2.5 A 28nm Physical-Based Ray-Tracing Rendering Processor for Photorealistic Augmented Reality with Inverse Rendering and Background Clustering for Mobile Devices

Shiyu Guo[1], Sachin Sapatnekar[2], Jie Gu[1]

[1]Northwestern University, Evanston, IL
[2]University of Minnesota, Minneapolis, MN

2024 IEEE International Solid-State Circuits Conference (ISSCC) | 979-8-3503-0620-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISSCC49657.2024.10454394

As the applications of Augmented Reality (AR) or Virtual Reality (VR) expand rapidly with the growing demands on enhanced visual realism, photorealistic image generation and insertion has become an essential feature for the emerging AR applications providing real-time workplace/household visual assistance. Physical Based Ray-Tracing (PBRT) 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, soft shadows, etc. PBRT is widely used in product design, medical visualization, video games and movie effects. To enable photorealistic rendering, there is a strong demand to support ray-tracing (RT) on mobile devices [1]. However, the challenges are: (1) unstructured memory access pattern and complex control flow lead to scheduling difficulty; (2) high memory requirements exhaust the limited SRAM space on edge devices; (3) low error tolerance requires high precision for computing; (4) complex computations, such as division and square root, require significant computing resources for the edge devices. As a result, common rendering engines such as Apple ARKit, OpenGL, are mainly based on the lower cost rasterization rendering technique. Unfortunately, rasterization rendering fails to produce photorealistic synthesis as shown in Fig. 2.5.1. Few ASICs have been fabricated so far as a mobile photorealistic rendering solution solution, however, they may not support RT [2], or may suffer from low efficiency [3]. This work has developed a ray-tracing processor, which also supports inverse rendering (IR) for background extraction [4]. The key features of this work include: (1) an ASIC rendering processor that embeds an end-to-end PBRT solution with IR for AR on mobile devices, (2) a reconfigurable mixed-precision PE design supporting diverse computing tasks for both IR and RT, (3) background clustered Field of View (FOV)-focused 3D construction reducing conventional background scene complexity from O(nlogn) to O(1), (4) scalable partitioning scheme for complex 3D objects, with an average of 13× speed up on test scenes, (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 run-time with overall 684× speedup compared with the baseline design. The 28nm test chip achieves 3.95-28.8× higher rendering efficiency compared with existing ASIC solutions, enabling real-time PBRT rendering on mobile edge devices.

Figure 2.5.2 illustrates the computing flow of this work. In the first step, a 2D image captured by a regular camera is sent through a CNN-based physical decoder and encoder for IR to obtain the background physical attributes, including four major background physical attribute (PA) maps: albedo, normal, lighting and depth maps. To save the compressed PA map on chip, a background clustering scheme is developed based on the similarity of neighbor pixel values of the background map by applying an average filter. The result PA maps are stored in a Physical Attributes MEM (PAMEM). Each PE accessing the PAMEM passes through the Per Pixel Compression Decoder (PPCD) and the Unified Address Converter (UAC) to fetch the corresponding background physical attributes parameter based on the PE task ID from the GRTS scheme. In this way, 145-4800× of memory saving for different backgrounds is achieved with 0.06% hardware overhead compared with the baseline design. In the second step, the IR result is used for Camera FOV-focused 3D construction. As shown in Fig. 2.5.2, the background scene is constructed only for the 3D space covered by the user camera FOV. In this way, the background scene complexity is reduced from O(nlogn) to O(1) compared with conventional RT solutions (n refers to the number of geometric primitives in the scene) [5]. In the last step, PBRT rendering is implemented to render 3D virtual objects with an average of 76% RT workload reduction compared with the conventional RT solutions as shown in Fig. 2.5.2.

Figure 2.5.3 shows the top-level architecture of this design. To increase the utilization of the PE and address the irregular access pattern to memory, an 8×6 PE array is implemented with a GRTS and a GMAC. To support computation for both IR and RT modes, a reconfigurable mixed-precision PE is developed as shown in Fig. 2.5.3. Each PE contains a local PE Controller, clock-gating control, a computing core which supports 8b, 16b and 32b MAC, 64b division and 64b square root operations and a local OBJMEM. Clock gating disables excessive computing units and local MEM in IR and RT modes with 32% power saving. In RT mode, a scalable 3D model partitioning flow shown in Fig. 2.5.3 is implemented. In contrast to the conventional solution that builds the BBOX acceleration [6], this work introduces two types of object Bounding Box (BBOX): Empty BBOX (EBBOX) and Target BBOX (TBBOX). EBBOX is only used for light transportation estimation and shadow purposes, while the shading computation is skipped. TBBOX includes a sub-group of user-defined objects inside. After the BBOX Intersection Evaluator (BBIE) detects intersection with TBBOX, the Triangle Mesh Intersection Evaluator (TIE) computes the triangle intersection, and the result is sent for shading computing. With this scheme, complex 3D objects could be segmented for RT processing without losing the ray-tracing effect. As a result, a linear scalability in RT rendering time and an average of 13× speed up with only 5.6% memory overhead is achieved compared with the baseline design.

Figure 2.5.4 shows on-chip data movement in IR and RT modes. In IR inference mode, double input and weight stationary are supported. In RT mode, multiple PEs have memory access conflicts, as shown in Fig. 2.5.4. To address the global PAMEM access conflict and varied PE run-time, GRTS and GMAC are implemented. With the RT Token Checker (RTTC) checking one PE status every clock cycle, GRTS and GMAC process the RTTC selected PE request individually while GRTS refreshes the checked PE status if the computation is done. In this way, we achieve 42.8× overall speed up from GRTS and 16× overall speed up from GMAC compared with the baseline design by introducing only 2.8% and 0.6% hardware cost. The detailed RT shading algorithm to compute the color of each pixel is also shown in Fig. 2.5.4. RT shading demands complex operations, such as sqrt and division. In this work, a PE Compute Unit (PCU) is implemented inside each PE. As shown in Fig. 2.5.4, PAMEM, OBJMEM data are sent to PCU for shading computation. PCU's output is stored in a local shading register for lighting effect accumulation. By implementing a PCU in each PE, all the RT computation can be finished individually inside each PE.

Figure 2.5.5 shows the total runtime breakdown for the IR-RT flow and demonstrates the background clustering scheme by showing the background reflection is able to light up the object properly by using clustered PA map compared with the baseline design with the detailed background PA map. Four virtual object insertion test cases with teacup, Utah teapot, Stanford bunny and four spheres are demonstrated in Fig. 2.5.5 using the IR-RT rendering scheme. Different materials, such as glass, mirror and ivory are displayed with photorealistic rendering effects of reflection, refraction and shadow. The IR-RT flow achieves an average of 26fps for real-time RT rendering with complex 3D objects. Figure 2.5.5 also showed the 3D rendering case without IR with a predefined 3D background. Four spheres with different materials are inserted. Photorealistic effects of refraction, reflection and object shadow are properly rendered to the image with 78fps, meeting the requirement of real-time AR applications.

A 28nm test chip was fabricated with 0.9V nominal supply voltage. Figure 2.5.6 shows more measurement and comparison results. Power and frequency scaling are shown in Fig. 2.5.6 with a supply voltage from 0.6V to 0.9V. 500fps/W and 1418fps/W power efficiency has been achieved at 0.9V for IR and RT modes, respectively. The comparison table with prior art is provided in Fig. 2.5.6. This work implements IR-RT-based rendering solutions, achieving 28.8× and 3.95× higher ray-tracing rendering efficiency compared with prior ASIC designs, enabling real-time PBRT on mobile edge devices. Figure 2.5.7 shows the die photo and chip specifications.

Acknowledgement:
This work is supported in part by AFRL under the DARPA RTML program under award FA8650-20-2-7009 and NSF grant CCF-2008906.

References:

[1] Y. Deng et al., "Toward Real-Time Ray Tracing: A Survey On Hardware Acceleration and Microarchitecture Techniques," *ACM Computing Surveys*, vol. 50, no. 4, pp. 1–41, 2017.
[2] D. Han et al., "MetaVRain: A 133mW Real-Time Hyper-Realistic 3D-NeRF Processor with 1D-2D Hybrid-Neural Engines for Metaverse on Mobile Devices," *ISSCC*, 2023.
[3] H.-Y. Kim et al., "A Reconfigurable SIMT Processor for Mobile Ray Tracing With Contention Reduction in Shared Memory," *IEEE TCAS-I*, vol. 60, no. 4, pp. 938–950, 2013.
[4] Z. Li et al., "Openrooms: An Open Framework For Photorealistic Indoor Scene Datasets," *IEEE CVPR*, pp. 7190–7199, 2021.
[5] S. G. Parker et al., "OptiX: A General Purpose Ray Tracing Engine," *ACM Trans. Graph.*, vol. 29, no. 4, p. 66:1-66:13, 2010.
[6] I. Wald, "On fast Construction of SAH-based Bounding Volume Hierarchies," *IEEE Symp. on Interactive Ray Tracing*, pp. 33–40, 2007.

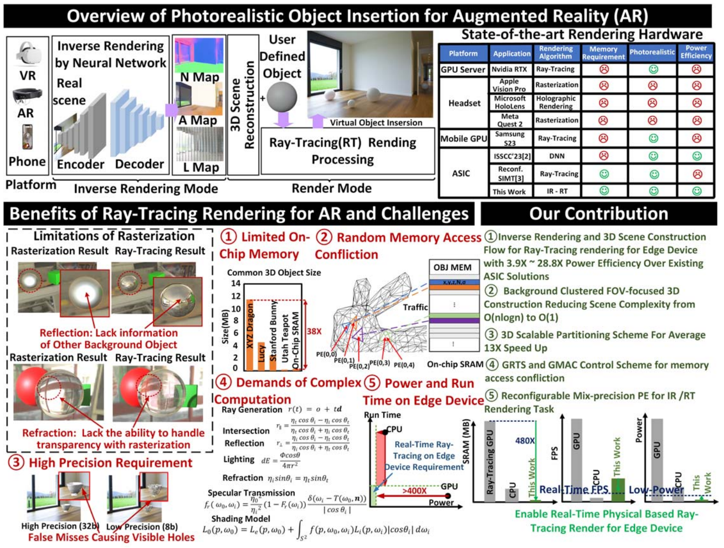

Overview of Photorealistic Object Insertion for Augmented Reality (AR)
VR
AR
Phone
Platform
Inverse Rendering by Neural Network
Real scene
Encoder
Decoder
N Map
A Map
L Map
Inverse Rendering Mode
3D Scene Reconstruction
User Defined Object
Virtual Object Insersion
Ray-Tracing(RT) Rending Processing
Render Mode
State-of-the-art Rendering Hardware
Platform
Application
Rendering Algorithm
Memory Requirement
Photorealistic
Power Efficiency
GPU Server
Headset
Mobile GPU
ASIC
Benefits of Ray-Tracing Rendering for AR and Challenges
Limitations of Rasterization
Rasterization Result
Ray-Tracing Result
Reflection: Lack information of Other Background Object
Refraction: Lack the ability to handle transparency with refraction
③ High Precision Requirement
High Precision (32b)
Low Precision (8b)
False Misses Causing Visible Holes
① Limited On-Chip Memory
② Random Memory Access Conflict
④ Demands of Complex Computation
⑤ Power and Run Time on Edge Device
Real-Time Ray-Tracing on Edge Device Requirement
Real-Time FPS
Low-Power
Our Contribution
① Inverse Rendering and 3D Scene Construction Flow for Ray-Tracing rendering for Edge Device with 3.9X ~ 28.8X Power Efficiency Over Existing ASIC Solutions
② Background Clustered FOV-focused 3D Construction Reducing Scene Complexity from O(nlogn) to O(1)
③ 3D Scalable Partitioning Scheme For Average 13X Speed Up
④ GRTS and GMAC Control Scheme for memory access conflication
⑤ Reconfigurable Mix-precision PE for IR /RT Rendering Task
Enable Real-Time Physical Based Ray-Tracing Render for Edge Device

Figure 2.5.1: Overview of the photorealistic object insertion flow for AR applications. Limitation of rasterization. Challenges for ray-tracing rendering and contributions of this work.

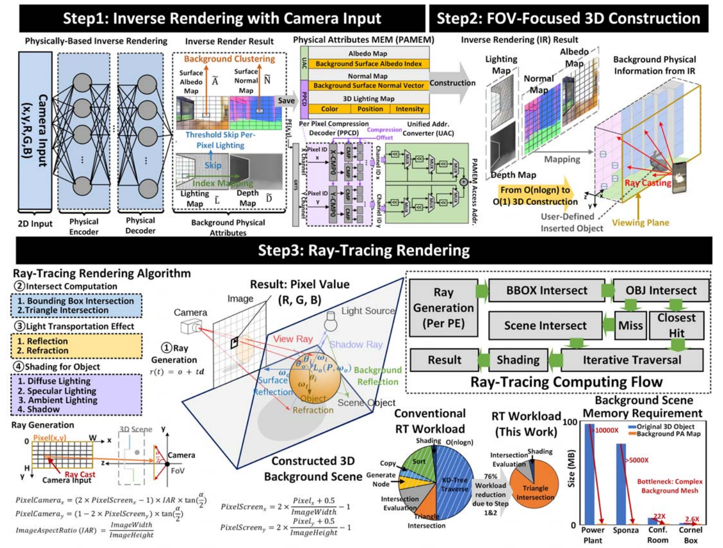

Step1: Inverse Rendering with Camera Input
Step2: FOV-Focused 3D Construction
Physically-Based Inverse Rendering
Camera Input (x,y,R,G,B)
2D Input
Physical Encoder
Physical Decoder
Inverse Render Result
Background Clustering
Physical Attributes MEM (PAMEM)
Per Pixel Compression Decoder (PPCD)
Unified Addr. Converter (UAC)
Inverse Rendering (IR) Result
Background Physical Information from IR
From O(nlogn) to O(1) 3D Construction
User-Defined Inserted Object
Viewing Plane
Ray Casting
Step3: Ray-Tracing Rendering
Ray-Tracing Rendering Algorithm
Result: Pixel Value (R, G, B)
Camera
Light Source
Constructed 3D Background Scene
Ray Generation (Per PE)
BBOX Intersect
OBJ Intersect
Scene Intersect
Miss
Closest Hit
Result
Shading
Iterative Traversal
Ray-Tracing Computing Flow
Conventional RT Workload
RT Workload (This Work)
Background Scene Memory Requirement

Figure 2.5.2: Inverse Rendering (IR) and Ray Tracing (RT) rendering flow with background physical attributes and camera field of view (FOV)-focused 3D construction.

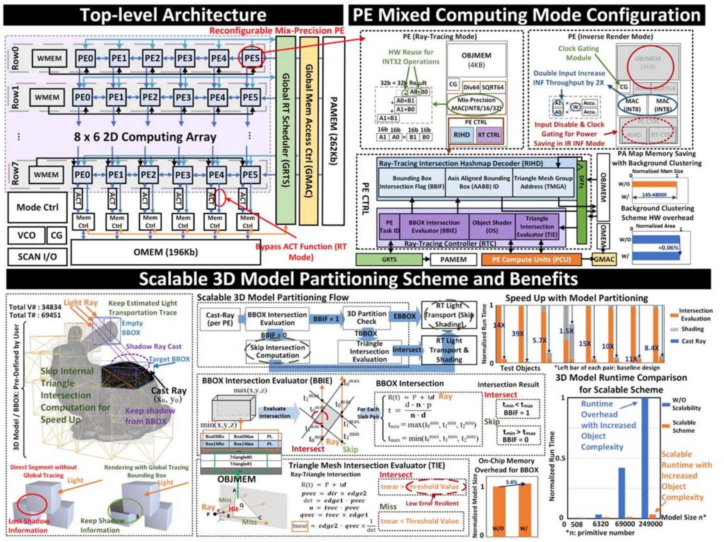

Top-level Architecture
Reconfigurable Mix-Precision PE
8 x 6 2D Computing Array
Global RT Scheduler (GRTS)
Global Mem Access Ctrl (GMAC)
PAMEM (262Kb)
OMEM (196Kb)
Mode Ctrl
VCO
CG
SCAN I/O
Bypass ACT Function (RT Mode)
PE Mixed Computing Mode Configuration
PE (Ray-Tracing Mode)
PE (Inverse Render Mode)
Scalable 3D Model Partitioning Scheme and Benefits
Scalable 3D Model Partitioning Flow
Speed Up with Model Partitioning
BBOX Intersection Evaluator (BBIE)
Triangle Mesh Intersection Evaluator (TIE)
3D Model Runtime Comparison for Scalable Scheme
Scalable Runtime with Increased Object Complexity

Figure 2.5.3: Top-level chip architecture of this design. Reconfigurable mixed-precision PE architecture and scalable 3D model partitioning scheme. BBOX intersection evaluator and triangle mesh intersection evaluator.

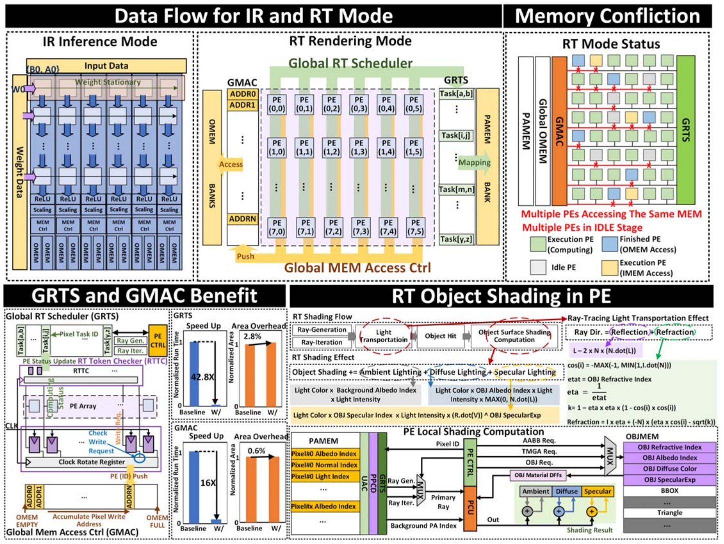

Data Flow for IR and RT Mode
IR Inference Mode
RT Rendering Mode
Global RT Scheduler
Global MEM Access Ctrl
Memory Conflication
RT Mode Status
Multiple PEs Accessing The Same MEM
Multiple PEs in IDLE Stage
GRTS and GMAC Benefit
RT Object Shading in PE
RT Shading Flow
RT Shading Effect
Object Shading += Ambient Lighting + Diffuse Lighting + Specular Lighting
PE Local Shading Computation

Figure 2.5.4: Global Ray-Tracing Scheduler (GRTS) and Global Mem Access Controller (GMAC) scheme and benefits. Ray tracing object shading computation in each PE.

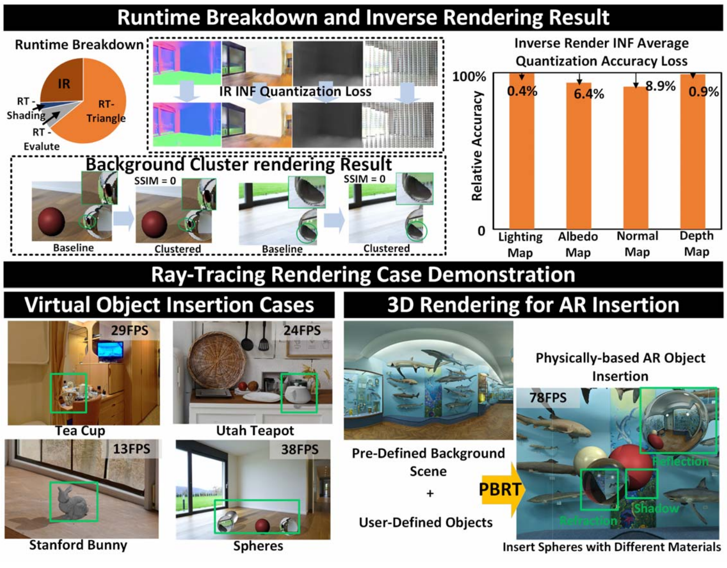
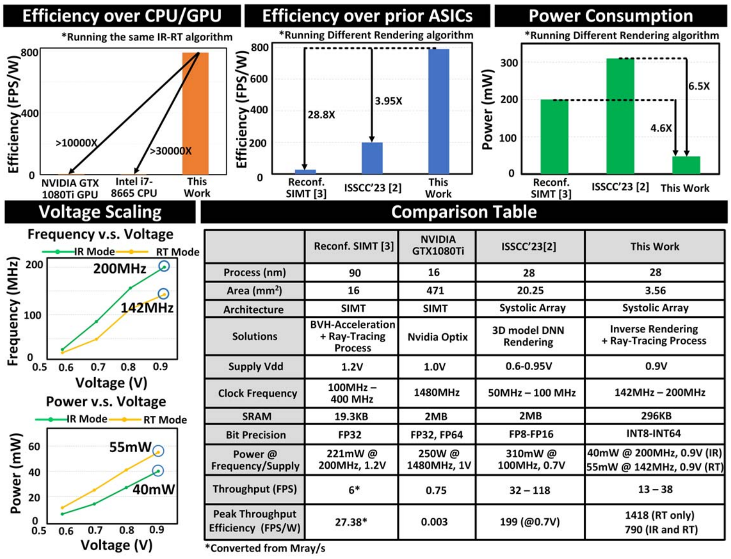


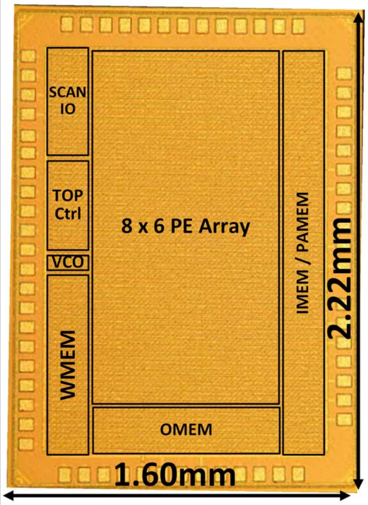
Figure 2.5.5: Inverse rendering quantization accuracy loss. Examples of virtual object insertion with IR-RT flow and 3D rendering for AR object insertion.



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

*Converted from Mray/s

Figure 2.5.6: Measurement results and comparison table with prior work.



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

Figure 2.5.7: Die micrograph.