Software-Hardware Codesign of Ray-Tracing Accelerator for Edge AR/VR with Viewpoint-Focused 3D Construction and Efficient Data Structure

Shiyu Guo[1], Sachin S. Sapatnekar[2], Jie Gu[1]
[1]Northwestern University, Evanston, IL – Email: shiyuguo2021@u.northwestern.edu, jgu@northwestern.edu
[2]University of Minnesota, Minneapolis, MN – Email: sachin@umn.edu

2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS) | 979-8-3503-8717-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MWSCAS60917.2024.10658949

***Abstract*— Physics Based Ray-Tracing (PBRT) rendering is a process to generate synthesized images by simulating real environment using the spatial, reflection, refraction, diffusion information from the scene to achieve photorealism. Although PBRT is widely used in architecture, video games, movie effects, and VR/AR applications, the ray-tracing technique in PBRT for realistic modeling of light transport suffers from major difficulties on resource-limited edge devices due to its overwhelming workload of computing and irregular memory access pattern limiting to its real-time usage only on the state-of-the-art GPUs. In this paper, we proposed a hardware acceleration solution which incorporates novel techniques including customized hardware data structure acceleration, scene background information clustering, 3D Construction and adaptive mix-precision computing scheme leading to a low-cost ray-tracing rendering solution suitable for edge devices. The result design has been implemented in 28nm CMOS technology. Experimental results on open-source dataset show a 6X reduction of computing workload, 56X saving of background memory requirements and 33% power saving compared with baseline design, enabling hardware adaptation of PBRT accelerator on a resource-limited edge device.**

Keywords*— *Physics Based Ray-tracing Rendering, Edge Devices, AR/VR, Virtual Object Rendering, Software-hardware codesign

I. Introduction

Physics based ray-tracing [1] is a photorealistic rendering technique in computer graphics to generate computer-synthesized image from a 3D space. This technique works by tracing the light transportation bouncing off all the surface, which makes it capable of simulating various photorealistic effect such as reflection, refraction, soft shadows, scattering, depth of field, motion blur, caustics, ambient occlusion, and dispersion phenomena. To address the overwhelming workload, much work has been performed recently to accelerate ray-tracing rendering tasks using software approaches or general-purpose hardware processors, such as NVIDIA Optix [2], Intel Embree [3], OpenGL [4], MIC [5], NVIDIA RTX GPUs, etc.

As the prevalence of mobile devices supporting Augmented Reality (AR) and Virtual Reality (VR) applications continues to rise, there is an increasing interest in improving the visual realism of the image scene for enhanced user's experience. However, there are significant challenges: (1) Existing AR techniques based on rasterization rendering cannot replicate the same visual effect as a physical lens, thus failing to blend into the scene [7][10]. (2) Current AR solutions face the compute bottleneck of tracking and recognizing the real-world objects and environment because the tracking of the device's position and orientation need to be high precision for an object to be inserted properly. (3) The memory overhead for complex 3D scenes limits the usage of photorealistic rendering algorithm on the edge devices.

To address the above challenges for high-quality AR/VR applications, recent works have been integrating photorealistic ray-tracing techniques to high-quality AR applications [8]. Unfortunately, most prior works are implemented on GPCPU and GPGPU platforms aiming for the best rendering quality, which consume too much power and memory for resource-limited edge devices.

In this work, a power efficient ray-tracing solution is proposed to address the challenges of photorealistic ray-tracing tasks on edge devices. Fig. 1 shows the ray-tracing task for photorealistic AR/VR applications. The contributions from this wok include: (1) Software-hardware codesign of a customized Bounding Volume Hierarchy (BVH) control scheme and efficient data structure are implemented with more than 6X speed up on various test scenes; (2) Software-Hardware codesign of a 3D viewpoint based background clustering scheme is implemented with more than 56X memory saving on the background maps and is able to reduce background traversal complexity with negligible image quality loss; (3) A mix-precision computing scheme is implemented in different rendering stage for the optimal cost-quality trade-off for the rendering task, which leads to 21% reduction of the power compared with the baseline design with minimum image quality loss. At last, a 28nm ASIC is implemented, achieving 41.6 MRay/s with 69.5mW, leading to 28~44X higher power efficiency than existing ASIC solutions and 22X power efficiency then the state-of-the-art mobile GPU for the ray-tracing rendering task.

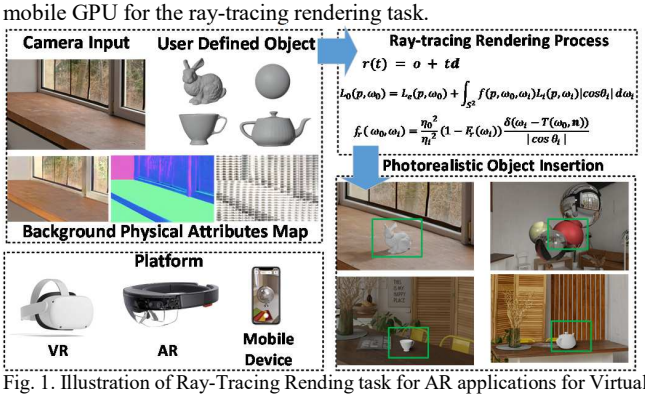

Fig. 1. Illustration of Ray-Tracing Rending task for AR applications for Virtual Object insertion

II. Background And Challenges

A. Ray-Tracing Rendering

Ray-tracing is a technique often used in computer graphics to generate a synthesized 2D image, by extending camera rays into a constructed 3D scene, performing the actual light ray transportation and calculating the approximate value of the pixels in the final rendered images.

Traditional rendering engines and applications are mainly based on low-cost rasterization technique, such as Microsoft Windows Mixed Reality Toolkit (MRTK), OpenGL[4], Apple ARKit and Google

ARCore. Different from rasterization technique, ray-tracing is a photorealistic rendering technique which will simulate the transportation of light rays in a 3D scene. Fig. 2(a) shows the ray-tracing algorithm. By considering all the light transportation on all the object and background surface, photorealism could be achieved. Fig. 2(b) shows the difference between ray-tracing and rasterization, light transportation effects such as reflection and shadow are hard to achieve in rasterization rendering.

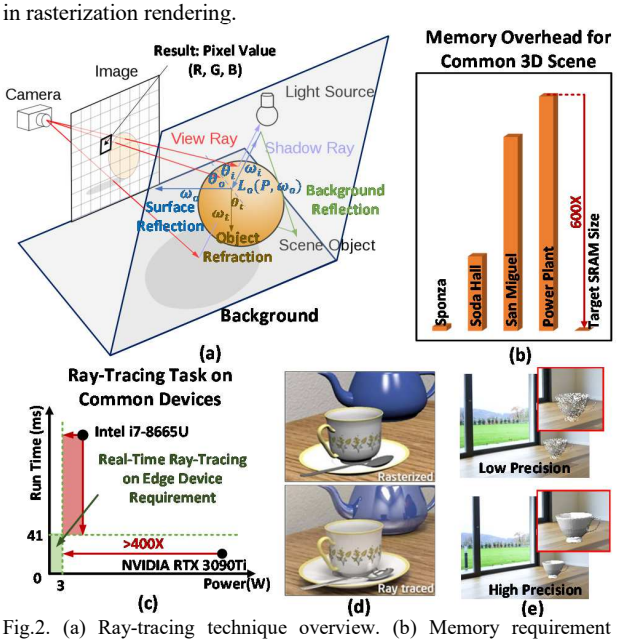

Fig.2. (a) Ray-tracing technique overview. (b) Memory requirement for common 3D mesh scenes compared with limited on-chip memory size. (c) Real-time ray-tracing requirement for edge devices. (d) Difference between ray-tracing and rasterization rendering. (e) Render results with low and high computing precisions.

Mathematically, ray-tracing algorithm is solving the rendering equation in (1) for every specific scene and ray casted by user camera:

$$L_o(P,\omega_o) = \int_{S^2} f(P,\omega_o,\omega_i) L_i(P,\omega_i) |cos\theta_i| \, d\omega_i \quad (1)$$

As shown in Fig. 2(a), L_0 is the radiance leaving at object point P along the direction ω_o; f is the Bidirectional Reflectance Distribution Function (BRDF) of the object surface, which takes this incoming light direction ω_i and outgoing direction ω_o at point P, returning the ratio of reflected radiance along ω_o to the irradiance incident on the surface from direction ω_i; L_i is the incoming light along direction ω_i, θ_i is the angle between ω_i and surface normal. The final pixel value is calculated by simulating the transportation of the light ray and integrating all the effects. Thus, ray-tracing is able to offer true ability for photorealistic image synthesis and thus enhance the visual quality of AR/VR applications [10][11].

B. *Hardware Design Challenges*

Modern processors such as GPUs are mostly used for ray-tracing rendering tasks, but the power consumption is far beyond the ability of resource-limited edge devices. For example, the power consumption for Apple A15 Bionic chip is only 3.9W under modest workload, while NVIDIA RTX 3090Ti consumes 398W under average workload. With mobile devices rapidly becoming the central device in people's daily life, it is crucial to develop low-power real-time rendering solutions on these resource-limited devices for mobile AR applications as shown in Fig. 2(c). At the same time, ray-tracing algorithm is well-known for its intense computing requirement and low error tolerance as shown in Fig. 2(e). As a result, most ray-tracing engines, such as NVIDIA Optix, support double-precision floating point computing precision, which results in higher cost and power, making it difficult to implement on resource-limited devices. Unstructured memory access pattern is also a bottleneck in ray-tracing applications. Each cast ray will evaluate intersections for every mesh primitive, which requires access to the complete object with unstructured requests. With the objects growing more and more complex shown in Fig. 2(b), it is impractical to save the complete object on chip.

There have been missing solutions for ray-tracing rendering on resource-limited edge devices. In this paper, a novel fixed-point power efficient hardware ray-tracing accelerator is implemented with 28nm CMOS technology, with customized scalable on-chip BVH acceleration data structure, viewpoint-focused background scene clustering for both power and memory saving and a mix-precision computing scheme to achieve the optimal cost-quality trade-off for the rendering tasks. A 22~44X higher power efficiency is achieved compared with GPGPU and state-of-the-art mobile GPU.

III. Proposed Ray-tracing Render on Edge Devices

In this section, a ray-tracing accelerating design is proposed with three distinct customized techniques through software-hardware codesign to address the current challenge in ray-tracing render on edge devices, including hardware data structure design for intersection acceleration, viewpoint-focused background clustering for memory saving and mix-precision computing for optimal power-quality trade-off. The performance and quality of the proposed techniques are evaluated in the next section.

A. *Bounding Volume Hierarchy (BVH) Control Scheme*

The main bottleneck for ray-tracing objects is intersection evaluation. As shown in Fig. 3(a), the 3D object is defined using a collection of triangle meshes. To render an object, each triangle is evaluated for ray-intersection. Increasing the number of triangles enhances the smoothness of the object surface, however, it also increases additional overhead in terms of memory storage and computing time. Without any GPU or multicore acceleration, rendering an opensource Stanford bunny object with 69451 triangle primitives takes 2.5 hours on Intel® Core™ i7-8665U CPU for the image resolution of 640×480, with more than 90% of time performing exhaustive intersection search. To achieve faster ray-tracing process, faster intersection checking process is needed. In this work, we implement the concepts of Bounding Volume Hierarchy (BVH) inference and Axis-Aligned Bounding Box (AABB) on a column-wise ASIC controller to speed up triangle intersection checks. As shown in Fig. 3(a), AABB is a 3D bounding box for all the triangles inside it. Each node in a BVH tree represents a bounding box. Algorithm 1 shows the traversal algorithm AABB intersection evaluation. AABB evaluation starts from the root node and proceeds with the closest hit for AABB until reaching the leaf node.

As shown in Fig. 3(b), the root Node N1 holds the bounding box geometry of the entire object, and it has two child nodes at the next layer which will store the bounding box of the sub-level primitives. The rays are cast from the camera viewpoint and evaluated for AABB intersection layer by layer. A Column BVH Controller is implemented in each RU computing logic that will be able to accelerate all the tree structure following the user-defined data format shown in Fig. 4(d). By introducing BVH in the design before the triangle intersection stage, large portions of the object can be quickly eliminated from the evaluation list, thus speeding up the whole rendering process. As Fig. 3(c) shows, the final hardware implementation integrates the performance optimization on the algorithm level and the resources optimization on the silicon level to generate the final ASIC implementation for a resource-limited edge device.

The top-level architecture of the proposed design is shown in Fig. 4(a). Render Unit (RU) is the main computing unit specifically designed for ray-tracing rendering applications. The detail of each RU is shown in Fig. 4(b), RT Controller inside each RU is used to navigate the computing through different stages shown in Fig. 4(d). Local Mem in

each RU is used to store the physical attributes of the rendering background and user-defined object from each column BVH MEM. Different from the conventional approaches for multi-core CPU (e.g., OpenMP) or GPGPU (NVIDIA GPU acceleration), a BVH traversal acceleration along each RU column to save hardware cost while ease software scheduling overhead. Fig. 4(d) shows the rendering executing process within each individual RU. When the RU is running AABB evaluation, the memory access pattern is irregular and unpredictable because the scene and object geometries are not determined until the rendering process started. Each Column BVH CTRL (CBC) is used to address the memory access conflict within each RU column. After AABB and triangle intersection evaluation, RU returns shading value as the pixel value.

B. *Background Scene Clustering*

In photorealistic AR/VR applications, it is necessary to obtain background physical attributes for ray-based effects, such as refraction, reflection, and shadows. In conventional solutions, complete 3D scene files are provided with a traversal complexity of O(nlogn) [6]. N is the number of triangles in the given 3D scene. It is too expensive to store a complete 3D background mesh on resource-limited devices. In this work, a viewpoint focused background scene clustering technique is developed and tested to simplify the background. With the 2D physical attribute map for background scene, the 3D physical scene can be rebuilt with little effort.

As shown in Fig. 5, the indoor scenes have the feature of clustered pixels in the same plane with similar surface normal value (e.g. wall, table, floor), and there is a great dissimilarity between different planes and the materials. In the proposed scheme, albedo and normal maps are clustered using Simple Linear Iterative Clustering (SLIC) and saved on chip with an efficient data structure shown in Fig. 4(c). The proposed background clustering scheme encodes a 3D complex background scenes in 2D representations, introducing the background geometry and material effects into the rendering process with minimal cost and runtime.

Algorithm 1 BVH Traversal

```
while true do
  if NodeisLeaf then
    for i < triCount do
      IntersectTri(ray, tri[node → LeftFirst + i])
      if stackPtr == 0 then
        break
      else
        node = stack[–stackPtr]
      end if
    end for
    dist1 = IntersectAABB(ray, LNode)
    dist2 = IntersectAABB(ray, RNode)
    if dist1 > dist2 then
      swap(LNode, RNode)
      if LNodeMiss then
        if stackPtr == 0 then
          break
        else
          node = stack[–stackPtr]
        end if
      else
        Node = LNodeMiss
        if RNodeIntersect then
          stack[–stackPtr] = RNodeIntersect
        end if
      end if
    end if
  end if
end while
```

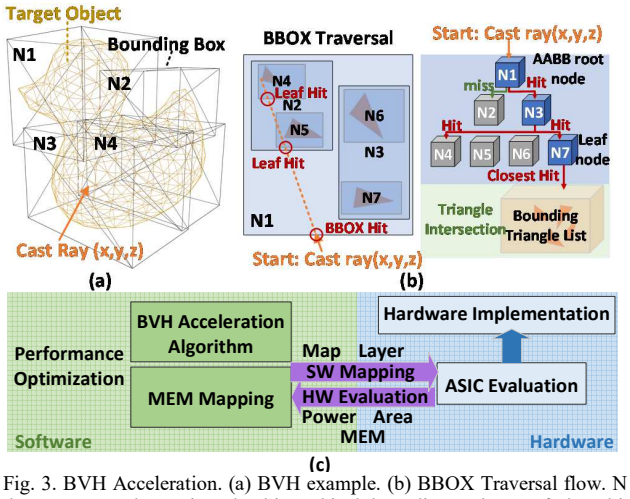

Fig. 3. BVH Acceleration. (a) BVH example. (b) BBOX Traversal flow. N is the AABB node storing the hierarchical bounding volume of the object triangles, when the ray hit the leaf node, it will perform triangle intersection in the leaf AABB. (c) Software-hardware codesign flow of the proposed acceleration scheme.

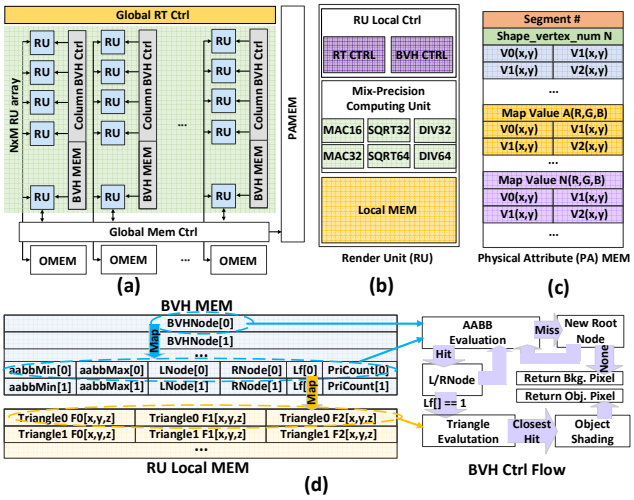

Fig. 4. Overall block diagram for the render design and memory mapping. (a) Design top-level block diagram. The main component of the design is a N×M Rendering Unit (RU) array with BVH structure. (b) Block diagram for a single Rendering Unit. (c) Memory mapping in Physical Attribute MEM. (d) BVH control flow and column MEM mapping.

C. *Mix-precision Computing: cost and quality trade-off*

Ray-tracing is a computing intensive task with extremely low error tolerance. Previous work has been done to implement reduced precision architecture on high-end GPU, however, this solution may not be feasible for certain GPGPUs that need to support rasterization due to their inherent nature of general-purpose computing workloads and hardware limitation [14][15]. In this work, a mix-precision hardware scheme is implemented with the optimal power-quality trade-off in the proposed designs.

Fig. 6(a) shows the relative power consumption comparison among various combinations of mix-precision settings. Each design is named after the fixed-point precision of the critical computing units within the RU: mac, sqrt and division. For example, 24b24s24d represents 24bit mac with 24bit sqrt and 24bit division. As shown in Fig. 6(a), as the precision reduces, the power and area cost will be lower, however, lowering precision may cause rendering failure during intersection computing. In the evaluation section, we added render image quality as

an evaluation metric. 8 different combinations of design with proposed mix-precision scheme, are evaluated. By considering all the factors that contribute to the final rendering result, the optimal tradeoff in the power-quality space is achieved.

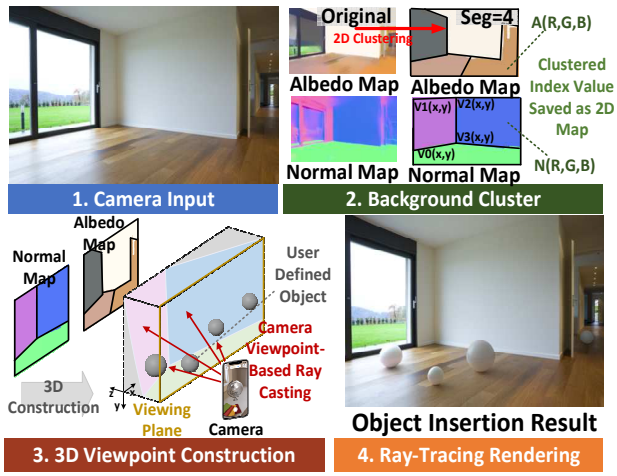

Fig. 5. Proposed ray-tracing flow with background cluster and 3D construction.

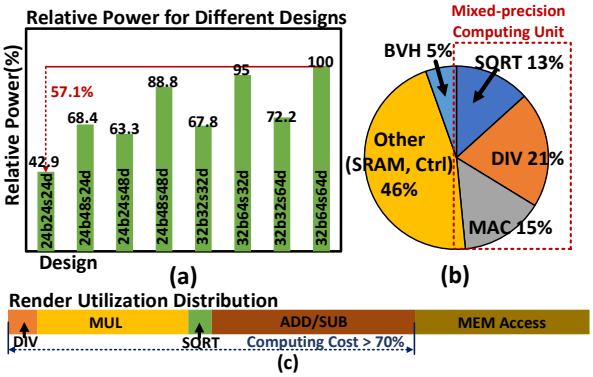

Fig. 6. Power and Area cost break down. (a) Relative power consumption for different combination of the design. (b) Area cost for different hardware components for each RU. (c) Rendering stages distribution in each RU.

IV. Evaluation And Results

This section shows the evaluation method and the experiment result for the proposed schemes.

A. BVH Acceleration Evaluation

As Fig. 7(a) shows, 7 common complex 3D objects are tested:

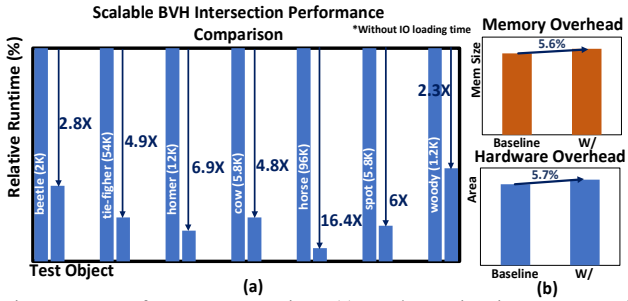

Fig. 7. BVH Performance Comparison. (a) Render runtime improvement with BVH acceleration on different testing objects. (b) Memory and hardware overhead for BVH acceleration hardware.

By incorporating BVH structure into the design, an average of 6.3X speed up with only 5.7% average area overhead and 5.6% increase in memory storage for the hierarchical AABB are achieved compared with the baseline design.

B. Background Scene Clustering and Image Quality Evalutation

In photosynthesis, objective comparison between pixel-level value between the testing image and the reference image sometimes does not match human observations of the image [12]. As a result, Deep Image Structure and Texture Similarity (DISTS) [13] metric is used in our evaluation to quantitively evaluate the rendering result. DISTS is based on human vision system and provides a more accurate measure than other image metrics Mean Squared Error (MSE), which only considers the pixel-level differences. DISTS offers the ability to capture and measure the structural and texture similarity of the original image and the "degraded" image.

The original raw RGB albedo and normal map representing background information with the resolution of 640×480 is 921KB. With the proposed efficient data structure, only 16KB is needed to store the background albedo or normal map. Hence, 56X reduction of memory storage is achieved, making it possible to store the background scene on the chip without accessing off-chip memory, which makes it possible for real-time end-to-end rendering tasks.

C. Hardware Evaluation, Cost-Quality Trade-off for Mix-Precision Scheme

To find out the optimal tradeoff in cost-quality space, we proposed a customized matric for evaluation:

$$RenderIndex(p) = Norm(DISTSLoss(p)) \times Norm(Power(p)) \times Norm(Area(p))$$

p represents a combination of hardware configuration. RenderIndex(p) represents the normalized cost value of each design p. DISTSLoss(p) represents the DISTS index loss of the rendering result for design p. Power(p) and Area(p) represent the normalized power and area of design p. In the testing scene, 8 different objects are tested for the rendering task with different hardware configurations. Fig. 8(a) shows the quantitative rendering results for different 3D objects. As shown in Fig. 8(b), RenderIndex(32b32s64d) has the lowest value, which represents the optimal cost-quality among all design configurations.

Design 32b32s64d is implemented. Compared with the baseline scheme, the proposed implementation achieves 16.7% of area saving and 33.4% of power saving with only 0.6% to 1.65% loss.

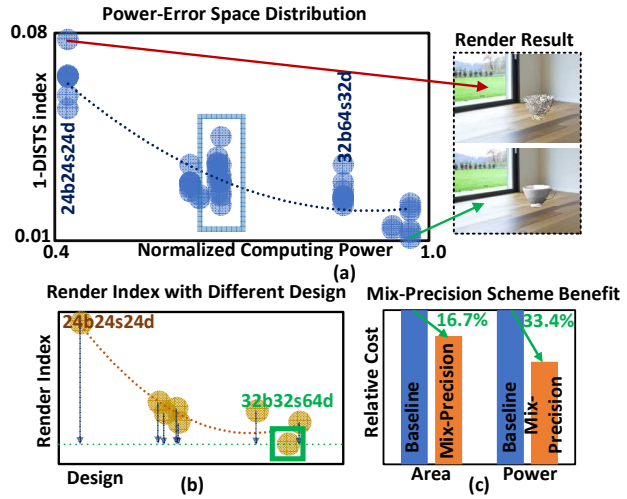

Fig. 8. Result for mix-precision computing. (a) Illustration of power-error tradeoff rendering and image visual quality. (b) Average Render Index after

testing 8 Objects: armadillo, chebura, fandisk, rocker-arm, stanford bunny, Utah teapot, dragon, and duck. (c) Area and power saving for mix-precision computing scheme compared with baseline design.

D. Design Implementation and performance

A mixed-precision ray-tracing accelerator with 10×8 RU array is implemented in 28nm CMOS technology. The area for the design is 5.5275 mm^2. Fig. 9 shows the layout of 8 BVH traversal implemented RU column. The design runs at 200MHz under 0.9V supply. By implementing hardware data structure design for intersection acceleration for every RU column, the rendering throughput of the proposed design is improved by 6X compared with the baseline design without the optimization. Background clustering results in 56X saving on-chip memory storage for background albedo and normal map. By using mix-precision computing scheme for optimal power-quality trade-off, 16.7% of area saving and 33.4% of power saving is achieved.

Table 1. Performance comparison with previous ray-tracing hardware implementations

Solutions	Reconf. SIMT [9]	Ray Core [6]	NVIDIA GTX108Ti	Qualcomm Adreno 740	This Work
Process (nm)	90	-	16	4	28
Area (mm^2)	16	-	471	-	5.5
Architecture	SIMT	FPGA Prototype	SIMT	Mobile SIMT	ASIC
Solutions	BVH-building + Ray-Tracing	BVH-building + Ray-Tracing	Nvidia Optix	Vulkan, OpenGL	Viewpoint Reconstruction + Scalable BVH Ray-Tracing
Clock Frequency	100MHz – 400 MHz	500MHz	1480MHz	980MHz	200MHz
Bit Precision	FP32	FP8, FP16, FP24	FP32, FP64	FP16, FP32, FP64	INT8-INT64
Power	221mW	1W	250W	9.4W	69.5mW
Throughput Efficiency * (FPS/W)	27.38	17.7	0.003	35.21	789.7

*Report at peak performance

Table 1 compares this work with previous solutions. The proposed design reaches 41.6 MRay/s with the power consumption of 69.5mW at peak performance. As a result, 22X~44X higher power efficiency (MRay/s per watt) is achieved compared with the existing ASIC and the state-of-the-art mobile GPU solutions, providing a highly efficient solution to image insertion for AR/VR in highly resource-limited edge devices.

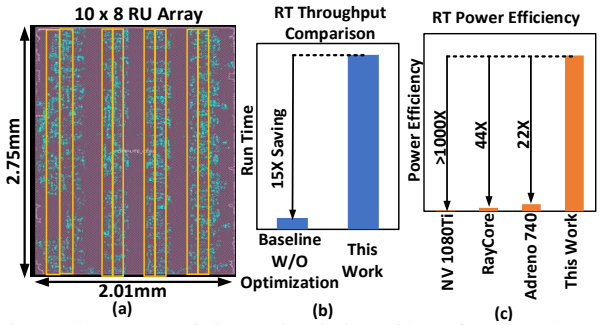

Fig. 9. (a) Layout of the render design. (b) Performance (MRay/s) improvement of baseline design. (c) RT power efficiency.

V. Conclusion

In this work, a low-power hardware acceleration solution with software-hardware codesign is proposed to deliver the challenging ray-tracing rendering operations for AR/VR on mobile edge devices. Special data structure acceleration hardware and viewpoint-focused background clustering method with inverse rendering are implemented for speed up and memory saving. In addition, a mix-precision computing scheme is adopted for optimal cost-quality trade-off. Experiments on an implementation in 28nm show that the overall rendering process is speed up by 6X on average by implementing BVH acceleration structure with only 5% overhead for on-chip memory and power cost. Finally, the implemented ray-tracing render accelerator achieves a 28X~44X higher power efficiency compared with existing ASIC and 22X compared with the state-of-the-art mobile GPU, enabling real-time ray-tracing on low-power edge devices.

VI. 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] T. Whitted, “An improved illumination model for shaded display,” *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, Jun. 1980, doi: https://doi.org/10.1145/358876.358882.

[2] S. G. Parker *et al.*, “OptiX: A General Purpose Ray Tracing Engine,” *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–13, Jul. 2010, doi: https://doi.org/10.1145/1778765.1778803.

[3] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst, “Embree: a kernel framework for efficient CPU ray tracing,” ACM Trans. Graph., vol. 33, no. 4, pp. 1–8, 2014, doi: 10.1145/2601097.2601199.

[4] J.-H. Nah, Y.-S. Kang, K.-J. Lee, S.-J. Lee, T.-D. Han, and S.-B. Yang, “MobiRT,” presented at the ACM SIGGRAPH ASIA 2010 Sketches, 2010. doi: 10.1145/1899950.1900000.

[5] C. Benthin, I. Wald, S. Woop, M. Ernst, and W. R. Mark, “Combining Single and Packet-Ray Tracing for Arbitrary Ray Distributions on the Intel MIC Architecture,” IEEE Trans. Vis. Comput. Graph., vol. 18, no. 9, pp. 1438–1448, 2012, doi: 10.1109/tvcg.2011.277.

[6] J.-H. Nah et al., “RayCore,” ACM Trans. Graph., vol. 33, no. 5, pp. 1–15, 2014, doi: 10.1145/2629634.

[7] A. L. Dos Santos, D. Lemos, J. E. F. Lindoso, and V. Teichrieb, “Real time ray tracing for augmented reality,” in 2012 14th Symposium on Virtual and Augmented Reality, 2012, pp. 131–140.

[8] Y. Zhao, S. Gong, X. Gao, W. Ai, and S.-C. Zhu, “VRKitchen2. 0-IndoorKit: A Tutorial for Augmented Indoor Scene Building in Omniverse,” ArXiv Prepr. ArXiv220611887, 2022.

[9] 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. Regul. Pap., vol. 60, no. 4, pp. 938–950, Apr. 2013, doi: 10.1109/TCSI.2012.2209302.

[10] P. H. Christensen, J. Fong, D. M. Laur, and D. Batali, “Ray tracing for the movieCars’,” in 2006 IEEE Symposium on Interactive Ray Tracing, 2006, pp. 1–6.

[11] J. Stinstra, “GTC 2020: Medical Volume Raytracing in virtual reality,” NVIDIA Developer, 10-Jun-2020. [Online]. Available: https://developer.nvidia.com/gtc/2020/video/s22030-vid. [Accessed: 20-Mar-2023].

[12] S. Kastryulin, J. Zakirov, D. Prokopenko, and D. V. Dylov, “PyTorch Image Quality: Metrics for Image Quality Assessment.” arXiv, Aug. 31, 2022. doi: 10.48550/arXiv.2208.14818.

[13] K. Ding, K. Ma, S. Wang, and E. P. Simoncelli, “Image quality assessment: Unifying structure and texture similarity,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 5, pp. 2567–2581, 2020.

[14] K. Vaidyanathan, T. Akenine-Möller, and M. Salvi, “Watertight ray traversal with reduced precision.,” in High Performance Graphics, 2016, pp. 33–40.

[15] W.-J. Lee et al., “SGRT: a mobile GPU architecture for real-time ray tracing,” in Proceedings of the 5th High-Performance Graphics Conference, in HPG ’13. New York, NY, USA: Association for Computing Machinery, Jul. 2013, pp. 109–119. doi: 10.1145/2492045.2492057.

[16] B. T. Phong, “Illumination for computer generated pictures,” Commun. ACM, vol. 18, no. 6, pp. 311–317, 1975.