MGIA)*, Article No. 2.

HWANG, S. J., LEE, J., SHIN, Y., LEE, W.-J., AND RYU, S. 2015. A mobile ray tracing engine with hybrid number representation. In *Proceedings of ACM SIGGRAPH Asia 2015, Symposium on Mobile Graphics and Interactive Applications (MGIA)*, Article No. 3.

JIN, B., IHM, I., CHANG, B., PARK, C., LEE, W.-J., AND JUNG, S. 2009. Selective and adaptive supersampling for real-time ray tracing. In *Proceedings of ACM High Performance Graphics (HPG) 2009*, 117–125.

JOHNSTONE, B., SOMMEFELDT, R., PETERSON, L., AND DAVIS, J. 2015. PowerVR graphics keynote: Latest developments and future plans. In *Imagination Developers Connection (IDC) 2015, Technical Talk*.

KAJIYA, J. 1986. The rendering equation. In *Proceedings of ACM SIGGRAPH 1986*, 143–150.

KIM, H.-Y., KIM, Y.-J., OH, J., AND KIM, L.-S. 2012. A reconfigurable SIMT processor for mobile ray tracing with contention reduction in shared memory. *IEEE Transactions on Circuits and Systems (TCS) 1*, 99, 1–13.

KIM, Y., SEO, W., KIM, Y., LIM, Y., NAH, J.-H., AND IHM, I. 2016. Adaptive undersampling for efficient mobile ray tracing. *The Visual Computer (Proceedings of Computer Graphics International 2016)*, 1–11.

KOPTA, D., SPJUT, J., DAVIS, A., AND BRUNVAND, E. 2010. Efficient MIMD architectures for high-performance ray tracing. In *Proceedings of the 28th IEEE International Conference on Computer Design (ICCD) 2010*, 9–16.

LEE, W.-J., WOO, S.-O., KWON, K.-T., SON, S.-J., MIN, K.-J., LEE, C.-H., JANG, K.-J., PARK, C.-M., JUNG, S.-Y., , AND LEE, S.-H. 2011. A scalable GPU architecture based on dynamically embedded reconfigurable processor. In *Proceedings of ACM High Performance Graphics 2011, Posters*.

LEE, W.-J., LEE, S., NAH, J.-H., KIM, J.-W., SHIN, Y., LEE, J., AND JUNG, S. 2012. SGRT: A scalable mobile GPU architecture based on ray tracing. In *Proceedings of ACM SIGGRAPH 2012, Talks*.

LEE, W.-J., SHIN, Y., LEE, J., ANDJAE HO NAH, J.-W. K., JUNG, S.-Y., LEE, S.-H., AND HAN, H.-S. P. T.-D. 2013. SGRT: A mobile GPU architecture for real-time ray tracing. In *Proceedings of ACM High Performance Graphics (HPG) 2013*, 109–119.

LEE, J., LEE, W.-J., SHIN, Y., HWANG, S. J., RYU, S., AND KIM, J. 2014. Two-AABB traversal for mobile real-time ray tracing. In *Proceedings of ACM SIGGRAPH Asia 2014, Symposium on Mobile Graphics and Interactive Applications (MGIA)*, Article No. 14.

LEE, W.-J., HWANG, S. J., SHIN, Y., YOO, J.-J., AND RYU, S. 2015. An efficient hybrid ray tracing and rasterizer architecture for mobile gpu. In *Proceedings of ACM SIGGRAPH Asia 2015, Symposium on Mobile Graphics and Interactive Applications (MGIA)*, Article No. 2.

LEE, W.-J., SHIN, Y., HWANG, S. J., KANG, S., YOO, J.-J., AND RYU, S. 2015. Reorder Buffer: An energy-efficient multithreading architecture for hardware mimd ray traversal. In *Proceedings of ACM High Performance Graphics (HPG) 2015*, 21–32.

MCCOMBE, J. 2014. New techniques made possible by PowerVR ray tracing hardware. In *Game Developer Conferecne (GDC) 2014, Technical Talk*.

MITCHELL, D. 1987. Generating antialiased images at low sampling densities. In *Proceedings of ACM SIGGRAPH 1987*, 65–72.

NAH, J.-H., KWON, H.-J., KIM, D.-S., JEONG, C.-H., PARK, J., HAN, T.-D., MANOCHA, D., AND PARK, W.-C. 2014. RayCore: A ray-tracing hardware architecture for mobile devices. *ACM Transactions on Graphics (TOG) 33*, 6, Article No. 162.

SAMSUNG, 2016. 14 nano meter technology . http://www.samsung.com/semiconductor/foundry/process-technology/14nm/.

SHIN, Y., LEE, W.-J., LEE, J., LEE, S.-H., RYU, S., AND KIM, J. 2013. Energy efficient data transmission for ray tracing on mobile computing platform. In *Proceedings of ACM SIGGRAPH Asia 2013, Symposium on Mobile Graphics and Interactive Applications (MGIA)*, Article No. 64.

SPJUT, J., KOPTA, D., BRUNVAND, E., AND DAVIS, A. 2012. A mobile accelerator architecture for ray tracing. In *Proceedings of 3rd Workshop on SoCs, Heterogeneous Architectures and Workloads (SHAW-3)*.

SYNOPSYS, 2016. Primetime PX: Signoff power analysis tool . https://www.synopsys.com/apps/support/training/primetimepx_fcd.html.

VAIDYANATHAN, K., SALVI, M., TOTH, R., FOLEY, T., AKENINE-MOLLER, T., NILSSON, J., MUNKBERG, J., HASSELGREN, J., SUGIHARA, M., CLARBERG, P., JANCZAK, T., AND LEFOHN, A. 2014. Coarse pixel shading. In *Proceedings of ACM High Performance Graphics (HPG) 2014*, 9–18.

WHITTED, T. 1980. An improved model for shaded display. *Communications of the ACM 23*, 6, 343–349.