Efficient Progressive Radiance Estimation Engine Architecture and Implementation for Progressive Photon Mapping

Ching-Chieh Chiu, Lan-Da Van, *Senior Member, IEEE*, and Yu-Shu Lin

***Abstract*— We propose a progressive radiance estimation engine (PREE) hardware architecture to accelerate the processing of the progressive photon mapping with satisfactory graphic quality. The presented PREE architecture consists of four progressive radiance estimation units (PREUs), approximate full task schedule-oriented hit-point update operation controller (AFTSO-HpUOC) and approximate data-independent schedule-oriented radiance evaluation controller (ADISO-REC). The PREUs accelerate the radiance estimation computation by a pipeline technique and share and configure the hardware resource for hit-point update operation and radiance evaluation. Through AFTSO-HpUOC and ADISO-REC, the data can be efficiently dispatched to achieve better parallelism and the data dependence can be alleviated within the four PREUs, respectively. The core area of the proposed PREE architecture implemented in TSMC 90-nm CMOS process is 1.78 mm^2. According to the post-layout simulation results, the implementation achieves 496.79 million hit-point update operations per second (MHpUO/s) and consumes 184 mW at 125 MHz for Cornell box with three balls.**

***Index Terms*— Hardware architecture, progressive photon mapping, progressive radiance estimation.**

I. Introduction

PHOTON mapping [1]–[9] is one of global illumination algorithms, which is capable of providing caustics, diffuse interreflection and subsurface scattering effects. The photon mapping is first proposed by Jensen [1] in 1996; however, the large memory requirement due to storing a full photon map and the biased result as pointed out in [3], [5], and [7] are incurred. Thus, the photon mapping [1] is further improved over the past years in a considerable number of studies [2]–[8]. The basic concept, computation and the latest developments of the photon mapping are introduced and demonstrated in [9]. In order to accelerate the final gathering, Havran *et al.* [2] modified the algorithm by reorganizing the computation in the reverse order. Hachisuka *et al.* [3], [4] introduced the progressive photon mapping (PPM) algorithm without storing a full photon map to render an image with arbitrary accuracy. Then the authors further proposed the stochastic progressive photon mapping (SPPM) algorithm [5], where the SPPM algorithm is used to process the effects of glossy reflections, motion blur and depth-of-field by distributed ray tracing. Kaplanyan and Dachsbacher [8] presented an adaptive progressive photon mapping (APPM) algorithm to adaptively determine the crucial parameters of the PPM [3], [4] such that the better graphic quality can be obtained from simulation results [8].

Generally, the hardware approaches [10]–[19] include general-purpose graphics processing unit (GPGPU)/graphics processing unit (GPU), central processing unit (CPU) with multi-core architecture, or application-specific integrated circuit (ASIC) hardware implementation approaches to accelerate the photon mapping related algorithms. To our best knowledge, Purcell *et al.* [10] firstly used GPU to accelerate the photon mapping. In recent years, applying GPU/GPGPU to implement the photon mapping related algorithms has been proposed in [14], [16], and [19]. Pedersen [16] proposed a uniform grid approach on GPUs for the PPM algorithm. Hachisuka [19] proposed a rendering system using GLSL shaders. Kim *et al.* [18] utilized heterogeneous computing resources composed of GPU and CPU to achieve a high rendering throughput. Singh *et al.* [11], [12], [15] proposed the photon map searching and shading accelerator architectures for the reverse pipelined photon mapping [2], [11], [12] and the high-throughput photon mapping [15]. Pan [17] explored the multi-accelerator architecture that is mainly composed of the tree search accelerator(s) (TSA) and the shader operation accelerator(s) (SOA) for the photon mapping.

In terms of algorithms, it is known that, from simulation results, SPPM [5] and APPM [8] have better graphic quality than PPM [3], [4]. Compared with PPM, the SPPM [5] algorithm designed for glossy reflections, motion blur and depth-of-field takes longer computation time because of the distributed ray tracing passes. Compared with PPM, the APPM algorithm [8] needs to calculate additional 6 values in [8] to obtain optimal radius at each iteration. Thus, PPM is selected for our hardware architecture and implementation research from the computation viewpoint. In terms of hardware architectures, there exist four challenges and problems as mentioned

Manuscript received August 15, 2017; revised November 24, 2017; accepted December 20, 2017. Date of publication January 23, 2018; date of current version July 3, 2018. This work was supported by the Ministry of Science and Technology under Grant MOST 106-2221-E-009-028-MY3 and Grant MOST 103-2221-E-009-099-MY3. This paper was recommended by Associate Editor G. Masera. *(Corresponding author: Lan-Da Van.)*

C.-C. Chiu and L.-D. Van are with the Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: ufo.keb@gmail.com; ldvan@cs.nctu.edu.tw).

Y.-S. Lin was with the Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2018.2789680

in Section III from mapping the PPM algorithm [3], [4] into a hardware such that the designers have to pay more efforts/time to mitigate these issues from the program coding viewpoint by GPU/CPU. Thus, an ASIC approach is a potential promising solution. To our best knowledge, considering high computing performance and satisfactory graphic quality, how to efficiently realize the progressive radiance estimation of PPM by an ASIC approach has not yet been addressed and explored. Thus, in this work, the novel progressive radiance estimation engine (PREE) hardware architecture consisting of four progressive radiance estimation units (PREUs) and two controllers including approximate full task schedule oriented hit-point update operation controller (AFTSO-HpUOC) and approximate data independent schedule oriented radiance evaluation controller (ADISO-REC) is proposed. Using these two controllers, the computation of each photon can be almost performed in parallel and the data dependence can be alleviated, respectively. The rest of the paper is organized as follows. In Section II, the background of photon mapping and PPM are described. In Section III, the challenges and problems via a hardware approach are issued. In Section IV, the proposed PREE and the corresponding embedded units are addressed. The chip layout implementation and comparison results are shown in Section V. Finally, a brief statement concludes this work.

II. Background

In this section, the photon mapping, progressive photon mapping (PPM) and terminologies of the expected value and variance of radiance estimation are briefly reviewed.

A. Photon Mapping

The photon mapping is one of global illumination methods that approximates to the real radiance by simulating the behavior of photons. The algorithm needs two-pass operations to accomplish the rendering. The first pass phase of the photon mapping [1] is photon tracing, where photons are emitted from a light source to intersect a surface of a scene and the intersected photon information is stored in the photon map. The second pass phase of the photon mapping [1] is ray tracing and photon gathering. Rays are shot from the camera to the scene until hitting the surface. The photon gathering accumulates the flux and determines the radiance of each pixel. It searches the fixed number of photons that is closest to the hit-point by the photon map. In order to reduce the increased number of photons, the distribution number of photons and the length of the photon path are instructed by Russian Roulette. However, in high resolution scenes, a large amount of stored photon information occupies a large amount of memory space to build the photon map. Storing and building the full photon map in the memory incurs a memory bottleneck. In next subsection, the PPM algorithm that resolves the memory space issue will be described.

B. Progressive Photon Mapping (PPM)

The PPM algorithm does not need to store the full photon map in the memory, and only stores the hit-points. PPM [3], [4] is a reverse and multi-pass algorithm, where the first pass phase is ray tracing and all subsequent passes are photon tracing. Furthermore, the three main operations of the progressive radiance estimation of PPM [3] include radius reduction, flux correction and radiance evaluation. All hit-points are produced before photons generation and are ready for refining by photons. Then a photon is released after refining the flux of affected hit-points. Thus, the full photon map is not demanded. In the ray tracing pass phase, a hit-point will be produced and stored in the memory when a ray intersects a surface. The hit-point data not only has surface information, but also includes the amount of accumulated photons, radius and accumulated flux. In the photon tracing pass phase, Russian Roulette is used to determine the photon path in which each intersected photon acquires hit-points. The radius and flux of the hit-point at the surface location x with the incident ray direction $\vec{w}$ surrounding the intersected photons are reduced and corrected by Eq. (1) and Eq. (2), respectively [3], [4].

$$\widehat{R}(x) = R(x)\sqrt{\frac{N(x)+\alpha M(x)}{N(x)+M(x)}} \tag{1}$$

$$\begin{aligned}\tau_{\widehat{N}}(x,\vec{w}) &= (\tau_N(x,\vec{w})+\tau_M(x,\vec{w}))\frac{\pi \widehat{R}^2(x)}{\pi R^2(x)}\\ &= \tau_{N+M}(x,\vec{w})\frac{N(x)+\alpha M(x)}{N(x)+M(x)}\end{aligned} \tag{2}$$

where $R(x)$, $N(x)$, $M(x)$, α, and $\tau_N(x,\vec{w})$ denote the radius of a hit-point, the number of photons that are within the radius of a hit-point, the number of photons that are found in new photon tracing pass phase and within the radius of a hit-point, the parameter for controlling the photon number, and the total unnormalized flux received from photons, respectively. Note that, πR^2 denotes the locally flat surface with the radius R in which the photons are located [3]. Instead of requesting all new photons, α, between 0 and 1, is used to control how many photons are needed. The $\tau_N(x,\vec{w})$ is defined in Eq. (3) [3], [4].

$$\tau_N(x,\vec{w}) = \sum_{p=1}^{N(x)} f_r(x,\vec{w},\vec{w}_p)\phi_p(x_p,\vec{w}_p) \tag{3}$$

where $f_r(x,\vec{w},\vec{w}_p)$ and $\phi_p(x_p,\vec{w}_p)$ denote the bidirectional reflectance distribution function (BRDF) and the unnormalized flux induced by the photon p at the surface location x_p with the incident photon direction $\vec{w}_p$, respectively. After the photon tracing pass phase, the fluxes of hit-points are used to estimate the radiance of the corresponding pixel by Eq. (4) [3], [4].

$$L(x,\vec{w}) = \frac{1}{\pi R^2(x)}\frac{\tau(x,\vec{w})}{N_{emitted}} \tag{4}$$

where $N_{emitted}$ denotes the total number of emitted photons after the photon tracing pass phase. Another advantage of the PPM algorithm is that a photon can simultaneously update all related hit-points through parallel computing. By contrast, in the gathering step of the traditional photon mapping, each single hit-point is updated by a number of photons. To avoid concurrent writing, a hit-point is sequentially updated by each related photon.

C. Expected Value and Variance of Radiance Estimation

In addition to the large memory space requirement, another disadvantage of the photon mapping is that it cannot achieve low expected value/low bias and low variance/low noise at the same time. That means the photon mapping is a biased method as pointed out in [5] and [7]. On the contrary, PPM solves this issue by reducing the radius at each radiance estimate step. Knaus and Zwicker [7] systematically described the expected value and variance of radiance estimation, and how PPM decreases these two factors. Their work [7] proves that while the samples $N(x)$ is infinite, the variance and expected value of average error $\overline{\varepsilon}_N$ will approach zero. Further, the expected value of a pixel value using $N(x)$ samples is derived as follows [7].

$$E[\overline{c}_N] = c + E[\frac{W}{p_e}]E[\overline{\varepsilon}_N] \tag{5}$$

where, c is the exact pixel value, $\overline{c}_N$ is the estimate using $N(x)$ samples, p_e is a certain probability density, W is the contribution of eye paths, and $E[\overline{\varepsilon}_N]$ is the expected value of average error. Herein, $E[\overline{c}_N]$ is regarded as the theoretical graphic quality in this paper. Eq. (5) shows that if $E[\overline{\varepsilon}_N]$ approaches zero, $E[\overline{c}_N]$ is equal to c.

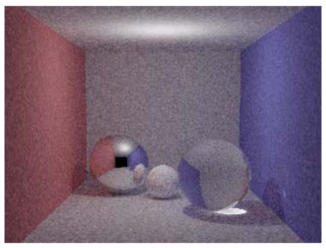
III. Challenges and Problems

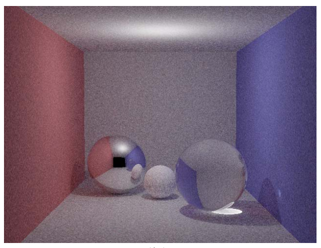
In this section, from computation profiling and hardware implementation viewpoints, the challenges and problems via a hardware approach are addressed.

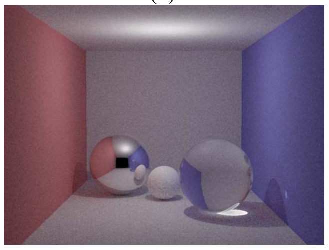
A. Progressive Radiance Estimation Computation

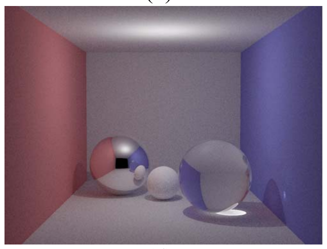
From the viewpoint of computation, except the radius reduction, flux correction, and radiance evaluation [3], actually, the distance calculation between a neighboring hit-point and a photon, comparison, photon count accumulation are needed. Thus, the computation including the distance calculation, comparison, radius reduction, flux correction, and photon count accumulation is named as a hit-point update operation for profiling analysis and hardware design in this paper. The profiling results in Fig. 1 and Fig. 2 are obtained by applying/modifying the source code [4] using Visual C++ with single thread in the environment of Intel CPU i7-2600 3.4GHz. Note that the image with ppm format is converted to that with png format throughout this paper. Figs. 1(a)~(d) show the Cornell box scenes with 3 balls at four different resolutions. For satisfactory graphic quality, the number of emitted photons is 10 times of the resolution. Therefore, 0.768 million photons in 320 × 240 resolution are emitted such that the number of hit-point update operations is 78.12 million according to the accumulated result of the PPM simulation. Fig. 1(e) shows the bar chart in terms of the number of hit-point update operations for each resolution. When the resolution is up to 1600 × 1200 with 19.2 million photons, the number of hit-point update operations is up to 1934.61 million. Fig. 2 profiles the quantitative analysis results of the Cornell box scene with 3 balls by emitting 5 million photons with resolution of 640 × 480.

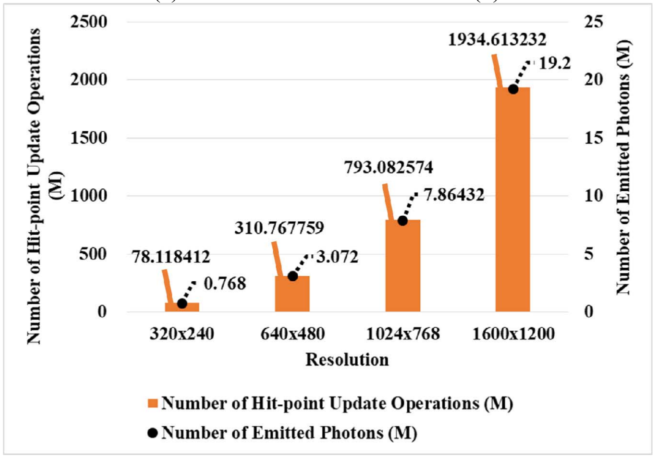
The analysis results in Fig. 2(a) indicate that the hit-point update operation has the largest amount of execution time


(a)


(b)


(c)


(d)



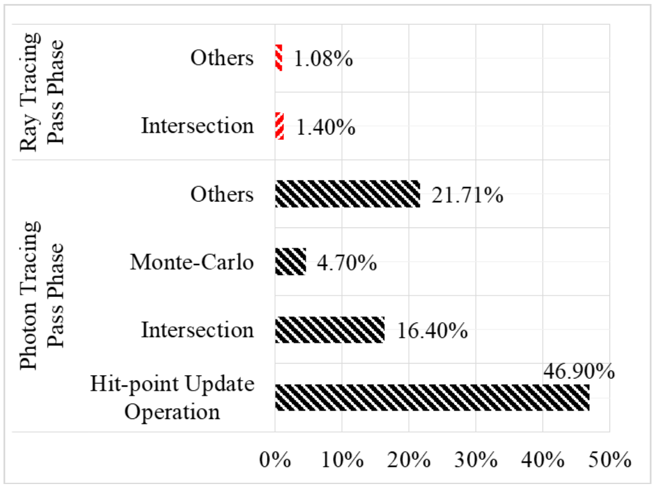
(e)

Fig. 1. Four different resolution qualities. (a) 320 × 240, (b) 640 × 480, (c) 1024 × 768, (d) 1600 × 1200, and (e) hit-point update operations.

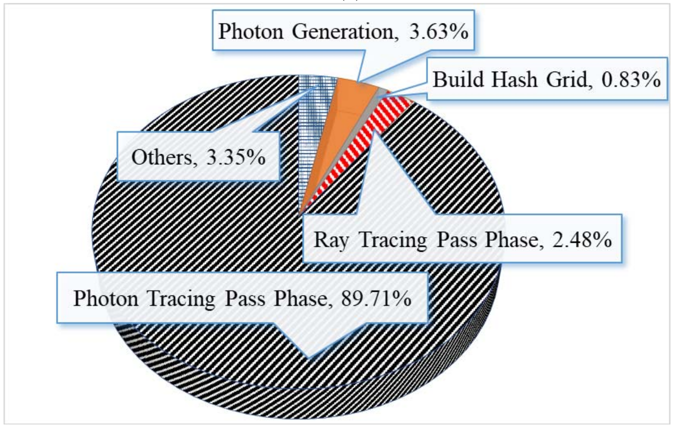
(i.e., 46.90%), where it is noted that the radiance-evaluation execution time is accounted in others (i.e., 3.35%) in Fig. 2(b). Thus, it is expected that the progressive radiance estimation including the hit-point update operation and radiance-evaluation operation can be accelerated by the hardware such that the overall performance of PPM might be reinforced. Note that in case a scene covers more triangles, the execution time of intersections will be largely increased in profiling. In this case, several famous tree methods can be used to mitigate this issue. In this paper, since how to design an efficient tree method is not our focus and beyond the scope of this paper, the accelerated progressive radiance estimation hardware architecture with satisfactory graphic quality is our concentration.

B. Hardware Resource Sharing and Configuration

After profiling PPM in Section III.A, it is observed that the hit-point update operation and radiance evaluation will not be concurrently performed from a hardware viewpoint. Thus, how to share and configure the hardware resource for these two main operations to reduce the cost will need attention.



(a)



(b)

Fig. 2. Quantitative profiling results of PPM. (a) Percentage distribution, and (b) breakdown of software execution time of PPM.

C. Approximate Full Task Scheduling Challenge

The larger resolution leads to the larger hit-point update operations in Fig. 1(e). Although PPM can use a photon to simultaneously update all the related hit-points through parallel computing, how to efficiently parallelize hit-point update operations via an ASIC approach has not yet been discussed. Intuitively, we can arrange these operations belonging to one photon to one unit. However, the number of hit-point update operations for different photons are usually different. Thus, other units will wait for the unit which has the longest execution time and the execution time will be increased. Another way to deal with this parallel problem is to distribute a set of hit-point update operations affected by this photon among parallel units. However, the number of hit-point update operations in a set will not be always full for distribution among the units. Hence, it will be the important requirement to distribute hit-point update operations of two or more sets to fill up the parallel units.

D. Data Dependence Problem

Since the ray tracing algorithm [20], [21] recursively traces each pixel by reflection and refraction, it is conceivable that one or more hit-points could be generated by tracing a pixel. For example, in Fig. 3, the hit-point memory of four PREUs

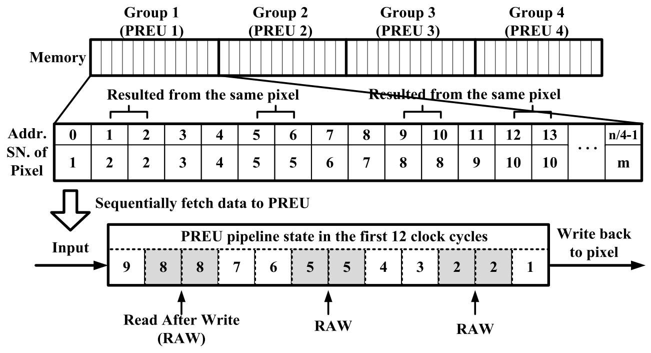

Group 1 (PREU 1)
Group 2 (PREU 2)
Group 3 (PREU 3)
Group 4 (PREU 4)
Memory
Resulted from the same pixel
Resulted from the same pixel
Addr. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... n/4-1
SN. of Pixel 1 2 2 3 4 5 5 6 7 8 8 9 10 10 m
Sequentially fetch data to PREU
Input
PREU pipeline state in the first 12 clock cycles
9 8 8 7 6 5 5 4 3 2 2 1
Write back to pixel
Read After Write (RAW)
RAW
RAW

Fig. 3. Data dependence resulting from sequentially allocated hit-points.

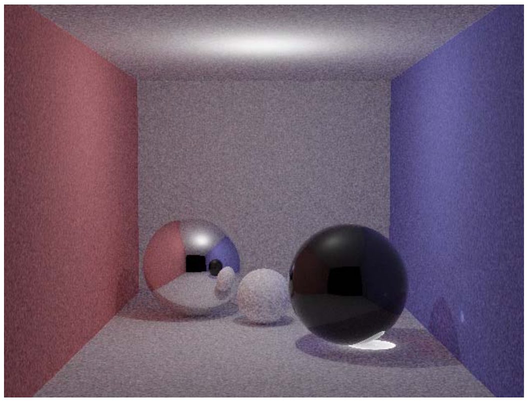


Fig. 4. Simulation result due to data dependence with the sequential allocation.

has four groups. In terms of function simulation, the hit-points resulting from the same pixel are sequentially allocated to adjacent addresses. This sequential allocation property can easily lead to read after write (RAW) data hazard as shown in Fig. 3. The serial number (SN) in the memory means which pixel the hit-point corresponds to, where the corresponding hit-point data is stored in the same memory space. In Fig. 4, RAW data hazard incurs incorrect rendered pixel values such that the glass ball's reflection effect directly overwrites its refraction effect. Based on the source code [4], the emulation program can be developed to simulate this effect. In this case, while two or more hit-points are produced, it leads to data dependence problems in multi-core or pipelined structures as ours.

IV. Proposed Progressive Radiance Estimation Engine Architecture

According to the motivation discussion in Section III.A, the radiance estimation occupies the largest amount of the execution time of the photon tracing pass phase for Cornell box with 3 balls. Therefore, the progressive radiance estimation engine (PREE) in Fig. 5 is proposed to accelerate the radiance estimation computation of PPM [3]. In order to speed up the PREE's performance, four progressive radiance estimation units (PREUs) are adopted such that the update operations of four hit-points can be computed in parallel. Due to the limited hardware resource, without loss of generality, four PREUs can be regarded as one case study and the methodology can be extended to more than four PREUs hardware. Meanwhile,

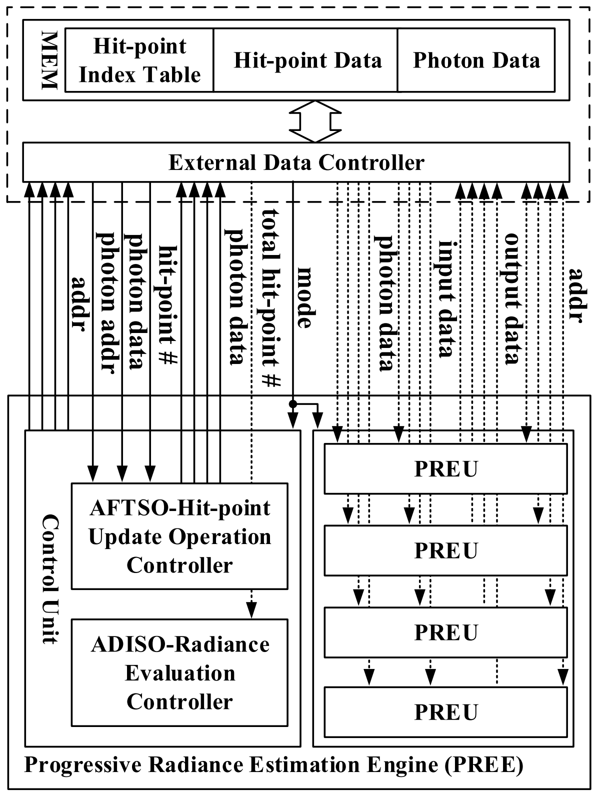


Fig. 5. System block diagram of PREE in the solid-line box.

we will address the utilization rate with more than four PREUs in Section IV.B. Furthermore, the control unit includes AFTSO-HpUOC controller and ADISO-REC controller to schedule photon data and hit-point data for parallel computing and to alleviate data dependence, respectively. The representative (i.e., not entire for simplicity) signals between PREE and the external data controller are shown in Fig. 5, where the dash-line box is not implemented in this paper. The main data flow in Fig. 5 is described as follows. During the hit-point update operation phase, the mode signal selects the fixed-point AFTSO-HpUOC controller to generate the four reference addresses (details will be addressed in Section IV.B), and then the photon data and hit-pint data with these four reference addresses are sent to four 32-bit floating-point PREUs to calculate the four hit-points' fluxes.

During the radiance evaluation phase, the mode signal selects the fixed-point ADISO-REC controller to generate the four leaping addresses (details will be addressed in Section IV.C), and then the fluxes of hit-points with these four leaping addresses are sent to four PREUs to calculate four radiance values. Each PREU uses the leaping address to avoid sequentially accessing the memory such that the data dependence can be alleviated. In this section, we will describe the architecture of the PREE and PREU in detail.

A. Progressive Radiance Estimation Unit (PREU)

Considering the issue in Section III.B, one configuration is hit-point update operation and another is radiance evaluation, where two computations of the progressive radiance estimation will not be performed at the same time. Through the hardware resource sharing and configuration, these two computations can be performed in one configurable data path. The detailed data path operations for above two configurations are described in the following sub-sections.

1) Configurable Data Path of Hit-Point Update Operation in the PREU: Assume one hit-point data (H) and one photon data (P) as inputs in this data path. Hit-point data includes position (x^H_{pos}, y^H_{pos}, z^H_{pos}), color (r^H_{color}, g^H_{color}, b^H_{color}), accumulated flux $\tau_N(x,\vec{w}) = (\tau^H_{N,r}, \tau^H_{N,g}, \tau^H_{N,b})$, radius square, and accumulated number of photons for one hit-point N, which is equal to the variable $N(x)$ of Eq. (1). Photon data includes position (x^P_{pos}, y^P_{pos}, z^P_{pos}), and flux $\phi_p(x_p, \vec{w}_p) = (\phi^P_r, \phi^P_g, \phi^P_b)$. There is a refined hit-point data including flux $\tau_{\hat{N}}(x,\vec{w}) = (\tau^H_{\hat{N},r}, \tau^H_{\hat{N},g}, \tau^H_{\hat{N},b})$, radius square $\hat{R}^2$, and updated accumulated number of photons for one hit-point $\hat{N}$ as outputs. Before determining whether the updated radius and flux are selected, the distance square between a hit-point and a photon has to be calculated in (6).

$$D^2_{HP} = (x^H_{pos} - x^P_{pos})^2 + (y^H_{pos} - y^P_{pos})^2 + (z^H_{pos} - z^P_{pos})^2 \quad (6)$$

where subscript *HP* denotes the hit-point data (H) and photon data (P). Herein, in order to avoid the square-root computation, Eq. (1) can be recast as follows.

$$\widehat{R}^2(x) = R^2(x)\frac{N(x)+\alpha M(x)}{N(x)+M(x)} = \widehat{R}^2 = R^2\frac{N+\alpha M}{N+M} \quad (7)$$

Then we can determine whether the photon is inside the radius of one hit-point through the comparison with $D^2_{HP} \le R^2$. Using the notations in this section, Eq. (2) can be recast as

$$\begin{aligned}\tau_{\widehat{N}}(x,\vec{w}) &= \tau_{\widehat{N}} = (\tau^H_{\widehat{N},r}, \tau^H_{\widehat{N},g}, \tau^H_{\widehat{N},b}) \\ &= (\tau_{N+M,r}\frac{N+\alpha M}{N+M}, \tau_{N+M,g}\frac{N+\alpha M}{N+M}, \tau_{N+M,b} \\ &\quad \times\frac{N+\alpha M}{N+M}) \end{aligned} \quad (8)$$

Thus, an accelerated hit-point update operation data path that performs distance calculation, comparison, radius reduction, flux correction, photon count accumulation is shown in Fig. 6 with $M = 1$. This unit consists of three blocks. Block 1 is used to calculate the distance square between a hit-point and a photon in Eq. (6). Block 2 is used to calculate and update the reduced radius square $\widehat{R}^2$ in Eq. (7), where the result depends on the accumulated number of photons. Block 3 is used to update the flux of the hit-point $\tau_{\widehat{N}}$ in Eq. (8), which was affected by the input photon. In Eq. (8), the values of three variables can be obtained by the product of correction factor $(N+\alpha M)/(N+M)$ and the sum of τ_N and τ_M, respectively, where $M = 1$ in Fig. 6. τ_N is input, and τ_M can be obtained by the product of three sets of input data [4]: the hit-point color consisting of r^H_{color}, g^H_{color} and b^H_{color}, the photon flux consisting of ϕ^P_r, ϕ^P_g and ϕ^P_b and $1/\pi$. At last, the updated accumulated number of photons, radius and flux will be output and be written back to the memory if the input photon is within the radius of the hit-point. The unit in Fig. 6 consists of the following 32-bit floating-point arithmetic components in IEEE 754 format presented in Table 1: divider, multiplier, squarer,

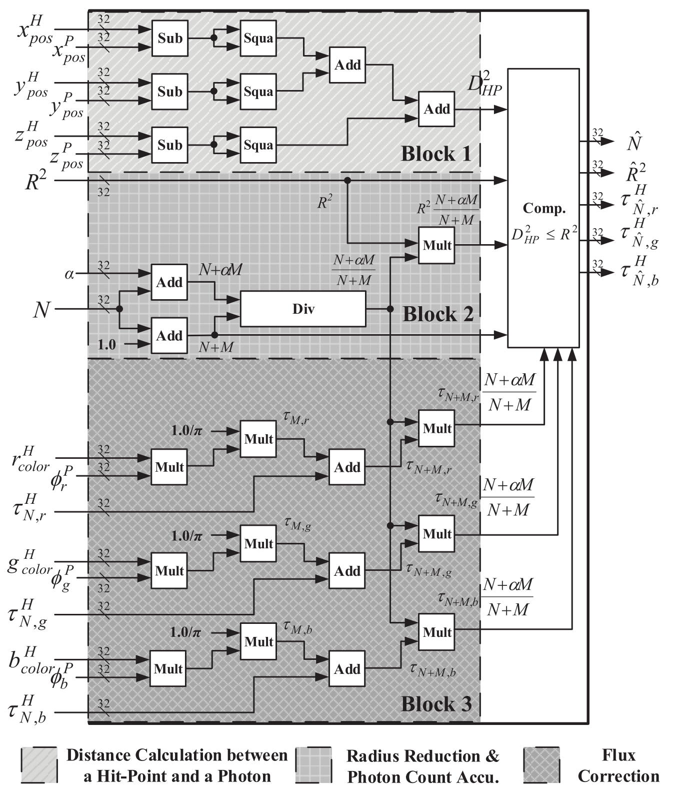

Sub
Squa
Add
Block 1
Comp.
Block 2
Block 3
Div
Mult
Distance Calculation between a Hit-Point and a Photon
Radius Reduction & Photon Count Accu.
Flux Correction

Fig. 6. Configurable data path for the hit-point update operation in the PREU.

TABLE I
ARITHMETIC COMPONENTS OF ONE PREU

Arithmetic Component	Synopsys DesignWare Lib.	Format	Rounding Type	# of Pip. Stages
Divider	DW_fp_div	32-bit floating point in IEEE 754	IEEE round to nearest	4
Multiplier	DW_fp_mult			2
Squarer	DW_fp_square			2
Add	DW_fp_add			2
Subtractor	DW_fp_sub			2
Comparator	DW_fp_cmp		NA	0

adder, subtractor, and comparator. The first five arithmetic components use IEEE round to nearest. The divider has four pipeline stages, multiplier, squarer, adder as well as subtractor have two pipeline stages, and the comparator is a non-pipelined combinational logic.

2) Configurable Data Path of Radiance Evaluation in the PREU and Hardware Sharing: When it comes to the final stage of the PPM algorithm [3], [4], the hardware resource can be configured to compute final pixel color in Eq. (4). The input hit-point data (H) includes accumulated flux ($\tau^H_{\hat{N},r}$, $\tau^H_{\hat{N},g}$, $\tau^H_{\hat{N},b}$), radius square, the corresponding pixel data (L_r, L_g, L_b) of the hit-point and the number of emitted photons $N_{emitted}$. There is a refined pixel data ($\widehat{L}_r$,$\widehat{L}_g$,$\widehat{L}_b$) as outputs in this data path. Using the notations in this section, Eq. (4) can be recursively realized and rewritten as

$$L(x,\vec{w}) = (\widehat{L}_r, \widehat{L}_g, \widehat{L}_b) = (L_r, L_g, L_b) + \frac{1}{\pi R^2 N_{emitted}}(\tau^H_{\widehat{N},r}, \tau^H_{\widehat{N},g}, \tau^H_{\widehat{N},b}) \quad (9)$$

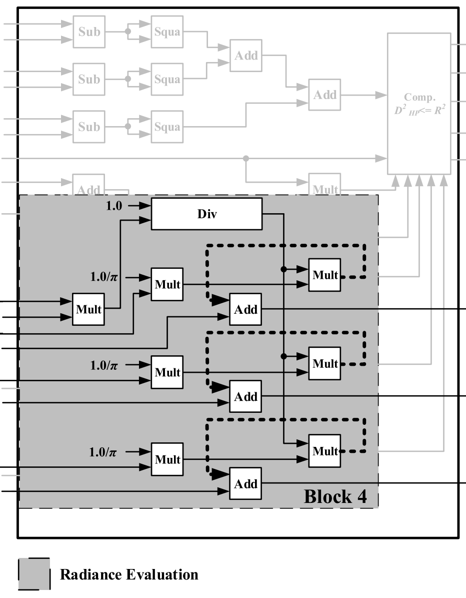


Fig. 7. Configurable data path for the radiance evaluation in the PREU.

The corresponding data path architecture of Eq. (9) is shown in Block 4 of Fig. 7. Observing Block 2, Block 3 in Fig. 6 and Block 4 in Fig. 7, the data path can be shared since the radius reduction as well as flux correction and radiance evaluation will be not computed at the same time. Therefore, the PREU can be configured two modes to achieve either the hit-point update operation or the radiance evaluation by a mode signal in Fig. 5. For simplicity, the mode signal is not shown in Fig. 6 and Fig. 7. In this work, the hit-point update operation data path can be configured to ten-pipeline stages and the radiance evaluation data path is configured to twelve-pipeline stages. The data path of radiance evaluation increases extra two pipeline stages because it feeds back the data to the previous adder stage, where the feedback path is depicted with dotted lines in Fig. 7.

Totally, the shared data path of a PREU saves seven multipliers, three adders and one divider compared to the separated hardware resources. In addition, since the hit-point address and the pixel index are used to indicate the address of the output data in memory, these two signals are kept in PREUs. Herein, for simplicity, the signals are not shown in Fig. 6 and Fig. 7, respectively,

B. Approximate Full Task Schedule Oriented Hit-Point Update Operation Controller (AFTSO-HpUOC)

Considering the challenge in Section III.C, in order to increase the utilization of four PREUs, AFTSO-HpUOC as shown in Fig. 8(a) is designed to efficiently schedule at most four different photons within four parallel computed PREUs. As can be seen in Fig. 8(a), the inputs of AFTSO-HpUOC are the photon data (32 bits $\times$ 6), the corresponding photon address (32 bits, abbreviated as *p_addr*), and the number of hit-points (32 bits) that are affected by this photon, where the latter one is denoted as the notation $N^P_{hit-point}$ in Fig. 8(a). The outputs of AFTSO-HpUOC are the *reference addresses* (abbreviated as *ref_addr$_i$*) of the hit-point and the respective photon data. The *ref_addr* is obtained by adding the *p_addr*

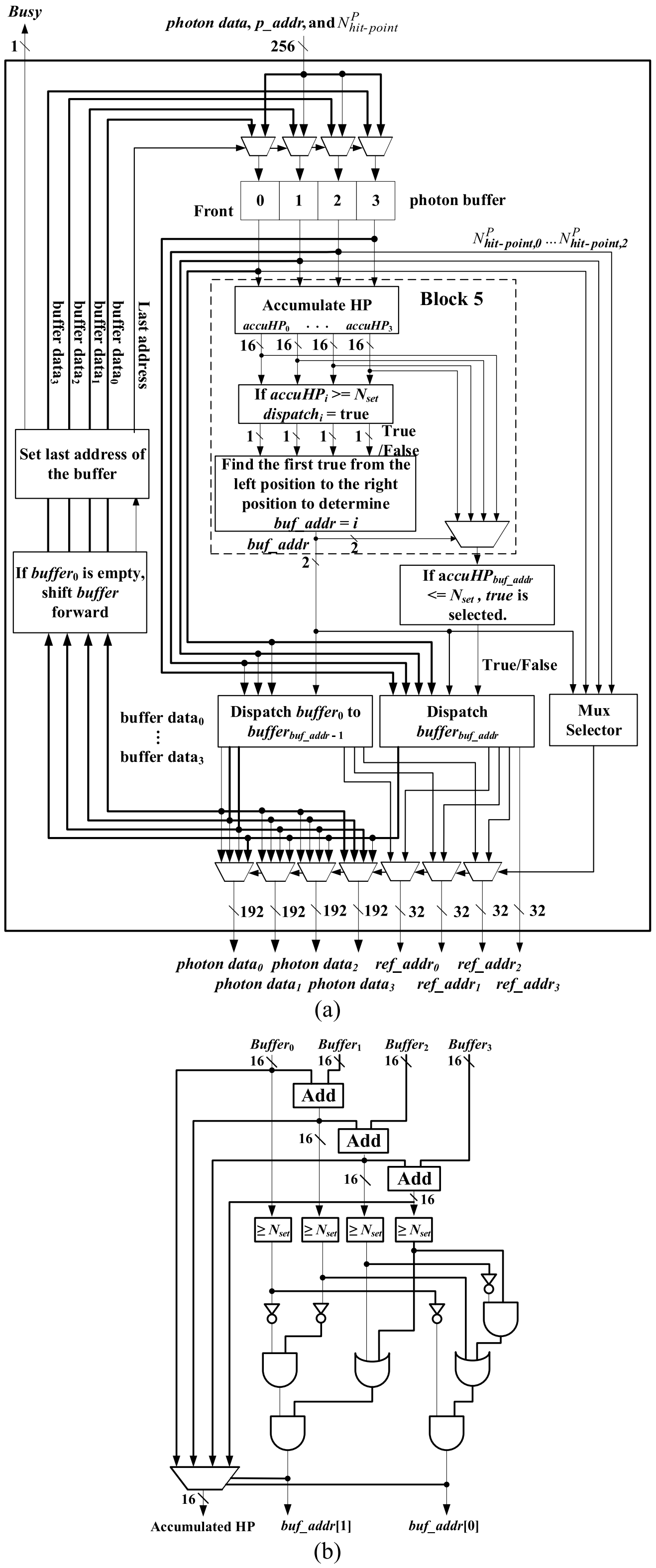

Busy
photon data, p_addr, and $N^P_{hit-point}$
photon buffer
Front
Accumulate HP
Block 5
If $accuHP_i >= N_{set}$ $dispatch_i$ = true
Find the first true from the left position to the right position to determine buf_addr = i
Set last address of the buffer
If $buffer_0$ is empty, shift buffer forward
If $accuHP_{buf\_addr}$ <= N_{set}, true is selected.
Dispatch $buffer_0$ to $buffer_{buf\_addr-1}$
Dispatch $buffer_{buf\_addr}$
Mux Selector
photon data₀ photon data₂ ref_addr₀ ref_addr₂
photon data₁ photon data₃ ref_addr₁ ref_addr₃
(a)
Add
Accumulated HP
buf_addr[1]
buf_addr[0]
(b)

Fig. 8. (a) Simplified block diagram of the AFTSO-HpUOC controller, and (b) the conceptual logic diagram of Bock 5 of (a).

and $N^P_{hit-point}$. For example, if $p\_addr = 100$, $N^P_{hit-point} = 3$, then $ref\_addr_0 = 100 + 3–1 = 102$, $ref\_addr_1 = 100 + 3–2 = 101$, $ref\_addr_2 = 100 + 3–3 = 100$. The ref_addr_i does not directly point to the hit-point data of the memory; instead, it points an address of an index table which records the addresses of the hit-points' data in the memory in Fig. 5. The ref_addr_i arrangement scheme of AFTSO-HpUOC is addressed as follows. AFTSO-HpUOC uses a pointer named last, to indicate which buffer to save when new data is coming. When the buffer is not empty, AFTSO-HpUOC will accumulate each buffer's hit-point number. That means $accuHP_0 = buffer_0$, $accuHP_1 = accuHP_0 + buffer_1$, $accuHP_2 = accuHP_1 + buffer_2$, and $accuHP_3 = accuHP_2 + buffer_3$. Next, it will check each *accuHP* whether is greater than or equal to N_{set} or not, where N_{set} denotes the number of PREUs. If yes, set $dispatch_i$ = true, where $0 \le i < N_{set}$. Then AFTSO-HpUOC will find the first true from $dispatch_0$ to $dispatch_{N_{set}-1}$ and set its address as *buf_addr*. The corresponding conceptual logic diagram of Block 5 in Fig. 8(a) is depicted in Fig. 8(b). Next, it will check whether $accuHP_{buf\_addr}$ is less than or equal to N_{set}. If true, it will dispatch all hit-points of $buffer_{buf\_addr}$; otherwise, it will only dispatch partial hit-points of $buffer_{buf\_addr}$. At the same time, it will dispatch all hit-points in the buffer where the address number is less than *buf_addr*. The outputs of the left side dispatch and the right side dispatch consist of buffer data and the resulting *ref_addr*. According to the *buf_addr* and $N^P_{hit-point,0}$ to $N^P_{hit-point,2}$, Mux Selector generates the select signals to determine the photon data and ref_addr_0 to ref_addr_2 from either the left side dispatch or the right side dispatch operations. After that, it shifts all of the undispatched photons to the front of the buffer. Finally, it sets the last address of the buffer and checks whether the whole buffer has been occupied or not. If yes, set *busy* up until the buffer is not full. According to the software simulation results, AFTSO-HpUOC can efficiently dispatch the data, especially in a large number of parallel PREUs. Note that when increasing the number of PREUs to 8 and 16, AFTSO-HpUOC still just needs one four-entry buffer to dispatch data to PREUs. Table 2 shows the utilization rate with 4, 8 and 16 parallel PREUs. As can be seen in Table 2, the utilization rate with AFTSO-HpUOC can achieve 99.99%. The reason is that the coming data is blocked when the buffer is full and the total number of hit-points in the buffer is equal to N_{set} such that AFTSO-HpUOC cannot achieve 100% utilization.

TABLE II
UTILIZATION OF PREUs

PREU Number	Without AFTSO-HpUOC	With AFTSO-HpUOC	Utilization Improvement
4	95.75%	99.99%	4.24%
8	90.53%	99.99%	9.46%
16	81.75%	99.99%	18.24%

C. Approximate Data Independent Schedule Oriented Radiance Evaluation Controller (ADISO-REC)

The data dependence problem of the hit-points caused by the sequential allocation property mentioned in Section III.D leads to the problem that the same pixel is concurrently computed among PREUs. The data dependence problem can be alleviated by discontinuous address fetching. The flow chart

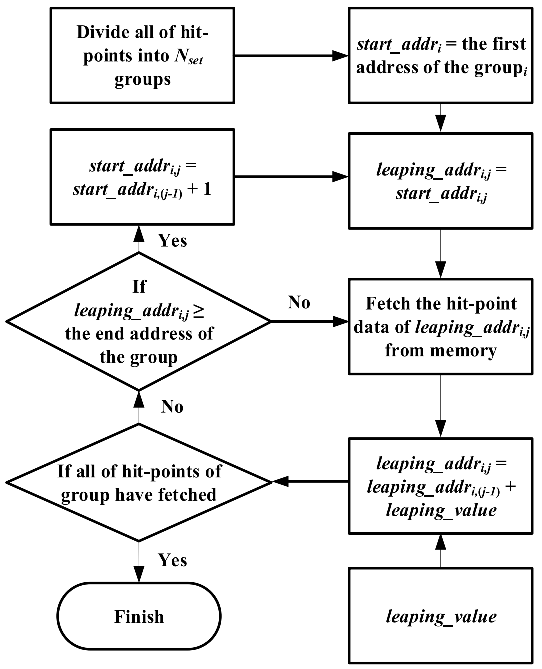

Divide all of hit-points into N_{set} groups
$start\_addr_i$ = the first address of the $group_i$
$start\_addr_{i,j}$ = $start\_addr_{i,(j-1)}$ + 1
$leaping\_addr_{i,j}$ = $start\_addr_{i,j}$
Yes
If $leaping\_addr_{i,j} \geq$ the end address of the group
No
Fetch the hit-point data of $leaping\_addr_{i,j}$ from memory
No
If all of hit-points of group have fetched
$leaping\_addr_{i,j}$ = $leaping\_addr_{i,(j-1)}$ + $leaping\_value$
Yes
Finish
$leaping\_value$

Fig. 9. Flow chart of generating the discontinuous leaping addresses.

provided in Fig. 9 is used to obtain the leaping addresses for each group. The initialization of the computation flow is to divide the hit-point memory into N_{set} groups where each group corresponds to one PREU, sets *start_address$_i$* (abbreviated as *start_addr$_i$*) and calculates the *leaping_value*, where the subscript variable $0 \leq i < N_{set}$ denotes the respective group. The *leaping_value* ensures that the hit-points computed in twelve-pipeline stages are fetched by accessing discontinuous memory addresses, and can be calculated by the following equation.

$$leaping\_value = \left\lfloor \frac{N_{hit-point}}{2^{\lceil \log_2 (N_{set} \times N_{pip\_stage}) \rceil}} \right\rfloor \quad (10)$$

where the notations $N_{hit-point}$ and N_{pip_stgage} denote the total number of stored hit-points and the number of pipeline stages, respectively. The *start_addr$_i$* is used to determine the beginning address of the leaping address of the *group$_i$*. The first address of *start_addr$_i$* can be calculated by $size_{group} \times i$, where $size_{group}$ is the size of group. The j-th address of *start_addr$_i$* (abbreviated as *start_addr$_{i,j}$*) can be represented by the following equation.

$$\begin{aligned} &start\_addr_{i,j} \\ &= size_{group} \times i + \left\lfloor \frac{j}{\lceil size_{group}/leaping\_value \rceil} \right\rfloor \\ &= \begin{cases} size_{group} \times i, & \text{if } j == 0 \\ start\_addr_{i,j-1} + 1, & \text{else if } \left\lfloor \frac{j}{\lceil size_{group}/leaping\_value \rceil} \right\rfloor > \left\lfloor \frac{j-1}{\lceil size_{group}/leaping\_value \rceil} \right\rfloor \\ start\_addr_{i,j-1}, & \text{otherwise} \end{cases} \end{aligned} \quad (11)$$

where j is from 0 to $size_{group} - 1$.

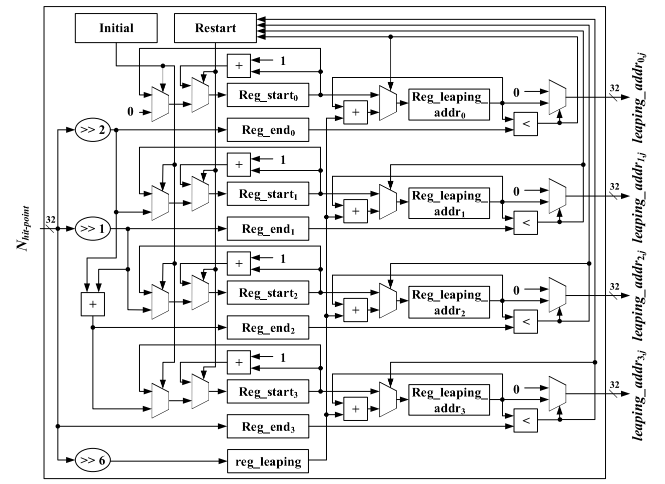


Fig. 10. Block diagram of the ADISO-REC controller.

The main computation of the flow chart is to repetitively calculate the *leaping_address$_{i,j}$* (abbreviated as *leaping_addr$_{i,j}$*) with *start_addr$_{i,j}$* and *leaping_value* until all of hit-point addresses have been generated. The first address and the second address of the *leaping_addr$_i$* can be represented as the $leaping\_addr_{i,0} = start\_addr_{i,0}$ and $leaping\_addr_{i,1} = leaping\_addr_{i,0} + leaping\_value$, respectively.

According to the flow chart, the j-th *leaping_addr$_i$* can be generally represented in the following equation.

$$\begin{aligned} &leaping\_addr_{i,j} \\ &= \begin{cases} start\_addr_{i,j}, & \text{if } leaping\_addr_{i,j-1} + leaping\_value \geq size_{group} \times (i+1) \\ leaping\_addr_{i,j-1} + leaping\_value, & \text{otherwise} \end{cases} \end{aligned} \quad (12)$$

The known $leaping\_addr_{i,j-1}$ can be used to simplify the calculation of the condition $\left\lfloor \frac{j}{\lceil size_{group}/leaping\_value \rceil} \right\rfloor > \left\lfloor \frac{j-1}{\lceil size_{group}/leaping\_value \rceil} \right\rfloor$ in Eq. (11), because this condition will be only satisfied when $leaping\_addr_{i,j-1}$ + *leaping_value* is larger than the end address of the i-th group. Thus, Eq. (11) can be recast as follows.

$$\begin{aligned} &start\_addr_{i,j} \\ &= \begin{cases} size_{group} \times i, & \text{if } j == 0 \\ start\_addr_{i,j-1} + 1, & \text{else if } leaping\_addr_{i,j-1} + leaping\_value \geq size_{group} \times (i+1) \\ start\_addr_{i,j-1}, & \text{otherwise} \end{cases} \end{aligned} \quad (13)$$

The corresponding proposed ADISO-REC block diagram of the flow chart in Fig. 9 and Eqs. (10), (12) and (13) with $N_{set} = 4$ and $N_{pip\_stage} = 12$ is shown in Fig. 10, where the input and outputs are $N_{hit-point}$ and four leaping addresses (i.e., $leaping\_addr_{0,j}$ to $leaping\_addr_{3,j}$), respectively. The four leaping addresses correspond to four PREUs, respectively,

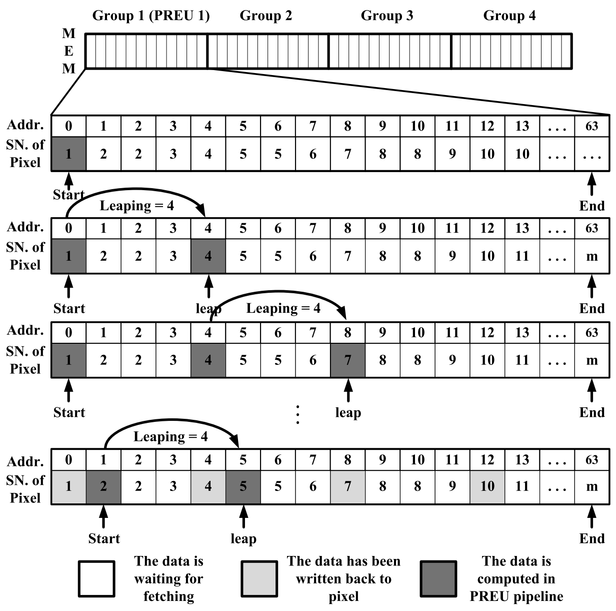

Group 1 (PREU 1)
Group 2
Group 3
Group 4
MEM
Addr.
SN. of Pixel
Start
Leaping = 4
leap
End
The data is waiting for fetching
The data has been written back to pixel
The data is computed in PREU pipeline

Fig. 11. An example of accessing the memory using ADISO-REC.

and each output can generate discontinuous addresses in turn. Since $N_{set} = 4$, $N_{pip\_stage} = 12$, the denominator of Eq. (10) is $2^{\lceil \log_2(N_{set} \times N_{pip\_stage}) \rceil} = 2^{\lceil \log_2(4\times12) \rceil} = 2^6$. That means the leaping value can be obtained by dividing $N_{hit-point}$ by 2^6, where 2^6 can be implemented by shifting right 6 bits (i.e., $\gg$6 in Fig. 10). The $start\_addr_{i,j}$ in Fig. 9 and Eq. (13) can be saved to the four registers named as Reg_start_0 to Reg_start_3 in Fig. 10, respectively. The end address of the group in Fig. 9 can be saved to the four registers named as Reg_end_0 to Reg_end_3 in Fig. 10, respectively. Herein, $Reg\_end_0 = N_{hit-point}/4 = N_{hit-point} \gg 2$, $Reg\_end_1 = N_{hit-point} \times 2/4 = N_{hit-point} \gg 1$, $Reg\_end_2 = N_{hit-point} \times 3/4 = N_{hit-point} \times (1/4 + 2/4) = N_{hit-point} \gg 2 + N_{hit-point} \gg 1$, $Reg\_end_3 = N_{hit-point}$.

Finally, the $leaping\_addr_{i,j}$ in Fig. 9 and Eq. (12) can be saved to the four registers named as $Reg\_leaping\_addr_0$ to $Reg\_leaping\_addr_3$, respectively, in Fig. 10. These four registers and four 32-bit fixed-point adders are used to perform the calculation in Eq. (12). Fig. 11 shows an example of how ADISO-REC fetches non-sequential hit-point data from the memory using above flow chart. Note that the leaping value approach is very close to or the same as cycle-by-cycle interleaving [22] to eliminate data dependence.

V. Comparison Results and Chip Layout Implementation

The evaluation and chip layout characteristics as well as implementation of the proposed PREE architecture are addressed in this section. The cell-based design flow with standard cell library in TSMC 90 nm CMOS process with 1.0 V power supply voltage is adopted. The Synopsys Design Compiler is used to synthesize the register-transfer level (RTL) design and Cadence SOC Encounter is adopted to place and route for the proposed architecture. For the purpose of attaining the reliable PREE's performance and verification, 140MHz is used for the logic synthesis frequency. Consequently, the post-layout frequency is 125 MHz, where the margin 15MHz is used for tolerance of operating frequency degradation due to the place and route for PREE. The chip layout of the proposed PREE is shown in Fig. 12, where the ruler indicates the chip layout size by 1.336 × 1.333 mm^2.

Cornell box with 3 balls, Cornell box with teapot as well as hexahedron, Sibenik Cathedral, and Conference Room as shown in Fig. 13, Fig. 14, Fig. 15, and Fig. 16 are used to validate the functions of PREE via software and RTL simulations, respectively. The zoom-in images inside the red boxes are shown to compare software and RTL hardware results for each scene. The reflection, refraction and caustics effects can be observed in Fig. 13(c) and (d) and (e) and (f). The diffuse interreflection effect is shown in Fig. 14(c) and (d).

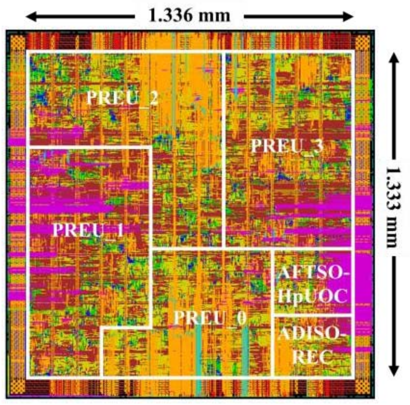


Fig. 12. Chip layout of PREE.

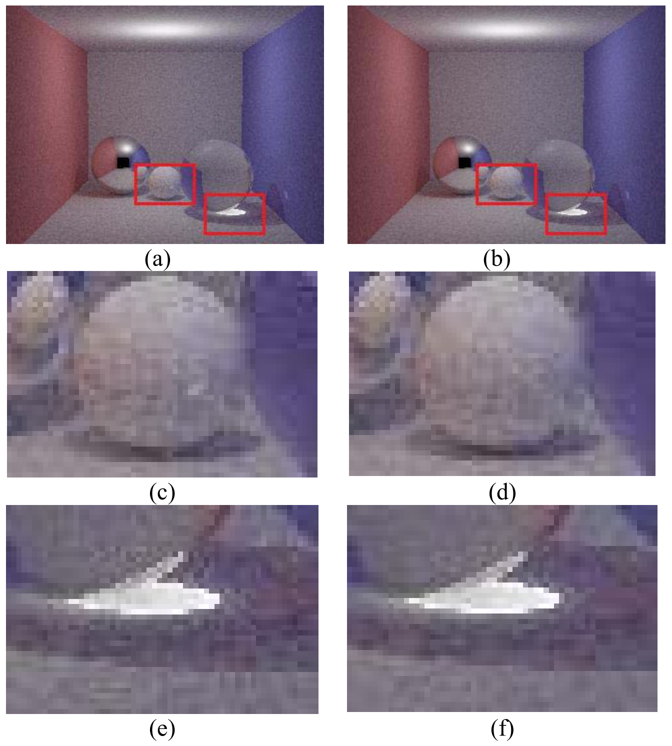


Fig. 13. Comparison result of Cornell box with 3 balls in 320 × 240 resolution. (a) Software, (b) RTL simulation, (c) reflection and refraction of software result, (d) reflection and refraction of RTL simulation result, (e) caustics of software result, and (f) caustics of RTL simulation result.

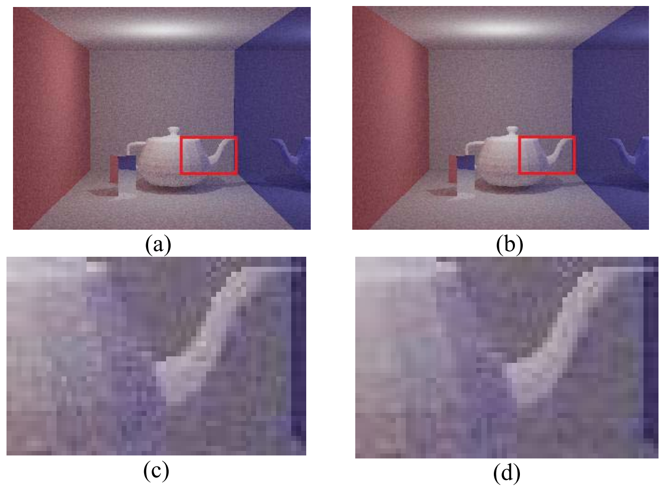


Fig. 14. Comparison result of Cornell box with teapot and hexahedron in 320 × 240 resolution. (a) Software, (b) RTL Simulation, (c) diffuse interreflection of software result, and (d) diffuse interreflection of RTL simulation result. (teapot source: http://cs.au.dk/∼miguel/ar/3dModel.l3d.small).

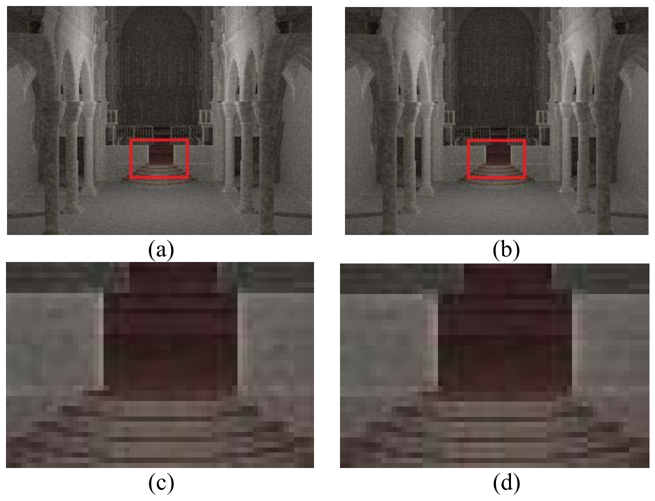


Fig. 15. Comparison result of Sibenik Cathedral in 320 × 240 resolution. (a) Software, (b) RTL Simulation, (c) software result, and (d) RTL simulation result. (http://graphics.cs.williams.edu/data/meshes.xml).

The rendering results for more triangles can be observed in Fig. 15(c) and (d). The changes in light and shadow are shown in Fig. 16(c) and (d). We compare the two images in terms of the structural similarity (SSIM) defined in [23] and signal to noise ratio (SNR), where the SSIM and SNR indices are used to evaluate the similarity quality between two images. The SSIMs of Fig. 13, Fig. 14, Fig. 15, and Fig. 16 between software and RTL simulations are 0.995461, 0.995420, 0.996328, and 0.995236, respectively. Considering software and RTL hardware simulation results, the SNRs of these four scenes are 83.35dB, 83dB, 81.57dB and 81.91dB, respectively. From about zoom-in images and quantitative numbers in terms of SSIM and SNR, the proposed PREE has satisfactory quality and slight difference via comparison between software and RTL simulations. In Table 3, the software approach adopting/modifying the source code [4] on Intel CPU i7-2600 3.4GHz with single thread can achieve 13.93 million hit-point update operations per second (MHpUO/s) at most from provided simulations

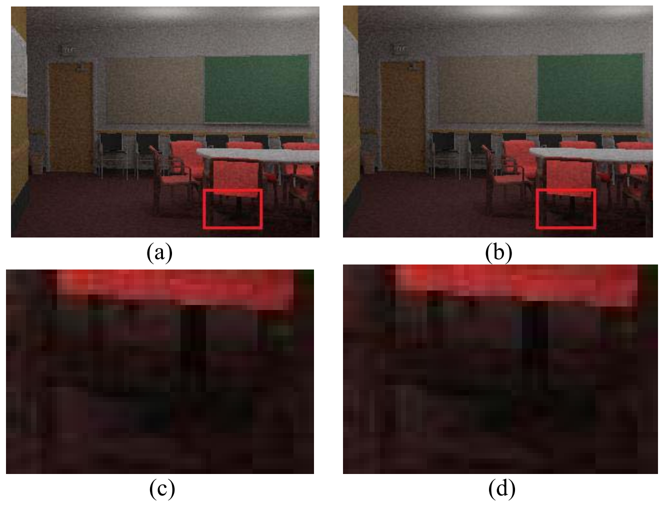


Fig. 16. Comparison result of Conference Room in 320 × 240 resolution. (a) Software, (b) RTL Simulation, (c) light and shadow of software result, and (d) light and shadow of RTL simulation result. (http://graphics.cs.williams.edu/data/meshes.xml#7).

according to four different scene resolutions in Fig. 1. Note that the above performance in terms of MHpUO/s is obtained by the number of hit-point update operations over the progressive radiance estimation execution time. Since the post-layout simulation is very time consuming for the high resolution scene, the chip layout implementation is validated by the post-layout simulation for emitting 1,000 photons in 20 × 15 resolution through 4 test scenes, where the power consumption is 184 mw, 162 mw, 155 mw, and 178 mw, respectively via Synopsys PrimeTime. According to the post-layout simulation results, the computation times of the proposed PREE implementation for 4 test scenes are 0.25 ms, 0.21 ms, 0.17 ms, and 0.10 ms and hit-point update operations rate can achieve 496.79 MHpUO/s, 497.71 MHpUO/s, 497.10 MHpUO/s, and 495.43 MHpUO/s, respectively. The proposed PREE can speed up the progressive radiance estimation by 35.66 times with emitting 1,000 photons in 20 × 15 resolution compared to the software approach with single thread with 768,000 photons in 320x240 resolution for Cornell box with 3 balls. The speedup number is calculated by (496.79MHpUO/s)/(13.93MHpUO/s) = 35.66, where 496.79MHpUO/s is listed in the last column in Table 3 and 13.93MHpUO/s is listed in note 7 in Table 3. Considering similarity quality, the SSIMs of four scenes in 20 × 15 resolution between software and post layout simulations are 0.980437, 0.975521, 0.975886, and 0.978602, respectively, and the SNRs of these four scenes are 48.92dB, 46.69dB, 45.37dB and 47.39dB, respectively. The comparison results and chip layout characteristics are summarized in Table 3. The memory space requirement in the last second row in Table 3 is determined by the number of stored photons N_{photon}, the number of stored hit-points $N_{hit-point}$ or the size of the scene resolution N_r, where the numbers depend on what algorithm is used. The expected value of the pixel estimate in the last row in Table 3 adopting (5) [7] in Section II.C is used to represent the theoretical graphic quality. The ASIC approaches [15], [17] are selected for comparison. In [15], the

TABLE III
COMPARISON RESULTS AND THE CHIP LAYOUT CHARACTERISTICS

Works	[8]	[15]	[16]	[17]	[19]	Software Approach	PREE [this work]			
Adopted Algorithm	APPM	Photon Mapping	PPM	Photon Mapping	SPPM	PPM	PPM			
Device (Implementat ion)	GTX 570 (NVIDIA OptiX)	ASIC	GTX 480 (NVIDIA OptiX)	ASIC & Simulator	GeForce GTX 680 (GLSL)	Intel CPU i7-2600	ASIC			
Core #	480*1	2 (Search x2 or 3+Shader x2)	480 *3	{TSMs, SOMs, SOAs}= {2/1/2, 16/16/2}*4	1536*5	4*6	4 (PREUs)			
Power Supply	NA	1.3 V	NA	Not clear	NA	NA	1.0 V			
Process Technology	40 nm*1	EDK SAED 90 nm	40 nm *3	Synopsys 90 nm Generic Library	TSMC 28 nm*5	32 nm*6	TSMC 90 nm CMOS			
Max. Clock	1464 MHz*1	1.5 GHz	1401 MHz *3	1.5 GHz	1006 MHz*5	3.4GHz*6	125 MHz			
Chip Layout /Package Area	520 mm^2*1 (Die Size)	< 1.5 mm^2	529 mm^2 *3 (Die Size)	{1.268, 19.619} mm^2	294 mm^2 *5 (Die Size)	37.5 x 37.5 mm^2*6 (Package Size)	1.336 x 1.333 mm^2 (not fabricated chip)			
Power	219 W*1	< 200 mW	250 W *3	{65.62, 976.64} mW	195 W*5	95 W*6	184*8	162*8	155*8	178*8
Scene	Cornell Box	Sponza or Sibenik	Cornell box	Cornell box	Cornell box	Cornell box + 3 balls	Cornell box + 3 balls	Cornell box + teapot	Sibenik	Conference Room
Scene Resolution	768x768	320x240	1024x768	450x450	*N/A*	320x240	20x15			
Performance	0.37 Mphotons/s	2 Billion shader operations/s	4.498 Mphotons/s	{2.498, 26.012} Billion shader operations/s	18~19 Mpaths/s	(*7)	496.79*9	497.72*9	497.10*9	495.43*9
Simulation Level	Profiling	Not clear*2	Profiling	Evaluation by Simulator using TSA & SOA synthesis information	Profiling	Profiling	Post-layout Simulation			
Mem. Space Requirement	$N_{hit\text{-}poiint}+N_r$	N_{photon}	$N_{hit\text{-}point}$	N_{photon}	$N_{hit\text{-}point}$	$N_{hit\text{-}point}$	$N_{hit\text{-}point}$			
Expected Value of the Pixel Estimate	c	$c+E[\frac{W}{p_e}]E[\bar{\varepsilon}_N]$	c	$c+E[\frac{W}{p_e}]E[\bar{\varepsilon}_N]$	c	c	c			

*1: Specification numbers are for the overall system card, where the information is available from NVIDIA. That means the numbers of cores, area, and maximum power are probably consumed not only by APPM but also by other operations in case this card is run for APPM.
*2: From the text description, although Singh *et al.* [15] did not clearly indicate the simulation level for the power and area values in Table 3, the authors clearly indicated that the data in Table II and Table III of [15] are obtained by synthesis simulation.
*3: The descriptions are the same as that of *1 except for PPM.
*4: Two configurations are picked up to show the performance of [17], where TSM and SOM denote tree search module and shader operation module, respectively.
*5: The descriptions are the same as that of *1 except for SPPM.
*6: The descriptions are the same as that of *1 except for Intel and PPM.
*7: 13.93 MHpUO/s (with single thread) and 50.92 MHpUO/s (with OpenMP to parallel for loops). *8: mw unit. *9: MHpUO/s (Peak Value).

proposed ASIC in 90 nm consists of the accelerated tree-search architecture and shader architecture for the photon mapping algorithm. In [17], the proposed ASIC in 90 nm consists of TSA and SOA. Compared with this work, although the speeds of [15] and [17] are faster in 1.5 GHz, both architectures in [15] and [17] realize the photon mapping algorithm such that the biased scene results will be incurred. Overall, in Table 3, the proposed PREE hardware design and implementation shows four features. First, to our best knowledge, compared with [8], [15]–[17], and [19] and the literatures in the reference, the proposed PREE hardware for PPM is the first efficient ASIC architecture work. Second, due to the PPM algorithm, the memory space requirement in our work is less than that of [15] and [17]. This is because $N_{hit-point}$ is much less than N_{photon} in most cases. Third, since the radius of [15] and [17] cannot be changed and updated, the PREE hardware for PPM should have better graphic quality than that of [15] and [17]. Fourth, this work has comparable performance to the software approach in Table 3. It is noted that ray tracing is one of phases of PPM and the accelerated ray tracing hardware implementations have been widely studied such as [20] and [21]. The accelerated hardware structures [20], [21] provide potential solutions for the ray tracing hardware of the complete PPM hardware. However, these ray tracing implementations cannot execute the radiance estimation computations including the hit-point update operation and radiance evaluation since they only focus on ray tracing rather than PPM.

VI. CONCLUSION AND FUTURE WORK

In this work, we propose the PREE architecture with four PREUs, AFTSO-HpUOC and ADISO-REC to accelerate the processing of the progressive radiance estimation of the PPM algorithm. By sharing the arithmetic resources, the presented pipelined data path of PREU can be reconfigured to implement either the hit-point update operation or the radiance evaluation. To increase the data parallelism degree, the proposed AFTSO-HpUOC controller with the four-entry buffer combining several distributed hit-points from at most four photons improves the utilization of the PREUs. During the radiance evaluation phase, the ADISO-REC controller adopts the low-complexity leaping access method to alleviate the data dependence. In summary, the hardware innovation and contributions of this paper are as follows. 1) A high computing PREE with satisfactory graphic quality is proposed to accelerate the progressive radiance estimation of PPM. 2) A novel configurable PREU is proposed to achieve resource sharing between hit-point update operation and radiance evaluation. 3) An AFTSO-HpUOC controller with a systematic PREU utilization is proposed to attain almost parallel operations. 4) An ADISO-REC controller applying the leading address method is proposed to alleviate the data dependence to avoid rendering incorrect pixel value. In future work, the FPGA implementation will be considered.

Acknowledgment

The authors would like to thank Associate Editor and anonymous referees for their valuable suggestions to this paper.

References

[1] H. W. Jensen, "Global illumination using photon maps," in *Proc. Eurogr. Workshop Rendering Techn.*, 1996, pp. 21–30.

[2] V. Havran, R. Herzog, and H.-P. Seidel, "Fast final gathering via reverse photon mapping," *Comput. Graph. Forum*, vol. 24, no. 3, pp. 323–333, Sep. 2005.

[3] T. Hachisuka, S. Ogaki, and H. W. Jensen, "Progressive photon mapping," *ACM Trans. Graph.*, vol. 27, no. 5, Dec. 2008, Art. no. 130.

[4] *PPM Source Code*. [Online]. Available: http://www.ci.i.u-tokyo.ac.jp/~hachisuka/smallppm_exp.cpp

[5] T. Hachisuka and H. W. Jensen, "Stochastic progressive photon mapping," *ACM Trans. Graph.*, vol. 28, no. 5, Dec. 2009, Art. no. 141.

[6] T. Hachisuka, W. Jarosz, and H. W. Jensen, "A progressive error estimation framework for photon density estimation," *ACM Trans. Graph.*, vol. 29, no. 6, Dec. 2010, Art. no. 144.

[7] C. Knaus and M. Zwicker, "Progressive photon mapping: A probabilistic approach," *ACM Trans. Graph.*, vol. 30, no. 3, May 2011, Art. no. 25.

[8] A. S. Kaplanyan and C. Dachsbacher, "Adaptive progressive photon mapping," *ACM Trans. Graph.*, vol. 32, no. 2, Apr. 2013, Art. no. 16.

[9] C. M. Kang, L. Wang, Y. N. Xu, and X. X. Meng, "A survey of photon mapping state-of-the-art research and future challenges," *Frontiers Inf. Technol. Electron. Eng.*, vol. 17, no. 3, pp. 185–199, 2016.

[10] T. J. Purcell, C. Donner, M. Cammarano, H. W. Jensen, and P. Hanrahan, "Photon mapping on programmable graphics hardware," in *Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. Graph. Hardw.*, 2003, pp. 41–50.

[11] S. Singh, "The photon pipeline," in *Proc. 4th Int. Conf. Comput. Graph. Interact. Techn. Australasia Southeast Asia ACM*, 2006, pp. 333–340.

[12] S. Singh and P. Faloutsos, "The photon pipeline revisited," *Vis. Comput.*, vol. 23, no. 7, pp. 479–492, 2007.

[13] M. McGuire and D. Luebke, "Hardware-accelerated global illumination by image space photon mapping," in *Proc. Conf. High Perform. Graph. (HPG)*, 2009, pp. 77–89.

[14] T. Hachisuka and H. W. Jensen, "Parallel progressive photon mapping on GPUs," *ACM SIGGRAPH ASIA Sketches ACM*, 2010, p. 54.

[15] S. Singh, S. H. Pan, and M. Ercegovac, "Accelerating the photon mapping algorithm and its hardware implementation," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Process. (ASAP)*, Sep. 2011, pp. 149–157.

[16] S. A. Pedersen, "Progressive photon mapping on GPUs," M.S. thesis, Dept. Comput. Inf. Sci., Norwegian Univ. Sci. Technol., Trondheim, Norway, 2013.

[17] S. H. Pan, "A multi-accelerator architecture for photon mapping," Ph.D. dissertation, Dept. Comput. Sci., Univ. of California, Los Angeles, CA, USA, 2014.

[18] T.-J. Kim, X. Sun, and S.-E. Yoon, "T-ReX: Interactive global illumination of massive models on heterogeneous computing resources," *IEEE Trans. Vis. Comput. Graphics*, vol. 20, no. 3, pp. 481–494, Mar. 2014.

[19] T. Hachisuka. (May 2015). "Implementing a photorealistic rendering system using GLSL." [Online]. Available: https://arxiv.org/abs/1505.06022

[20] Y. Gao, L. Zhou, Y. Chen, Y. Wang, J. Wang, and T. Sun, "A hardware acceleration engine for ray tracing," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng.*, Dec. 2014, pp. 505–508.

[21] X. Liu, Y. Deng, Y. Ni, and Z. Li, "FastTree: A hardware KD-tree construction acceleration engine for real-time ray tracing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 1595–1598.

[22] J. Silc, B. Robic, and T. Ungerer, *Processor Architecture: From Dataflow to Superscalar and Beyond*. Springer, 1999, ch. 6.

[23] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

Ching-Chieh Chiu received the B.S. and M.S. degrees from the National Pingtung Institute of Commerce, Pingtung City, Taiwan, in 2009 and 2011, respectively. He is currently pursuing the Ph.D. degree with the Faculty of the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. His research intersects include VLSI algorithms and 3-D computer graphics.

Lan-Da Van (S'98–M'02–SM'16) received the B.S. (Hons.) and M.S. degrees from the Tatung Institute of Technology, Taipei, Taiwan, in 1995 and 1997, respectively, and the Ph.D. degree from National Taiwan University (NTU), Taipei, in 2001, all in electrical engineering. From 2001 to 2006, he was an Associate Researcher with the National Chip Implementation Center, Hsinchu, Taiwan. In 2006, he joined the Faculty of the Department of Computer Science, National Chiao Tung University, Hsinchu, where he is currently an Associate Professor. His research interests are in VLSI algorithms, architectures, and chips for digital signal processing and biomedical signal processing. This includes the design of low-power/ high-performance/ cost-effective 3-D graphics processing system, adaptive/learning systems, computer arithmetic, filters, transforms, and intelligent IoT/M2M applications. He has published over 60 journals and conference papers in these areas.

Dr. Van was a recipient of the Chunghwa Picture Tube and the Motorola Fellowships in 1995 and 1997, respectively. He was an elected Chairman of the IEEE NTU Student Branch in 2000. In 2001, he has received the IEEE Award for outstanding leadership and service to the IEEE NTU Student Branch. He was a recipient of the Best Poster Award at an iNEER Conference for Engineering Education and Research in 2005, the Teaching Award of the Computer Science College, National Chiao Tung University in 2014, and the Best Paper Award in the IEEE International Conference on Internet of Things in 2014. Since 2009, he has been serving as the Officer of the IEEE Taipei Section. In 2014, he serves as a Track Co-Chair of the 22nd IFIP/IEEE International Conference on Very Large Scale Integration. In 2016, he served as a Program Co-Chair of the NCTU Forum of Technology and Application of Internet of Things. He serves as a Special Session Co-Chair of the 2018 IEEE International Conference on DSP. Since 2014, he has been serving as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS. Since 2017, he has been serving as a Board Member of the *Journal of Medical Imaging and Health Informatics*.

Yu-Shu Lin was born in Taipei, Taiwan, in 1987. He received the B.S. degree in computer science from Chung Yuan Christian University, Taoyuan, Taiwan, in 2010, and the M.S. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2015. His research interests include 3-D graphics system design.