Mobile-PBR: A 28-nm Energy-Efficient Rendering Processor for Photorealistic Augmented Reality With Inverse Rendering and Background Clustering

Shiyu Guo, *Graduate Student Member, IEEE*, Yuhao Ju, *Member, IEEE*, Xi Chen, *Student Member, IEEE*, Sachin S. Sapatnekar, *Fellow, IEEE*, and Jie Gu, *Senior Member, IEEE*

***Abstract*— This work presents a low-power physical-based ray-tracing (PBRT) rendering processor for photorealistic augmented reality (AR) rendering applications on mobile devices, referred to as mobile physical-based renderer (Mobile-PBR). By introducing inverse rendering (IR) and background clustering, Mobile-PBR enables complicated photorealistic lighting effects such as reflection, refraction, and shadow with minimum resources on mobile edge devices. The key features of this work include: 1) an ASIC rendering processor that embeds an end-to-end ray-tracing (RT) solution with IR for AR on mobile devices; 2) a reconfigurable mixed-precision processing element (PE) design supporting diverse computing tasks for both IR and RT modes; 3) background clustered field of view (FOV)-focused 3-D construction reducing conventional background scene complexity from O($n$log$n$) to O(1); 4) scalable partitioning scheme for complex 3-D objects with an average of 13× speed up on test scenes; and 5) use of global RT scheduler (GRTS) and global memory access controller (GMAC) to overcome the challenges of irregular memory access pattern and varied PE runtime with overall 684× speed up compared with the baseline design. A 28-nm test chip was fabricated demonstrating 500- and 1418-frames/s/W power efficiency in IR and RT modes, respectively, achieving 28.8× and 3.95× higher RT rendering efficiency compared with existing ASIC solutions, and having an average performance of 25.8 frames/s on various testing scenes, enabling real-time physical-based RT rendering on mobile edge devices.**

***Index Terms*— 3-D construction, deep neural network (DNN), inverse rendering (IR), low-power processor, physical-based ray-tracing (RT) rendering, system on chip (SoC).**

I. Introduction

AS THE applications of augmented reality (AR) or virtual reality (VR) expand rapidly with growing demands for enhanced visual realism, photorealistic image generation and insertion have become essential features for the emerging AR applications providing real-time workplace and household visual assistance. Physically based ray-tracing (PBRT) [1] is often used where synthesized images are generated by simulating the real environment and tracing the light transportation to achieve photorealistic effects, such as reflection, refraction, and soft shadows.

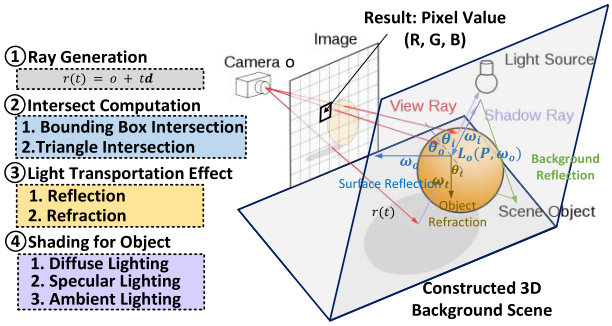


Fig. 1. Overview of RT rendering operation.

PBRT is widely used in product design, medical visualization, video games, and movie effects. To enable photorealistic rendering effects, there is a strong demand to support physical-based ray tracing (RT) on mobile devices [2].

RT is a rendering technique in computer graphics used to realistically simulate the way light interacts with objects to produce realistic images. The overview of the RT rendering algorithm is shown in Fig. 1. First, multiple camera rays $r(t)$ are generated from the origin and extended into 3-D space, which can be represented as a mathematical model $r(t) = o + t\mathbf{d}$. $r(t)$ represents the position of the ray and $\mathbf{d}$ represents the direction of the ray. Next, the intersection between rays and the 3-D primitives is checked. For all the intersections, the light transportation effects, such as reflection and refraction are computed during the ray-space interaction. Finally, all the shading effects for objects are combined and computed to determine the resulting pixel value.

However, the challenges of RT rendering on mobile devices are tremendous. To render a photorealistic object in 3-D space, all the information, including environmental lighting, geometry for background meshes, and material maps for all the surfaces is needed. Besides the large amount and various types of data maps, the iterative RT flow also causes uncertainty in computing time and thus requires powerful GPU platforms for real-time RT rendering [2].

Received 22 May 2024; revised 20 July 2024, 14 September 2024, and 10 October 2024; accepted 14 October 2024. This article was approved by Associate Editor Chia-Hsiang Yang. This work was supported in part by Air Force Research Laboratory (AFRL) through Defense Advanced Research Projects Agency (DARPA) Real Time Machine Learning (RTML) Program under Award FA8650-20-2-7009 and in part by NSF under Grant CCF-2008906. *(Corresponding author: Jie Gu.)*

Shiyu Guo, Yuhao Ju, Xi Chen, and Jie Gu are with the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208 USA (e-mail: shiyuguo2021@u.northwestern.edu; yuhaoju2017@u.northwestern.edu; xichen2020@u.northwestern.edu; jgu@northwestern.edu).

Sachin S. Sapatnekar is with the Electrical and Computer Engineering Department, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: sachin@umn.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2024.3484212.

Digital Object Identifier 10.1109/JSSC.2024.3484212

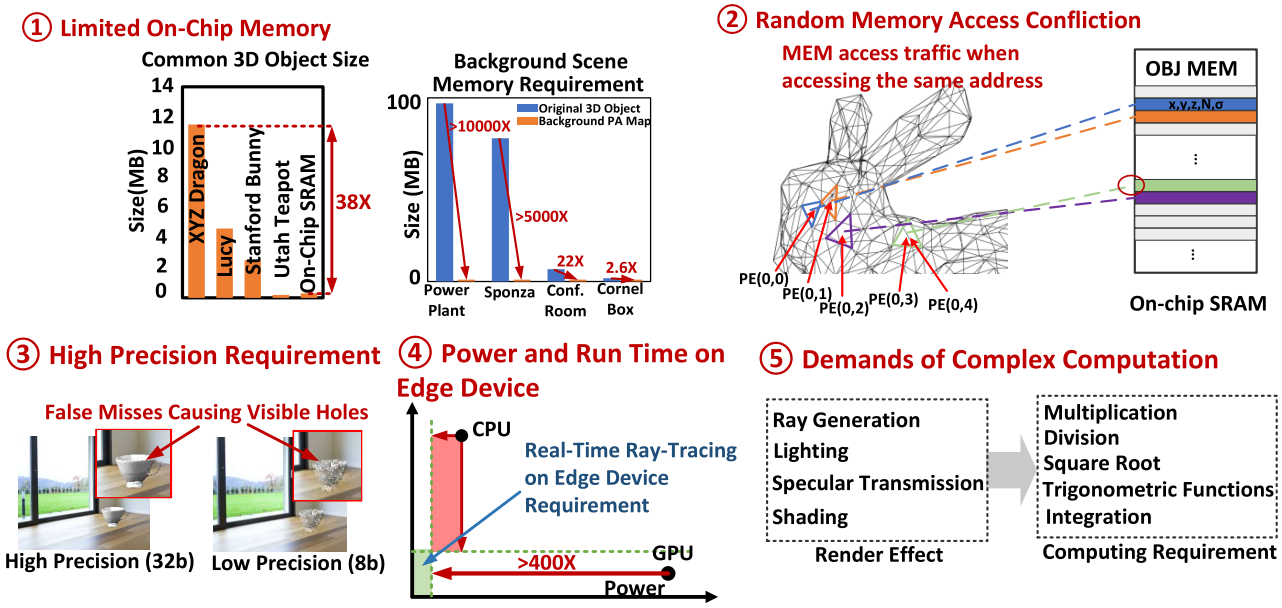


Fig. 2. Challenges for RT rendering on mobile edge devices.

TABLE I
STATE-OF-THE-ART RENDERING HARDWARE FOR AR/VR APPLICATIONS

Platform	Application	Rendering Algorithm	Memory Requirement	Photorealistic	Power Efficiency
GPU Server	Nvidia RTX	Ray-Tracing	☹	☺	☹
Headset	Apple Vision Pro	Rasterization	☹	☹	☹
	Microsoft HoloLens	Holographic Rendering	☹	☹	☹
	Meta Quest 2	Rasterization	☹	☹	☹
Mobile GPU	Samsung S23	Ray-Tracing	☹	☺	☹
ASIC	ISSCC'23[4]	DNN	☹	☺	☺
	ISSCC'24[5]	DNN	☺	☺	☹
	Reconf. SIMT[6]	Ray-Tracing	☺	☺	☹
	This Work	IR - RT	☺	☺	☺

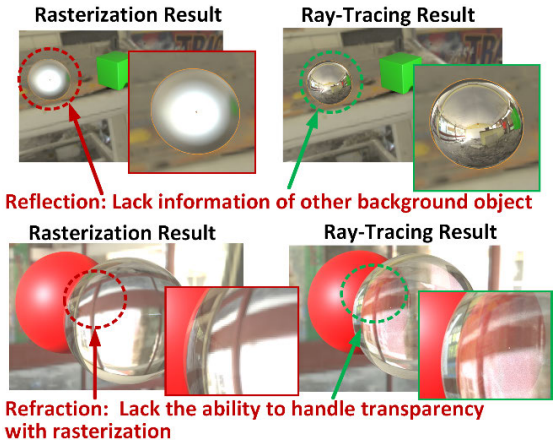


Fig. 3. Photorealistic rendering effect limitations for rasterization.

As Fig. 2 shows, the challenges in mobile RT rendering are summarized as follows: 1) complex common 3-D objects [3] with high memory requirements exhaust the limited on-chip memory space on edge devices; 2) unstructured memory access patterns between different processing elements (PEs) and complex control flow lead to scheduling difficulty during computation; 3) RT computing has extremely low error tolerance, which requests high precision for computing; 4) hardware resources limitation on edge devices; and 5) complex computations, such as division and square root require high computing resources and complex computing flow for the edge devices.

The current state-of-the-art rendering hardware implementations are shown in Table I. There are four main categories: 1) ASIC implementations [4], [5], [6]; 2) GPU servers [7]; 3) VR and AR headsets [8], [9], [10]; and 4) mobile GPUs [11]. Most of the platforms are using rasterization rendering as their rendering algorithm due to its lower cost and easier implementation.

Unfortunately, rasterization rendering breaks down the 3-D scene into separate graphic buffers and renders them onto a 2-D screen. It fails to produce photorealistic results due to its fundamental approach to rendering objects without recording the physical behavior of light rays and requires additional techniques for light approximations, as shown in Fig. 3.

Only a few ASICs have been fabricated so far as a mobile RT solution for real-time AR/VR rendering [2], [4], [5], [6]. In [4], the proposed processor bypasses the conventional RT rendering operations [12] by introducing a deep neural network (DNN) rendering technique called neural radiance field (NeRF, [13]). In this work, conventional RT computations are replaced by brain-inspired visual attention-based DNN operations. However, instant 3-D modeling is not supported in this work due to the complex memory requirements for the hash table (>23 MB) [5]. In [5], a NeuGPU is proposed, featuring a special segmented hashing with a spatial pruning module for both instant 3-D modeling and NeRF rendering. However, the original NeRF algorithm is limited to opaque surfaces and needs a special handle and more effort for complex lighting effects such as reflection, refraction, and shadow [14].

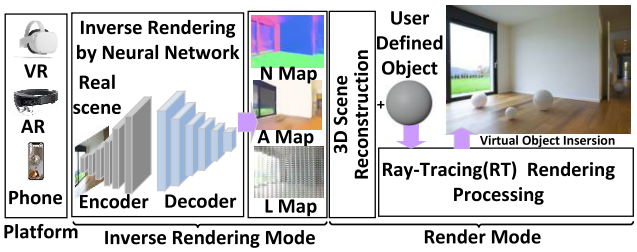


Fig. 4. Computing flow for Mobile-PBR.

In [6], the work proposed and implemented a processor for memory contention reduction during 3-D background scene construction but a large amount of computing and memory for background scene primitives are still needed.

To address the challenges for mobile RT rendering, in this article, we introduce a new processor, a mobile physical-based renderer (Mobile-PBR), which adopts inverse rendering (IR) [15] to extract background physical information and RT rendering for 3-D object insertion [16]. The rendering flow is shown in Fig. 4. In this flow, IR is used to acquire background geometry, reflectance, and lighting information from RGB images captured by regular cameras. The decoded 3-D geometry and background physical information are used for photorealistic RT rendering tasks. Mobile-PBR proposes an efficient data structure, and scalable computing scheme and therefore bypasses conventional background construction, enabling real-time photorealistic rendering on mobile devices with minimal cost.

The key features of this work include: 1) an ASIC rendering processor that embeds IR with RT solution for AR applications on mobile devices; 2) a reconfigurable mixed-precision PE design supporting diverse computing tasks for both IR and RT tasks; 3) background clustered field of view (FOV)-focused 3-D construction reducing conventional background scene complexity from O($n$log$n$) to O(1); 4) a scalable partitioning scheme for complex 3-D objects, with an average of 13× speed up on test scenes; and 5) use of global RT scheduler (GRTS) and global memory access controller (GMAC) to overcome the challenges of irregular memory access patterns and compute resources with overall 684× speedup compared to the baseline design. The 28-nm test chip achieves 3.95∼28.8× higher RT rendering efficiency compared to the existing ASIC solutions, enabling real-time PBRT rendering on mobile edge devices.

The rest of this article is organized as follows. In Section II, the rendering algorithm, scalability support of Mobile-PBR, and the benefits are introduced, and the following Section III describes the overall chip architecture and dataflow. Section IV explains the details of reconfiguration mode for RT and IR modes in PE. Section V shows the chip implementation results, measurement results, and rendering cases of Mobile-PBR.

II. Rendering Algorithm and Scalability for Mobile-PBR

A. Overall Rendering Flow for Mobile-PBR

There are three major computing steps for rendering tasks in Mobile-PBR: 1) IR; 2) 3-D background construction; and 3) RT rendering. The major complexity comes from the ray-object and ray-background intersection process, which has a computational complexity of O($n$log$n$) [2], where n represents the number of triangle primitives constructing 3-D objects and scenes. In this work, the proposed rendering flow greatly reduces the computational cost compared with conventional RT rendering tasks.

The first step is shown in Fig. 5(a). The IR flow proposed by [15] is implemented in Mobile-PBR. A 2-D RGB image captured by a regular camera is sent through a pre-trained DNN-based physical decoder and encoder for IR inference to estimate the background physical attributes (PAs). The background PAs are used to reconstruct geometry, reflection, and lighting for the 3-D background space. There are four major background PA maps acquired from IR: albedo, normal, lighting, and depth maps. For the lighting map, threshold per-pixel lighting is applied to skip the lighting mapping step for pixels that are dark or lack a light source, using a programmable threshold value t, as shown in Fig. 5(a). The 2-D lighting map and depth map are used to construct a 3-D lighting map, as shown in Fig. 5(b). To save the complete PA maps on-chip, a background clustering scheme is developed based on the similarity of neighbor pixel values of the 2-D background map by applying an average filter. The result PA maps are stored in an on-chip PAMEM. PAMEM is a shared global memory across the computing array. Each PE accessing the PAMEM passes through the per-pixel compression decoder (PPCD) and the unified address converter (UAC) to fetch the corresponding background PA parameters based on the PE task ID from GRTS. The detailed mechanism and implementation for GRTS are discussed in Section III, and the details for PPCD and UAC are discussed in Section IV.

The second step is the FOV-focused 3-D construction and geometry conversion using the IR PA maps, as shown in Fig. 5(c). The view rays start from the user camera origin and are cast to each pixel in the image viewing plane covered by the camera FOV. The background PAs are provided by the IR step in the first step. Fig. 6 shows the 2-D to 3-D ray geometry conversion in Mobile-PBR between (PixelCamera$_x$ and PixelCamera$_y$) and (PixelScreen$_x$ and PixelScreen$_y$). Each ray intersects with the background 3-D scene and takes the PA maps acquired in the first step for the following rendering operation. By introducing IR into 3-D RT flow, a large portion of background triangle primitive intersection computing is reduced to four constant-size 2-D FOV-focused PA maps. In this way, the background scene intersection complexity remains unaffected by the total number of primitives, reducing from O($n$log$n$) to O(1) compared to the conventional RT solutions [2].

In the last step, regular RT rendering is implemented with a customized hardware acceleration architecture to render 3-D virtual objects. The overall rendering flowcharts for the conventional RT workload and the RT workload proposed in this work are shown in Fig. 7(a) and (b), respectively. Compared with conventional RT workload, most of the iterative background triangle bounding volume hierarchy (BVH) [17], [18] construction operations such as copy, generate node, sort, and KD-tree traverse [19] are replaced by PA maps

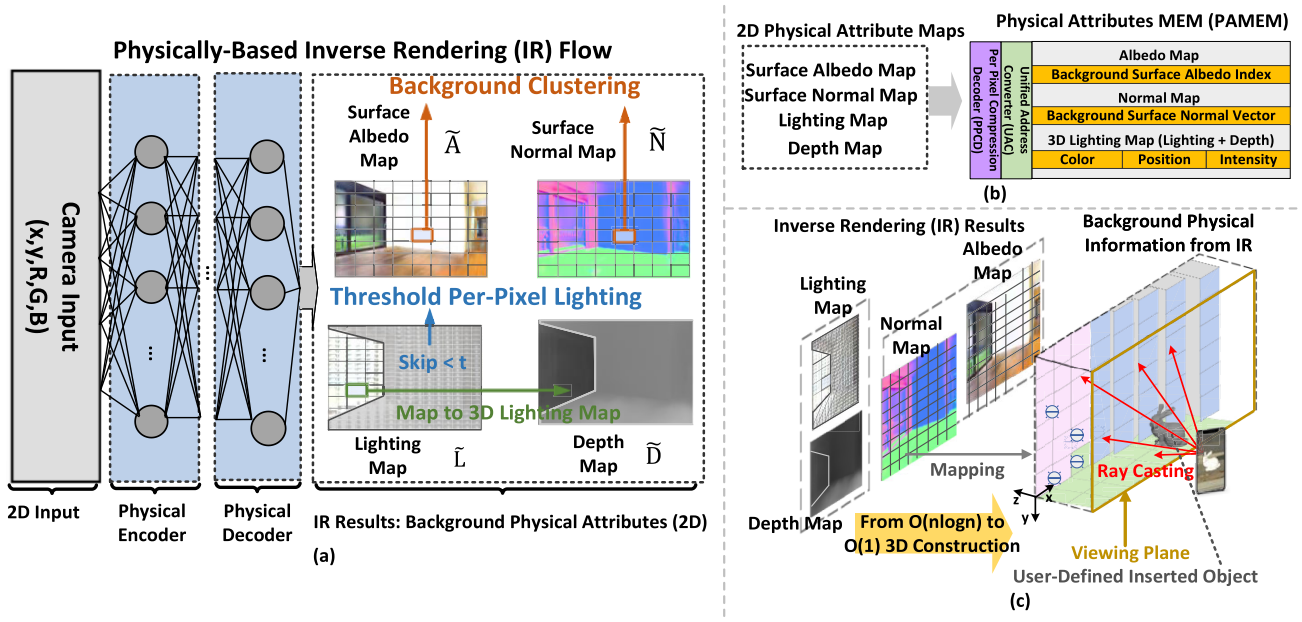


Fig. 5. IR and 3-D construction in Mobile-PBR. (a) IR flow. (b) Global PAMEM. (c) 3-D construction flow with IR results.

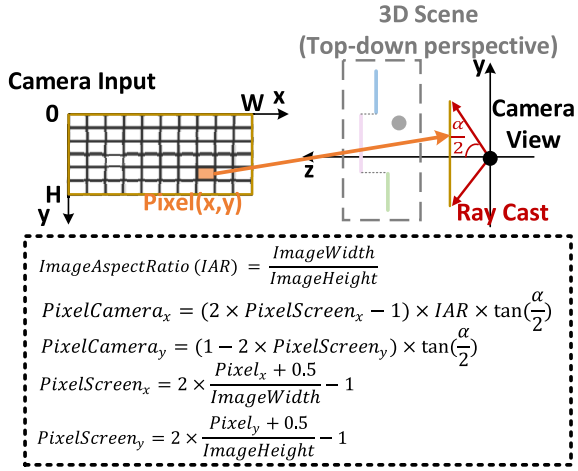


Fig. 6. Geometry conversion for 3-D construction in Mobile-PBR.

from IR. As a result, around 76% RT workload can be reduced compared with the conventional RT solution shown in Fig. 7(c). The bottleneck of storing the complete complex background triangular mesh geometry on-chip is addressed by storing four 2-D PA maps on-chip.

These three major steps can reduce both computational load and memory requirements for photorealistic RT rendering but it requires optimized ASIC for optimal performance. In Mobile-PBR, a computing core supporting all operations is implemented and will be discussed in Section III.

B. Scalable 3-D Object Partitioning Scheme for Mobile-PBR

As introduced earlier, there are two major bottlenecks for rendering objects in complex 3-D scenes on mobile devices: 1) limited on-chip memory for complex backgrounds and objects and 2) computing time for triangle intersection. To address these bottlenecks, we adopt IR into the rendering flow in this work to reduce the background mesh intersection complexity. However, the memory and computational complexity of the 3-D objects still remain the same. As shown in Fig. 2, common complex objects or 3-D background scene sizes can easily exceed the on-chip SRAM size for edge devices.

To address the memory overhead caused by complex 3-D objects, Mobile-PBR adopts a customized efficient data structure tailored to AR rendering applications on mobile edge devices: global tracing bounding box (BBOX). Different from the conventional acceleration solutions that build the BVH acceleration with the axis-aligned BBOX [18], we introduced two types of global tracing BBOX in this work: empty BBOX (EBBOX) and target BBOX (TBBOX), as shown in Fig. 8(a).

For each complex virtual 3-D object that could not directly fit into the on-chip memory, it is necessary to segment the object and process it segment by segment. Since RT is a global-scope iterative process, rendering the object segment by segment will lose the record of global light transportation effects and result in lost shadow information, as shown in Fig. 8(a). To address this issue, the object is defined as three components in this work: EBBOX, TBBOX, and a subgroup of user-defined triangle primitives inside TBBOX.

The detailed BBOX flow is shown in Fig. 8(b). If the cast ray intersects with BBOX, the BBOX intersection flag (BBIF) will be set to 1 and proceed to the 3-D partition check. Otherwise, BBIF will be set to 0 and the rest of the computation is skipped for the next iteration of RT light transport. In the 3-D partition check, the type of the intersected BBOX is checked. If the ray intersects with TBBOX, the BBOX intersection evaluator (BBIE) inside each PE will call the triangle mesh intersection evaluator (TIE) to compute the ray-triangle intersection, RT light transport with shading computation. If the ray intersects with EBBOX, the stage will proceed to RT light transport and skip shading computation. In this way, EBBOX is only used to record light transportation estimation and shadow computation without

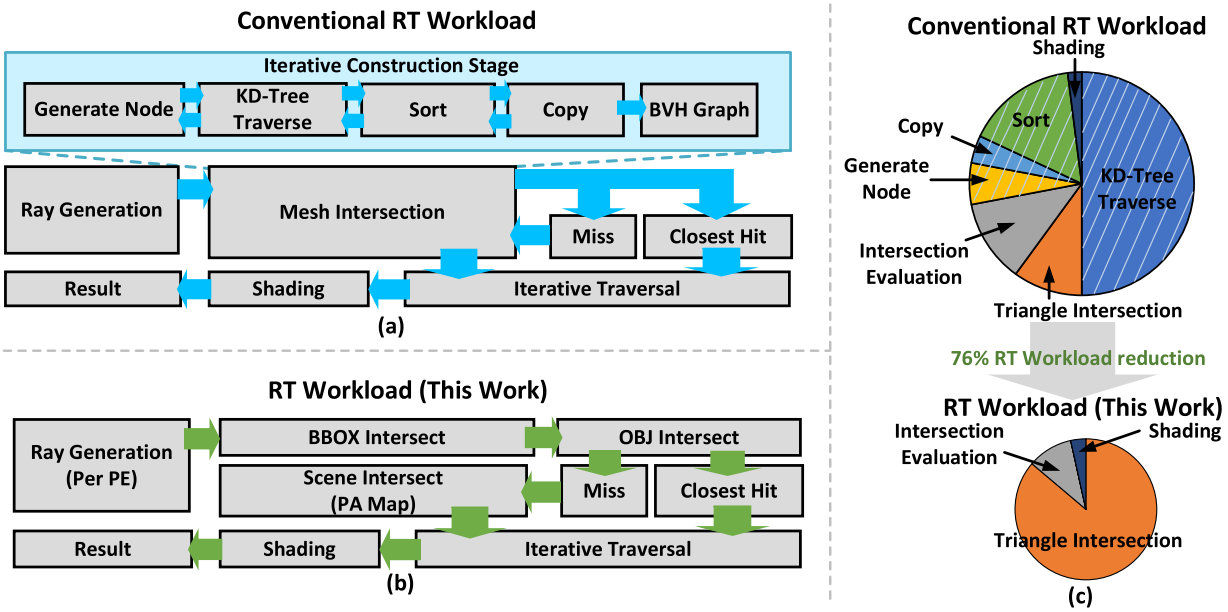


Fig. 7. Computing flow and comparison results between Mobile-PBR and conventional ray-tracing rendering. (a) Rendering flowchart for conventional RT workload. (b) Rendering flowchart for the RT workload in this work. (c) Workload reduction compared with conventional RT solutions.

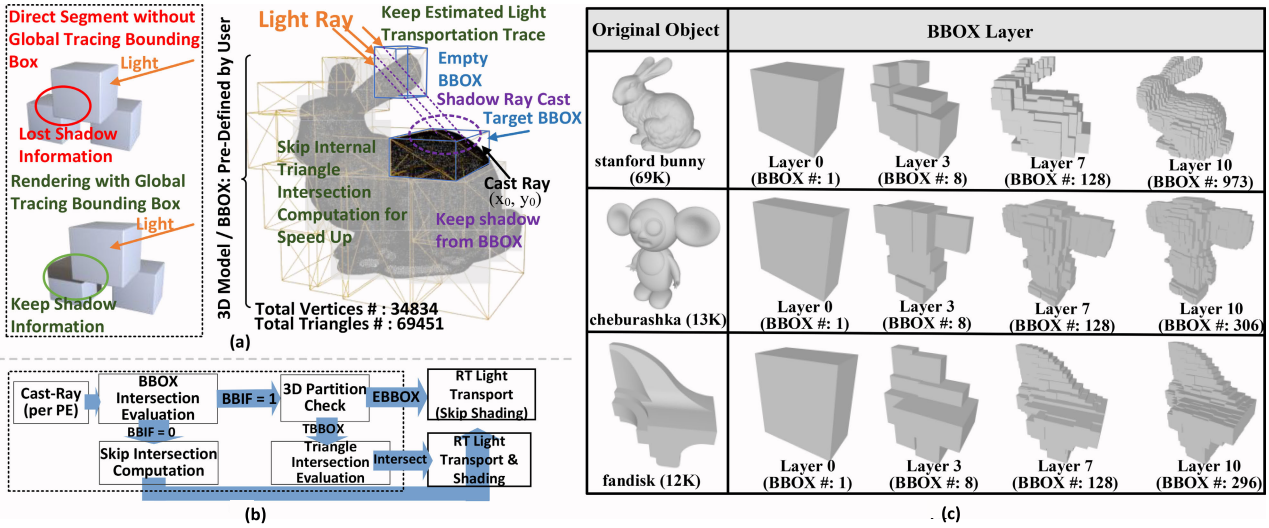


Fig. 8. 3-D object partitioning scheme in Mobile-PBR. (a) Global tracing bounding box with EBBOX and TBBOX. (b) Scalable 3-D model partitioning flow in Mobile-PBR. (c) Examples of common 3-D objects with the original triangle primitives number and different BBOX layers.

shading effect. During the ray-object intersection evaluation stage, the triangle primitives in both TBBOX and EBBOX can directly be skipped if the ray has no intersection with the BBOX. The segmentation for the objects is determined by both objects and on-chip buffer size. Examples of Stanford bunny, cheburashka, and fandisk are shown in Fig. 8(c). The rendering time for each 3-D object varies based on its complexity, as different shapes result in varying configurations of BBOX and differing numbers of BBOX layers. As the BBOX layer goes deeper, more BBOX coordinates need to be stored on-chip, but fewer triangle primitive coordinates are needed to store in the OBJMEM. Different layers of BBOX and corresponding triangle primitives are determined offline with the consideration of on-chip buffer size. With this scheme, the intersection evaluation time is greatly reduced, and complex 3-D objects can be segmented for the RT process without losing the RT effect. As a result, shown in Fig. 9(a) and (b), an average of 13× speed up for 3-D model intersection evaluation is achieved compared with the baseline design with only 5.6% memory overhead for storing extra BBOX information on-chip.

III. Overall Chip Architecture and Dataflow of Mobile-PBR

A. Overall Chip Architecture of Mobile-PBR

Fig. 10 shows the overall chip architecture of Mobile-PBR. A reconfigurable 8 × 6 PE array serves as the computing core in both IR and RT modes. IR inference is accelerated by the computing array in IR mode. RT rendering task is accelerated by the same computing array with a different configuration. In IR inference mode, the PAMEM banks are reconfigured as input MEM banks, and each PE row shares the same WMEM with weight stationary dataflow. The results of IR inference are saved to OMEM banks. A global digital controller oscillator (DCO) provides a tunable global clock to the chip. The top

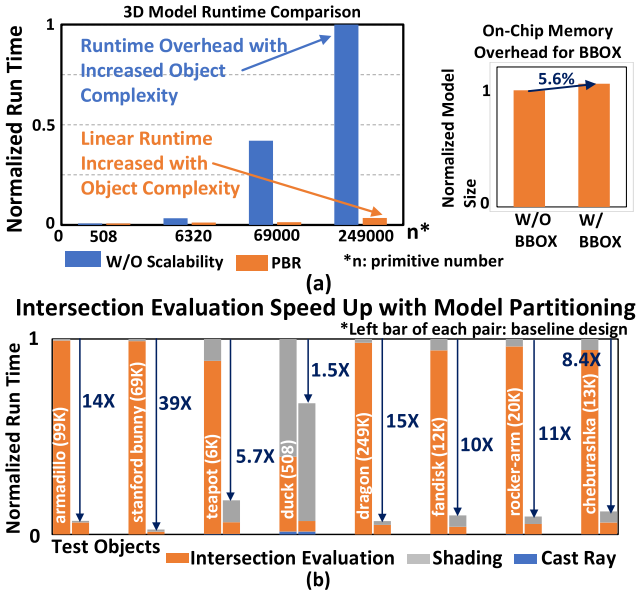


Fig. 9. Comparison results between the proposed 3-D object partitioning scheme and the baseline scheme without the BBOX evaluation scheme. (a) Scalability results for complex 3-D models and on-chip memory overhead for saving BBOX information. (b) Speed up with model partitioning on different test objects.

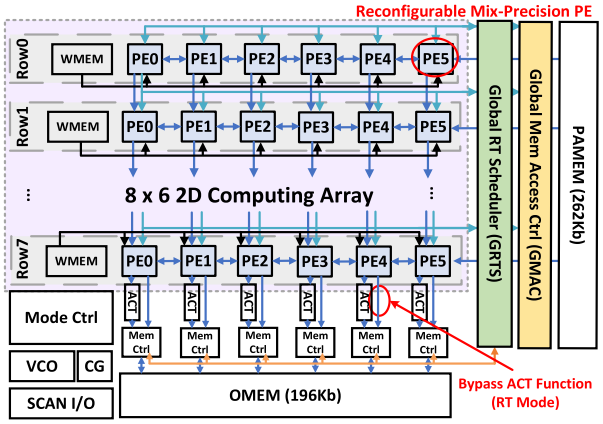


Fig. 10. Overall chip architecture of Mobile-PBR.

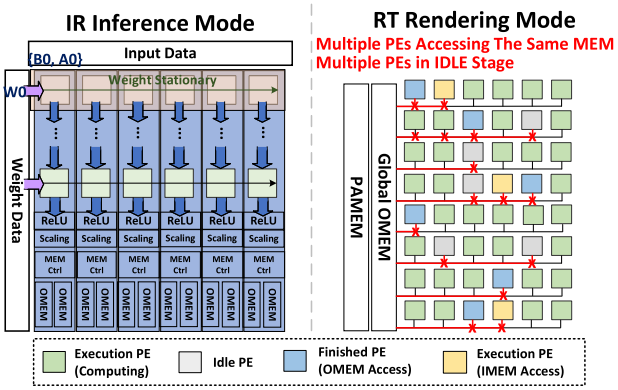


Fig. 11. Dataflow modes in PBR: IR inference mode and RT rendering mode.

mode controller consists of DNN control and RT control. They are used to control the computing mode switching and data flow switching. The GMAC is used for scheduling global memory access among the PE array in RT mode. The GRTS is used for scheduling dynamic computing requests from each PE in RT mode.

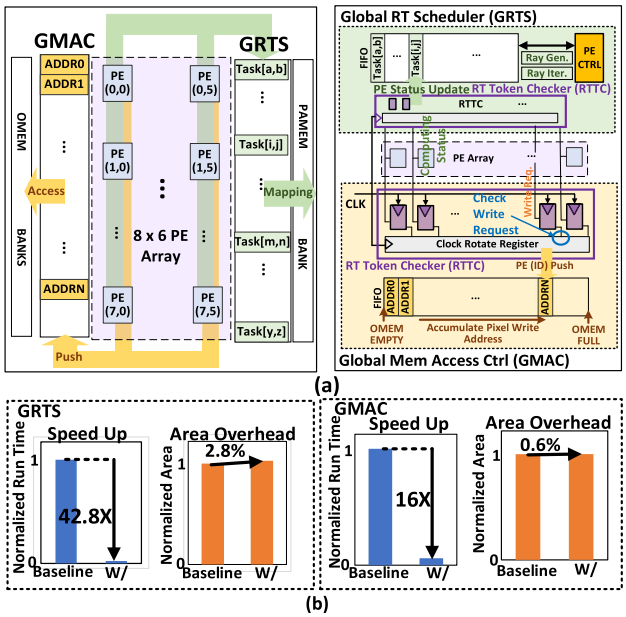


Fig. 12. Proposed computing flow and comparison results. (a) GMAC and GRTS top-level view and detailed implementation with RTTC. (b) Benefits and costs for GRTS and GMAC.

B. Dataflow Overview in Mobile-PBR

Two reconfigurable dataflows in Mobile-PBR are shown in Fig. 11: IR mode and RT mode. In IR inference mode, the PE array is configured as a systolic array for acceleration. The resulting PA maps are stored in OMEM. When switching to RT rendering mode, the PA maps from IR are reloaded to PAMEM for RT computation. The overall memory reloading latency is constrained by the limited I/O p-i-ns on-chip. The specifics of the I/O interface are discussed in Section V-A. Each ray cast from the camera origin into the 3-D space has a different and unpredictable workload. As a result, multiple PEs may request access to the same SRAM address at the same time, leading to memory access conflicts. For the same reason, multiple PEs may complete the rendering tasks simultaneously and enter the idle stage, resulting in low PE utilization.

To avoid memory conflicts and improve PE utilization, GRTS and GMAC are implemented in this work, as shown in Fig. 12(a). There is one RT token checker (RTTC) inside both GRTS and GMAC. The RTTC employs a clock-controlled rotating register to sequentially monitor the status or memory requests of each PE in the array, checking one PE per clock cycle and storing the results in a fixed size FIFO as shown in Fig. 12(a). The size of the FIFO corresponds to the number of PEs in the computing array, which is 48 in this test chip. GRTS uses the RTTC result from the PE array to refresh the checked PE status if the computation is done. GMAC uses the RTTC result to direct MEM write requests sequentially to avoid conflicts. By doing so, the PE refresh status and memory access requests can be pipelined with minimal cost: A 42.8× overall speedup from GRTS for total computing time and 16× overall speedup from GMAC for memory accessing time are achieved by introducing 2.8% and 0.6% extra hardware cost, comparing with the baseline scheme in which waits for all PE to finish and launch another round of computation for the PE array.

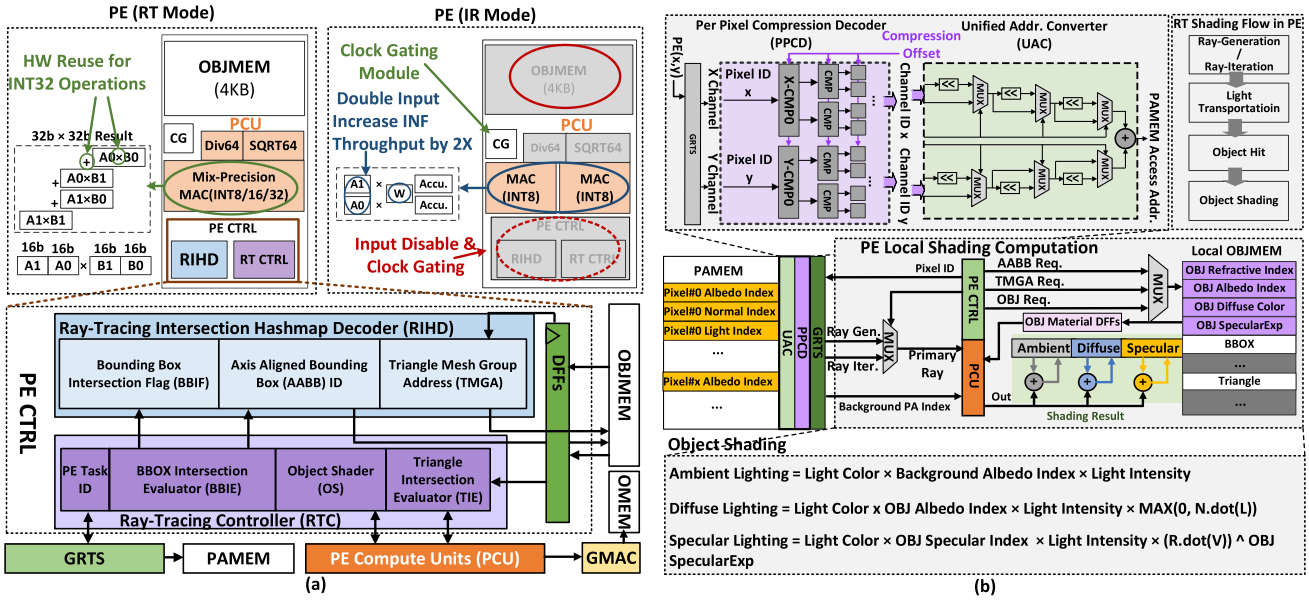


Fig. 13. Reconfigurable PE in Mobile-PBR. (a) PE configuration in RT and IR modes. (b) Datapath for PPCD, UAC, and object shading computation in each PE.

IV. Reconfigurable Computing Mode in Mobile-PBR

A. PE Configuration in RT Mode

In Mobile-PBR, each PE unit can be reconfigured to both IR and RT modes. As shown in Fig. 13(a), in RT mode, all the computing modules are enabled in the PE compute units (PCUs): 64-bit division, 64-bit sqrt, and a mixed-precision multiplication–accumulation (MAC). The mixed-precision MAC is implemented with a pipelined hardware reuse scheme: the 16-bit multiplier and adder calculate 32-bit multiplication in four clock cycles. During the object intersection evaluation stage, 64-bit precision is enabled to avoid false intersection results. The results from 64-bit MAC are used for 64-bit division and 64-bit sqrt. Each PE consists of a 4-kB local OBJMEM for storing object information such as object refractive index, object albedo index, object diffuse color, object specular exponent, BBOX coordinates, and triangle mesh coordinates as shown in Fig. 13(b).

Each PE unit communicates and fetches background PA data from PAMEM through GRTS, PPCD, and UAC. As shown in Fig. 13(b), each PE sends task requests (x, y) to the x channel and y channel in PPCD. The request IDs x and y are processed through a comparator tree for the channel ID of each task request. The compression offset is programmed in advance to align with the compressed PA maps. The output of PPCD is the channel ID of PA maps. Channel IDs x and y are sent through UAC to convert to the requested PAMEM access address.

During the intersection and shading computation, the data stored in the global PAMEM and local OBJMEM are sent to the PCU. The RT shading flow inside each PE is shown in Fig. 13(b). The computing requests for ray generation and ray iteration are issued from GRTS. In each PE controller, the RT intersection hashmap decoder (RIHD) and RT controller (RTC) are implemented for the customized computing flow in Mobile-PBR. During light transportation, the light ray direction is recorded in each PE. The BBIF, axis-aligned BBOX ID (AABB ID), and triangle mesh group address (TMGA) are stored in the local PE RIHD for geometry intersection evaluation. The PE task ID, BBIE, object shader (OS), and triangle intersection evaluator are stored in the local PE RTC for independent RT tasks. The RTC within each PE communicates with GRTS for dynamic task pipelining.

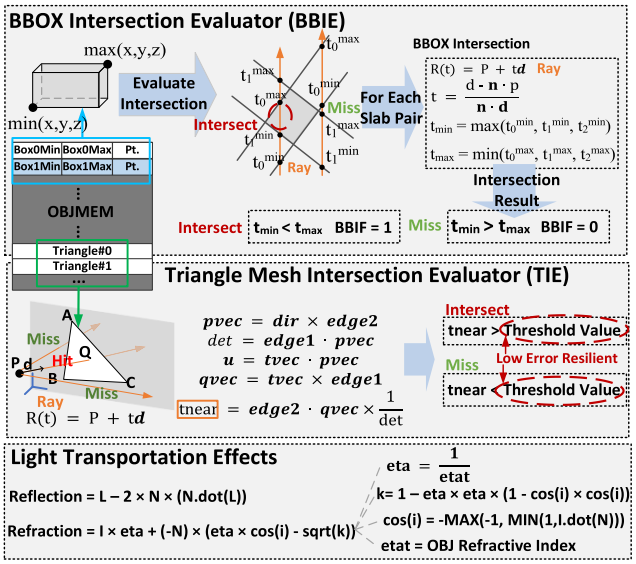
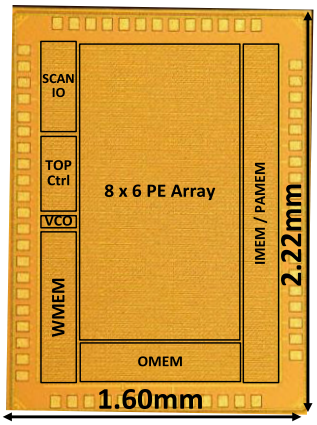


Fig. 14. Computation details for BBIE, TIE, and light transportation in each PE.

The detailed computations for BBIE and TIE are shown in Fig. 14. As introduced in the scalable 3-D object partitioning scheme, during the intersection evaluation stage, the BBOX intersection is evaluated first. In this work, each BBOX is defined using two coordinate pairs: min (x, y, z) and max (x, y, z). Each BBOX contains three pairs of slabs. The intersection is evaluated for each slab pair. For each slab pair, $t_{\min}$ and $t_{\max}$ are computed. If $t_{\min} < t_{\max}$, the ray intersects with the BBOX; otherwise, the ray misses the BBOX. Once the ray intersects with TBBOX, BBIF is set to 1 and TIE is



Process	28nm CMOS
Area (mm^2)	3.56mm^2
SRAM	296KB
Supply Vdd	0.6~0.9V
Bit Precision	INT8,16,32,64
Max IR Frequency	200MHz
Max RT Frequency	148MHz
IR Power	40mW @ 0.9V
RT Power	55mW @ 0.9V
IR Efficiency (FPS/W)	500
RT Efficiency (FPS/W)	1418

Fig. 15. Chip micrograph and specifications.

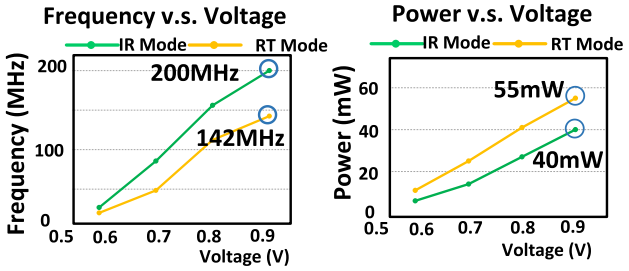


Fig. 16. Measured power and frequency with voltage scaling.

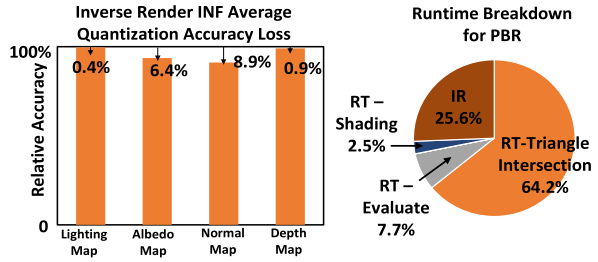


Fig. 17. IR average quantization accuracy loss and runtime breakdown for PBR.

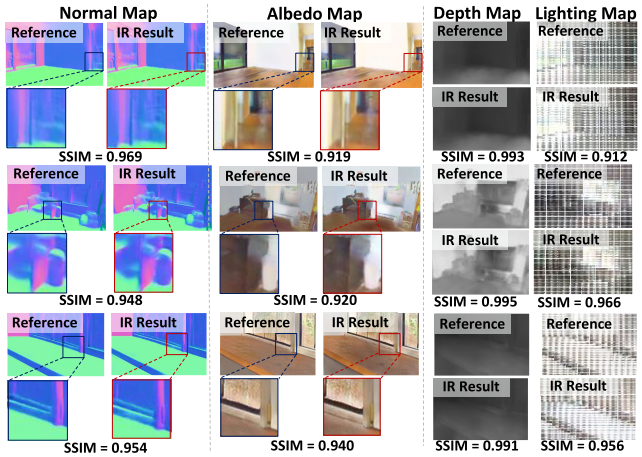


Fig. 18. IR results and SSIM on four PA maps.

launched for ray-triangle intersection evaluation. The detailed computation is shown in Fig. 14. The triangle mesh coordinates are stored in OBJMEM. In the end, the t_{near} value against a threshold value is used to determine if the ray intersects with the triangle mesh. Ideally, t_{near} should be compared with 0 for intersection evaluation, but due to the quantization effect, the threshold value is no longer 0. TIE requires extremely high computing accuracy and has very low error resilience. A false intersection or false miss will change the shape of the final rendered object [20]. As shown in Fig. 14, division and sqrt are used in light transportation computation, lower precision, such as 32-bit or 48-bit, will cause visible degradation in the rendering result [21]. In Mobile-PBR, 64-bit INT precision is supported in the PCU to address the high accuracy requirement.

After intersection evaluation, each PE computes the shading effect for the object surface, including ambient, diffuse, and specular lighting individually. The PCU results are stored in local shading registers for lighting effect accumulation as shown in Fig. 13(b). By implementing the proposed architecture in each PE, all the RT tasks can be completed individually within each PE.

B. PE Configuration in IR Mode

IR inference mode is shown in Fig. 13(a). Double input and weight stationary are supported in the IR dataflow. The clock gating (CG) and input gating module gate off the unused input ports, excessive computing logic in RT mode, and local OBJMEM banks for power-saving purposes. The PCU in IR mode supports two 8-bit MAC operations for 8-bit input and weight and a maximum 32-bit accumulation output. The whole 2-D computing array is configured as a systolic array to support DNN operations. The IR result is accumulated and saved to OMEM banks. Overall, around 32% of power saving is achieved in IR mode compared with the baseline implementation without CG techniques.

V. Chip Implementation, Measurement Results, and Case Study

A. Chip Implementation

A 2-D 8 × 6 Mobile-PBR processor was designed and fabricated using a 28-nm CMOS process. The chip micrograph and implementation are shown in Fig. 15. The active die area is 1.60 × 2.22 mm with a supply voltage of 0.6–0.9 V and 200- and 148-MHz operating frequency in IR and RT modes, respectively. IR mode can support 8-bit integer bit precision for DNN acceleration, and RT mode can support 8–64-bit mix-precision for MAC, division, and sqrt operations. The total on-chip SRAM is 296 kB. This chip provides a 32-bit scan I/O interface to load and read out all on-chip SRAM content. A field-programmable gate array (FPGA) board is engaged in chip testing for data streaming in and out of the test chip through scan IO ports for verification and measurement.

B. Performance Measurement Results

Fig. 16 shows the measured power and frequency with the voltage scaled down to 0.6 V. The nominal supply voltage for both IR mode and RT mode is 0.9 V with 40-mW IR power at 200 MHz and 55-mW RT power at 148 MHz.

In Mobile-PBR, a customized rendering architecture is implemented to accelerate the IR–RT rendering flow. GRTS and GMAC provide the ability for dynamic access and task scheduling for parallel computing processes across the whole

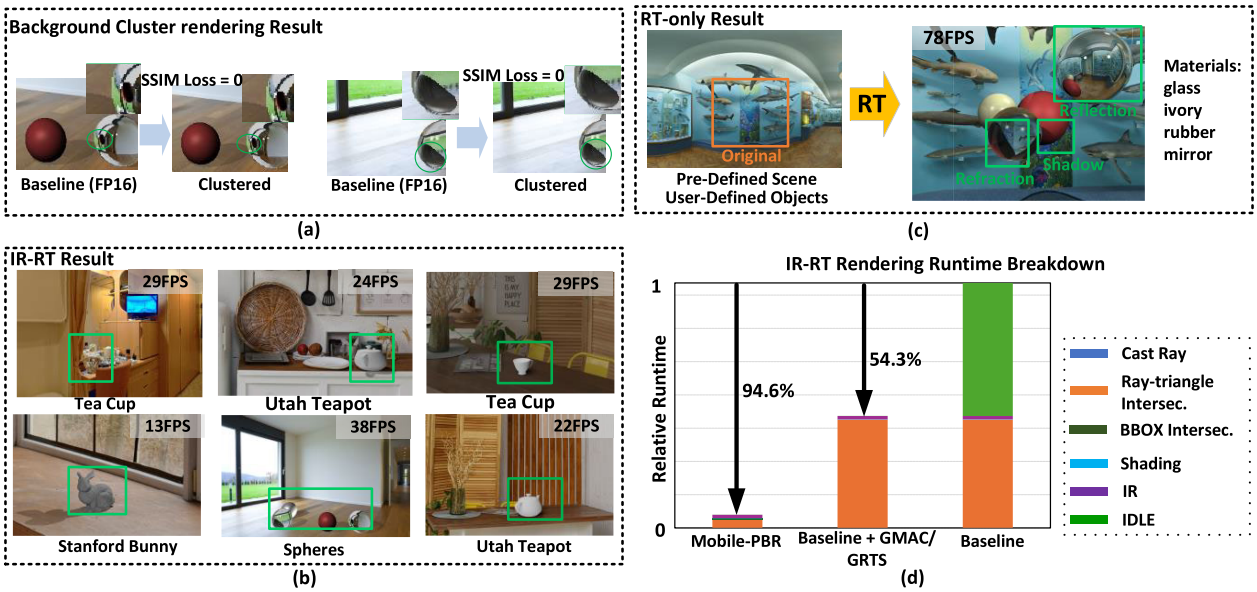


Fig. 19. Rendering results for Mobile-PBR. (a) Example for background cluster rendering result. (b) IR–RT rendering result and FPS performance on different 3-D objects with different materials. (c) RT-only rendering result and FPS performance. (d) Runtime breakdown for Mobile-PBR compared with baseline implementations.

TABLE II
IR–RT RENDER THROUGHPUT WITH DIFFERENT MODELS

Object	Size (Triangle #)	FPS	Object	Size (Triangle #)	FPS
armadillo	99K	11	beetle	2K	66
stanford bunny	69K	13	tie-fighter	54K	19
teapot	6K	23	homer	12K	20
dragon	249K	9	cow	6K	26
fandisk	12K	22	horse	96K	22
rocker-arm	20K	19	spot	6K	24
cheburashka	13K	22	woody	1K	79

* FPS varies between different spatial location and shape of the objects

computing array. An average of 25.8 frames/s is achieved when rendering various scenes. The 500- and 1418-frames/s/W power efficiency has been achieved at 0.9 V for IR and RT modes, respectively.

In the evaluation, floating point (FP) 16-bit results are used as ground truth maps. Fig. 17 shows the average quantization loss results on the IR synthetic dataset by [15] and the runtime breakdown for Mobile-PBR. The IR resolution is 480 × 640. In our experiment and evaluation, we used the same log-encoded loss function on physical maps proposed in [15] to calculate the accuracy drop caused by quantization. Fig. 18 shows two more examples of IR results for normal, albedo, depth, and lighting maps compared with the reference FP16-bit results. There is an average of 0.4%, 6.4%, 8.9%, and 0.9% accuracy loss on lighting, albedo, normal, and depth maps, respectively. No SSIM loss is observed in the final rendered result between the baseline background maps (FP16) and the quantized clustered maps as shown in Fig. 19(a).

C. Rendering Results for IR and RT

More IR–RT results with different objects and materials are shown in Fig. 19(b). The performance of each rendering case is also shown. There are two computing modes: IR and RT. PAMEM needs to be reloaded when switching between IR and RT modes. The reported performance excludes the off-chip MEM reloading time.

As a result, an average of 25.8 frames/s is achieved in the end-to-end IR–RT rendering flow running at maximum frequency for IR and RT modes, meeting the real-time rendering requirement of 24 frames/s [22]. In Fig. 19(c), the result of RT-only mode is shown. In this mode, users need to provide the pre-defined scene and its PA maps. Only RT computation is needed in this case. In RT-only mode, 78 frames/s is achieved when inserting four virtual spheres with different materials and geometries. Photorealistic effects such as reflection, shadow, and refraction are rendered properly in the result. Table II presents more IR–RT renders throughput for more common 3-D objects from [3]. Fig. 19(d) shows the overall on-chip runtime breakdown and improvement of Mobile-PBR and the baseline design.

D. Comparison Results

The comparison results with prior works are shown in Table III and Fig. 20. Mobile-PBR achieves much higher throughput and efficiency compared to the NVIDIA GTX 1080Ti GPU [23] and Intel i7-8665 CPU [24] running the same algorithm. Compared with the prior reconfigurable single instruction multiple threads (SIMTs) processor for mobile RT [6], Mobile-PBR achieves higher throughput with higher power efficiency and can support both DNN and RT operations. Compared with the DNN rendering accelerator designs from [4] and [5], this work has lower rendering throughput but offers better power efficiency in RT tasks and with the support of both DNN and RT modes. In addition, by following the proposed data flow and architecture in this article, Mobile-PBR offers flexible scalability to scale up the number of computing cores with larger silicon implementation with low effort. Mobile-PBR achieves 28.8× and 3.95× higher RT rendering efficiency compared to prior ASIC implementations [4], [5],

TABLE III
COMPARISON TABLE WITH PRIOR WORKS

	Reconf. SIMT [6]	NVIDIA GTX1080Ti	ISSCC'23[4]	ISSCC'24[5]	This Work
Process (nm)	90	16	28	28	28
Area (mm ^{2})	16	471	20.25	20.25	3.56
Architecture	SIMT	SIMT	Systolic Array	Neural-GPU	Systolic Array
Solutions	BVH-Acceleration + Ray-Tracing Process	Nvidia Optix	3D NeRF Rendering	3D modeling + NeRF Rendering	Inverse Rendering + Ray-Tracing Process
Supply Vdd	1.2V	1.0V	0.6-0.95V	0.68-0.9V	0.9V
Clock Frequency	100MHz – 400 MHz	1480MHz	50MHz – 250 MHz	100MHz – 200MHz	142MHz – 200MHz
SRAM	19.3KB	2MB	2MB	2MB	296KB
Bit Precision	FP32	FP32, FP64	FP8-FP16	FP4-FP16	INT8-INT64
Power @ Frequency/Supply	221mW @ 200MHz, 1.2V	250W @ 1480MHz, 1V	310mW @ 100MHz, 0.7V	297.8mW @ 100MHz, 0.77V	40mW @ 200MHz, 0.9V (IR) 55mW @ 142MHz, 0.9V (RT)
Throughput (FPS)	6*	0.75	32 – 118	36.7 – 73.5	13 – 38
Peak Throughput Efficiency (FPS/W)	27.38*	0.003	199 (@0.7V)	123 (@0.77V)	1418 (RT only) 790 (IR and RT)

*Converted from Mray/s

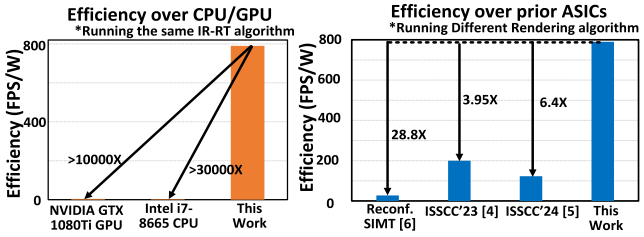


Fig. 20. Efficiency comparison over NVIDIA GTX 1080Ti GPU, Intel i7-8665 CPU, and prior ASICs.

and [6], enabling real-time PBRT on low-cost resource-limited edge devices.

VI. CONCLUSION

This article presented a novel RT rendering flow and scalable architecture, Mobile-PBR, combining the systolic DNN accelerator with individual RT processing cores for end-to-end photorealistic RT rendering tasks on mobile devices. Mobile-PBR is designed based on an 8 × 6 2-D computing array which can be reconfigured to a systolic array in IR mode and parallel processing cores in RT mode. This design achieves 500- and 1418-frames/s/W power efficiency in IR and RT modes, respectively. By implementing the IR–RT-based rendering flow, Mobile-PBR achieves 3.95× and 28.8× higher RT efficiency compared with prior ASIC designs. A test chip was fabricated using 28-nm CMOS technology under a 0.9-V supply voltage, with a 148-MHz operating frequency in RT mode and a 200-MHz operating frequency in IR mode. Mobile-PBR has an average performance of 25.8 frames/s in end-to-end IR–RT rendering tasks and 78 frames/s in RT-only rendering tasks, enabling real-time RT rendering on mobile devices.

REFERENCES

[1] M. Pharr, *Physically Based Rendering*. Cambridge, MA, USA: MIT Press, 2024. [Online]. Available: https://mitpress.mit.edu/9780262048026/physically-based-rendering/

[2] Y. Deng, Y. Ni, Z. Li, S. Mu, and W. Zhang, "Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques," *ACM Comput. Surveys*, vol. 50, no. 4, pp. 1–41, Jul. 2018.

[3] *The Stanford 3D Scanning Repository*. Accessed: May 18, 2024. [Online]. Available: https://graphics.stanford.edu/data/3Dscanrep/

[4] D. Han, J. Ryu, S. Kim, S. Kim, and H.-J. Yoo, "2.7 MetaVRain: A 133 mW real-time hyper-realistic 3D-NeRF processor with 1D-2D hybrid-neural engines for metaverse on mobile devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 50–52.

[5] J. Ryu et al., "20.7 NeuGPU: A 18.5 mJ/Iter neural-graphics processing unit for instant-modeling and real-time rendering with segmented-hashing architecture," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2024, pp. 372–374.

[6] H.-Y. Kim, Y.-J. Kim, J.-H. Oh, and L.-S. Kim, "A reconfigurable SIMT processor for mobile ray tracing with contention reduction in shared memory," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 4, pp. 938–950, Apr. 2013.

[7] *Introducing the NVIDIA RTX Ray Tracing Platform*, NVIDIA Developer, Santa Clara, CA, USA, 2024. [Online]. Available: https://developer.nvidia.com/rtx/ray-tracing

[8] *RealityKit*, Apple Developer Documentation, Apple Inc., Cupertino, CA, USA, 2024. [Online]. Available: https://developer.apple.com/documentation/realitykit

[9] *HoloLens 2 Hardware*. Accessed: May 18, 2024. [Online]. Available: https://learn.microsoft.com/en-us/hololens/hololens2-hardware

[10] *OpenXR Mobile SDK | Oculus Developers*. Accessed: May 18, 2024. [Online]. Available: https://developer.oculus.com/documentation/native/android/mobile-intro/

[11] *BLOG | Samsung Research*. Accessed: May 18, 2024. [Online]. Available: https://research.samsung.com/blog/Galaxy-S23-Series-Realistic-Graphics-Powered-by-Ray-Tracing-Technology

[12] S. G. Parker et al., "OptiX: A general purpose ray tracing engine," *ACM Trans. Graph.*, vol. 29, no. 4, p. 66, 2010.

[13] B. Mildenhall et al., "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Comput. Vis.*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham, Switzerland: Springer, 2020, pp. 405–421.

[14] Y. Zhan, S. Nobuhara, K. Nishino, and Y. Zheng, "NeRFrac: Neural radiance fields through refractive surface," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Paris, France, Oct. 2023, pp. 18356–18366.

[15] Z. Li et al., "OpenRooms: An open framework for photorealistic indoor scene datasets," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 7186–7195.

[16] T. Whitted, "An improved illumination model for shaded display," *Commun. ACM*, vol. 23, no. 6, pp. 343–349, Jun. 1980, doi: 10.1145/358876.358882.

[17] I. Wald, "On fast construction of SAH-based bounding volume hierarchies," in *Proc. IEEE Symp. Interact. Ray Tracing*, Sep. 2007, pp. 33–40.

[18] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Commun. ACM*, vol. 19, no. 10, pp. 547–554, Oct. 1976, doi: 10.1145/360349.360354.
[19] Z. Li, Y. Deng, and M. Gu, "Path compression kd-trees with multi-layer parallel construction a case study on ray tracing," in *Proc. 21st ACM SIGGRAPH Symp. Interact. 3D Graph. Games*. New York, NY, USA: ACM, Feb. 2017, pp. 1–8.
[20] K. Vaidyanathan et al., "Watertight ray traversal with reduced precision," in *High Performance Graphics*. Goslar, Germany: Eurographics Association, 2016, pp. 33–40.
[21] T. Ize. *Robust BVH Ray Traversal (JCGT)*. Accessed: May 18, 2024. [Online]. Available: https://jcgt.org/published/0002/02/02/
[22] X. Min, H. Duan, W. Sun, Y. Zhu, and G. Zhai, "Perceptual video quality assessment: A survey," 2024, *arXiv:2402.03413*.
[23] *GeForce GTX 1080 Ti | Specifications | GeForce*. Accessed: May 18, 2024. [Online]. Available: https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1080-ti/specifications/
[24] *Intel® CoreTM i7-8665U Processor (8M Cache, up to 4.80 GHz) Product Specifications*. Accessed: May 18, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/193563/intel-core-i7-8665u-processor-8m-cache-up-to-4-80-ghz.html

Shiyu Guo (Graduate Student Member, IEEE) received the B.S. degree from Southeast University, Nanjing, China, in 2019. She is currently pursuing the Ph.D. degree in computer engineering with Northwestern University, Evanston, IL, USA.

Her current research interests include computer architecture, graphics processors, and low-power machine learning accelerator design.

Yuhao Ju (Member, IEEE) received the B.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2017, and the M.S. degree from Northwestern University, Evanston, IL, USA, in 2019, where he is currently pursuing the Ph.D. degree in computer engineering.

His current research interests include computer architecture and machine learning accelerator design.

Xi Chen (Student Member, IEEE) received the B.E. degree in electrical engineering from Southeast University, Nanjing, China, in 2018. He is currently pursuing the Ph.D. degree in computer engineering with Northwestern University, Evanston, IL, USA.

His current research interests include power management circuit design and machine learning accelerator design.

Sachin S. Sapatnekar (Fellow, IEEE) received the B.Tech. degree from Indian Institute of Technology at Bombay, Bombay, India, the M.S. degree from Syracuse University, Syracuse, NY, USA, and the Ph.D. degree from the University of Illinois, Champaign, IL, USA.

He taught at Iowa State University, Ames, IA, USA, from 1992 to 1997, and has been at the University of Minnesota, Minneapolis, MN, USA, since 1997, where he holds a Distinguished McKnight University Professorship and the Robert and Marjorie Henle Chair.

Dr. Sapatnekar is a fellow of the ACM. He was a recipient of the nine conference Best Paper Awards, the Best Poster Award, the two ICCAD ten-year Retrospective Most Influential Paper Awards, the SRC Technical Excellence Award, and the SIA University Research Award.

Jie Gu (Senior Member, IEEE) received the B.S. degree from Tsinghua University, Beijing, China, the M.S. degree from Texas A&M University, College Station, TX, USA, and the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA.

He was an IC Design Engineer with Texas Instruments, Austin, TX, USA, from 2008 to 2010, focusing on ultralow-voltage mobile processor design and integrated power management techniques. He was a Senior Staff Engineer with Maxlinear, Inc., Carlsbad, CA, USA, from 2011 to 2014, focusing on low-power mixed-signal broadband SoC design. He is currently an Associate Professor with Northwestern University, Evanston, IL, USA. His research interests include novel circuits and architectures for emerging computing applications.

Dr. Gu was a recipient of the NSF CAREER Award. He has served as a program committee and conference organizer for numerous conferences, such as ISSCC, CICC, ISPLED, DAC, ICCAD, ICCD, and GLSVLSI. He is an Associate Editor of the *Journal of Solid-State Circuits* (JSSC) and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS (TCAS-II).