RAICHU: Rendering Acceleration Integrating CGRA arcHitecture for Unified VR Pipelines

Yanzhou Tang
New York University

Haiyu Wang
New York University

Cheng Tan
Google

Sai Qian Zhang
New York University

Abstract

Virtual reality (VR) systems often execute multiple tasks concurrently, among which rendering typically dominates overall execution time. This computational dominance poses a fundamental challenge, as the system must simultaneously deliver high visual fidelity and maintain real-time frame rates, both essential for achieving responsive and immersive user interaction. Ray tracing and neural rendering such as Neural Radiance Fields (NeRF) and 3D Gaussian Splatting (3DGS) each offer distinct tradeoffs between performance and fidelity, which in turn motivates the shift towards hybrid rendering in modern VR pipelines. However, the diverse workloads within these pipelines stress different resources, and typical edge devices lack the architectural capability to support all of them efficiently.

We present RAICHU, a **R**endering **A**cceleration **I**ntegrating **C**GRA arc**H**itecture for **U**nified VR Pipelines that directly addresses this challenge via a comprehensive software-hardware co-design. At the software level, a reconfigurable dataflow toolchain maps these heterogeneous kernels onto the CGRA and optimizes their dataflow. At the hardware level, a heterogeneous CGRA is co-designed with the compiler, addressing the inherent limitations of dataflow architectures in handling memory bound tasks. To our knowledge, RAICHU is the first rendering framework to accelerate multiple rendering pipelines within a single architecture. Evaluation shows up to 109.65× higher performance than NVIDIA Jetson Orin NX and 12.33× better energy efficiency than other unified rendering accelerators.

1 Introduction

Virtual Reality (VR) is rapidly evolving from an entertainment technology into a key next-generation computing platform. As display technologies advance towards higher resolutions and refresh rates, the boundary between physical and virtual worlds becomes increasingly seamless. This evolution demands highly realistic and responsive VR experiences to ensure user immersion, which places massive compute and memory demands on the underlying system.

Among the diverse components of a VR system, rendering remains the primary bottleneck and the critical determinant of perceptual quality. Fundamentally, rendering simulates light transport to generate realistic imagery. As illustrated in Figure 1, the process begins with ray marching from the virtual camera, where rays interact with the scene's spatial representation, such as Multilayer Perceptrons (MLPs) in NeRF [46], explicit Gaussians in 3DGS [24], or Bounding Volume Hierarchies (BVH) in ray tracing. These interactions are then processed through accumulation or intersection tests to compute the final pixel color. The sense of immersion hinges on achieving photorealistic imagery while adhering to hard real-time constraints, typically ranges from 50–70 ms [1]. This dual requirement creates a fundamental tension: advanced rendering algorithms capable of photorealism often exceed the tight timing and power budgets of standalone edge VR devices. Moreover, the rendering pipeline must share limited on-device compute and memory with other latency-sensitive tasks such as Simultaneous Localization and Mapping (SLAM), eye tracking, and Asynchronous Timewarp (ATW), leading to severe resource contention.

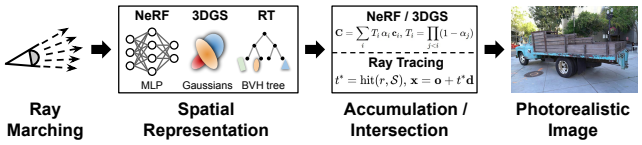

Figure 1: A rendering pipeline [69, 70] where rays interact with spatial representations such as MLPs, Gaussians, and BVHs to produce photorealistic images.

Beyond the challenge of resource contention, the rendering task itself presents a multi-faceted optimization problem. It must satisfy the concurrent demands for high visual quality, real time frame rates, and memory efficiency on resource constrained edge devices. As summarized in Table 1, different rendering methods exhibit a complex trade-off space across these three metrics. For instance, classic NeRF [47] offer high memory efficiency but suffer from poor rendering quality and slow performance. Newer variants such as Hash Grid NeRF [48] and Low Rank NeRF [5] improve quality and memory efficiency but still suffer from low rendering speed. Conversely, 3DGS [23] delivers both high quality and fast rendering, though its large memory footprint limits edge deployment. Finally, traditional ray tracing, while offering the highest visual fidelity, incurs the worst performance and memory efficiency.

The distinct tradeoffs of these rendering methods, as summarized in Table 1, naturally make them suitable for different roles within a complete VR scene. For instance, the high memory efficiency of NeRF is ideal for representing large, static backgrounds [48, 72], while the high speed and quality of 3DGS are well-suited for dynamic foreground objects or avatars [64, 84]. Selective ray tracing remains necessary for capturing physically accurate effects like transparent surfaces or hard shadows [3]. This motivates the need for a unified accelerator capable of efficiently processing all these diverse modalities, a capability that current edge architectures lack.

MICRO 2026, Athens, Greece
2026. ACM ISBN 978-X-XXXX-XXXX-X/XX/XX
https://doi.org/XXXXXXX.XXXXXXX

Table 1: Comparative overview of rendering methods. Static BG denotes static background, FG Objects denotes foreground objects, and Mem. Eff. refers to memory efficiency.

Attributes	M-NeRF	H-NeRF	L-NeRF	3DGS	RT
Speed	Medium	High	High	High	Low
Quality	Medium	High	High	High	Very High
Mem. Eff.	Very High	High	High	Medium	Medium
Typical Use Case	Legacy Low-End	Static BG	Static BG	FG Objects Avatars	Dynamic Effects
Key Refs	[35, 50, 54, 56]	[2, 22, 36, 39]	[3, 4, 43, 58]	[19, 21, 32, 61]	[15, 17, 45, 53]

Note: M-NeRF denotes MLP-based NeRF; H-NeRF denotes Hash Grid NeRF; L-NeRF denotes Low-Rank Decomposition NeRF; RT denotes ray tracing.

Building on these insights, we introduce *RAICHU* for Rendering Acceleration Integrating CGRA arcHitecture for Unified VR Pipelines. RAICHU presents a unified framework that accelerates diverse rendering algorithms via an integrated software–hardware co-design approach. To the best of our knowledge, RAICHU is the **first framework capable of efficiently accelerating *NeRF*, *3DGS*, and *Ray Tracing* within a single, cohesive edge architecture**. The key contributions of this work are summarized as follows:

- We propose a software–hardware co-design for unified VR rendering acceleration using a plug-in CGRA accelerator. Our compiler applies constant folding and operation fusion to shrink the data flow graph, improving CGRA mapping efficiency and data reuse.
- We implement a plug-in CGRA accelerator within the VR SoC, featuring a memory CGRA that decouples computation from memory access and delivers structured data streams to the PE array. This design maximizes PE utilization and fully exploits the CGRA's capability across diverse rendering workloads.
- We validate RAICHU through a user study demonstrating no perceptual quality degradation, and our evaluations across NeRF, 3DGS, and ray tracing show that RAICHU outperforms edge GPUs and prior unified rendering accelerators in both performance and energy efficiency.
- We introduce an end to end framework that enables seamless mapping from high level Python code to the heterogeneous reconfigurable fabric. Our evaluation demonstrates that RAICHU achieves up to 109.7× speedup over the Orin NX GPU, 6.1× over state of the art unified accelerators, and 13.1× over domain specific accelerators.

2 Background and Related Work

2.1 VR System

A typical standalone VR headset is built upon a heterogeneous System on Chip (SoC) architecture. This system integrates several specialized processing engines including a CPU, GPU, NPU, and a media processing unit (MPU). These components are interconnected via a shared Network on Chip (NoC) to a unified DRAM. High speed peripherals interface directly with this fabric, where image sensors stream data via MIPI CSI [26] to enable real time environment tracking and low latency interaction.

The CPU, built on a multi core architecture, manages system control, task scheduling, and peripheral coordination, directing data and workloads across heterogeneous processors. The GPU, with many streaming multiprocessors, handles geometry processing, shading, and pixel level computation, forming the backbone for real time stereoscopic rendering and scene reconstruction. The NPU focuses on machine learning inference for tasks such as tracking, SLAM, and foveated rendering, reducing latency and power by offloading AI workloads from the CPU and GPU. The MPU integrates specialized accelerators for latency critical sensing and media.

The end to end VR workflow forms a tightly coupled pipeline (*Capture* → *Perceive* → *Render* → *Display*). Sensors deliver data through MIPI, the NPU performs spatial and semantic inference, the GPU handles graphics generation, and the CPU manages scheduling and timing to maintain data consistency across modules. This coordinated use of heterogeneous processors enables low latency and energy efficient immersive rendering.

2.2 Coarse-Grained Reconfigurable Architecture

Coarse Grained Reconfigurable Arrays (CGRAs) are spatial dataflow architectures that balance the flexibility of general purpose processors with the energy efficiency of ASICs [39]. Unlike bit level FPGAs, CGRAs use word level processing elements with ALUs, multipliers, and local registers connected through a reconfigurable network. They exploit loop level parallelism by converting compute intensive loops into dataflow graphs that are mapped onto the PE array using modulo scheduling [55, 91], enabling spatial execution that avoids the von Neumann fetch and decode bottleneck and allows direct data movement between producer and consumer PEs. CGRAs offer high energy efficiency and flexibility, making them effective for regular streaming workloads and compute dense kernels. Originating in DSP [9] and extended to multimedia tasks such as video and image processing [66], CGRAs have more recently been used for energy efficient DNN inference by adapting to dense tensor operations, sparsity, and diverse operator patterns [57, 65].

2.3 Rendering Methods

Next, we summarize the principles of NeRF, 3DGS, and ray tracing, the three major rendering methods supported by RAICHU.

As shown in Figure 2a, the NeRF rendering pipeline consists of several main stages. It begins with *Ray Marching*, where rays are sent out from the camera and a sequence of three dimensional sample points is generated along each ray. The next stage is *Encoding*, which transforms the low dimensional coordinates of these points into high dimensional feature vectors so the network can capture fine detail. These encoded features then flow into a Multi Layer Perceptron (MLP), which predicts the volume density (σ) and the view dependent RGB color for every sampled point. The final stage is *Ray Composition*, where density and color predictions along each ray are accumulated through volume rendering to determine the pixel value. NeRF variants differ mainly in their encoding schemes. The original NeRF uses Positional Encoding, while newer approaches such as hash grid NeRF (e.g., Instant NGP [48]) and low rank NeRF

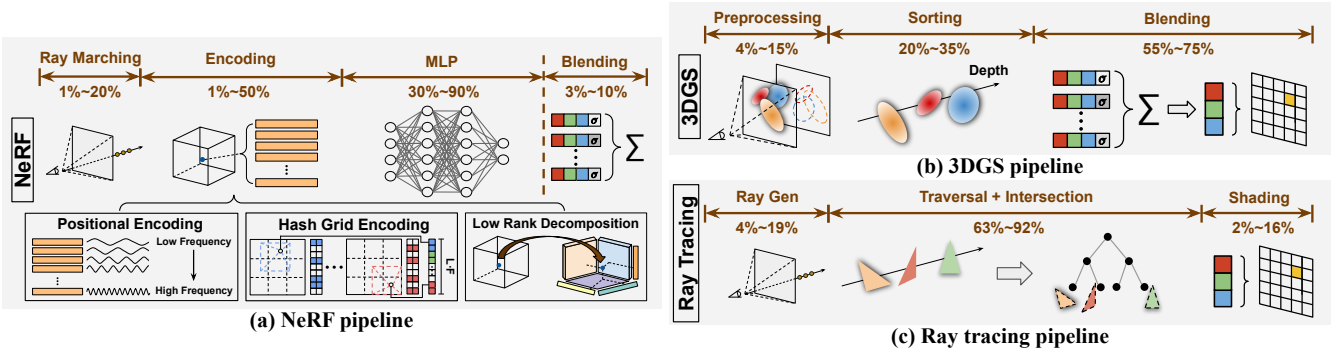

Figure 2: (a) NeRF pipeline breakdown. (b) 3DGS pipeline breakdown. (c) Ray tracing pipeline breakdown.

(e.g., TensorRF [5]) rely on Hash Encoding and Low Rank Decomposition. As indicated by the latency breakdown in Figure 3, although these newer encodings improve overall rendering speed, they also make the encoding stage a new performance bottleneck.

The pipeline for 3D Gaussian Splatting (3DGS), shown in Figure 2b, follows a rasterization based approach. It begins with a *Preprocessing* stage, where 3D Gaussian primitives are projected onto the 2D view plane. This step calculates their 2D covariance and screen space extent. The pipeline then proceeds to a *Sorting* stage that orders the projected splats by depth to ensure correct alpha blending and occlusion. The final stage is *Blending*, in which the sorted splats are drawn onto the pixel grid. Each splat contributes color and opacity to the pixels it overlaps, and these contributions are accumulated to form the final image. Among all stages, Blending has the highest latency and serves as the main computational bottleneck in the 3DGS pipeline.

The traditional ray tracing pipeline consists of three main stages. It begins with *Ray Generation*, where a primary ray is cast from each pixel in the camera view into the 3D scene. The next stage is *BVH Traversal* and *Intersection Test*, in which each ray travels through a Bounding Volume Hierarchy to locate the closest point of intersection with scene geometry. The final stage is *Shading*, where the color at the intersection point is computed, often by casting additional rays for shadows or reflections and evaluating the material and lighting conditions. Among these stages,BVH Traversal and Intersection Test is the most computationally demanding and therefore serves as the primary bottleneck in the pipeline.

2.4 Accelerators for Rendering

Hardware accelerators have been designed to support advanced rendering techniques such as NeRF, 3DGS, and ray tracing, each targeting the distinct bottlenecks of its respective method. NeRF accelerators focus on memory throughput challenges [6, 11, 27, 31, 61, 82, 88], including resolving bank conflicts introduced by hash encoding, and exploit sparsity by skipping computations for sample points that have minimal influence on the final image [18, 77]. Work on 3DGS targets its rasterization pipeline, including pruning Gaussian primitives [22, 29, 85] and optimizing the high bandwidth sorting stage required for alpha blending [29, 80] through improved memory access patterns and on chip buffering. Ray tracing acceleration focuses on reducing the cost of BVH traversal and intersection tests, using improved BVH structures [7, 21, 73], mitigating thread imbalance [71], and applying prediction mechanisms [37]. More recently, unified accelerators have begun to appear for neural rendering workloads such as NeRF and 3DGS [10, 30], leveraging shared computational structure to provide flexible and efficient hardware support across multiple rendering paradigms. However, most existing accelerators rely on fixed-function hardware pipelines designed for specific bottlenecks, and they lack a programmable framework for user-level configuration or pipeline adaptation. As a result, these architectures are usually limited to predefined execution flows and cannot easily support custom changes or evolving algorithmic structures.

3 Motivation

3.1 Rendering Performance Profiling in Multitask VR System

Modern VR systems concurrently execute multiple latency sensitive tasks, such as SLAM [4], eye tracking [40], and asynchronous timewarp [74], in addition to the primary rendering workload. As shown in Figure 3a, we profile six concurrent multitasking scenarios on a Jetson Orin NX based VR prototype, a platform widely used in prior work to emulate VR devices [30, 32, 34]. Our results show that rendering contributes the largest share of end to end latency and emerges as a primary system bottleneck, often preventing the device from meeting real time deadlines and leading to issues such as increased motion-to-photon delay [75, 76] and visual stutter [42]. Given the importance of rendering and its heavy computational demand, we believe that offloading this workload to a dedicated accelerator can reduce GPU pressure and enable other latency critical tasks to run more efficiently.

Figure 3b compares the performance and image quality of several rendering methods on an Jetson Orin. The 3DGS and NeRF variants are evaluated on the NeRF Synthetic dataset [47], and ray tracing is measured using LumiBench [38]. Ray tracing delivers the highest image quality at 43 PSNR but does so at a prohibitive cost, reaching only 0.5 frames-per-seconds (FPS). Neural rendering methods such as 3DGS and NeRF achieve much higher frame rates, with 3DGS reaching 6.4 FPS, but at the expense of peak fidelity.

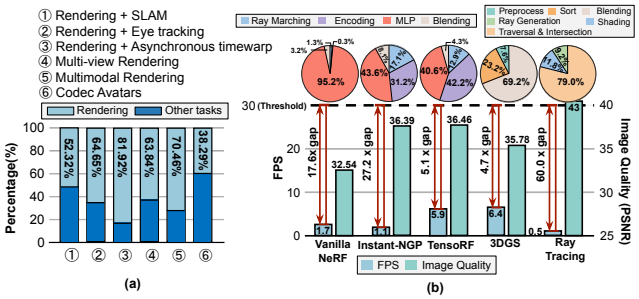

Figure 3: (a) Workload characterization of parallel applications in VR system. (b) Performance, image quality, and latency breakdown of rendering methods on an edge GPU.

3DGS also carries a large memory footprint, often several hundred megabytes, which poses challenges for deployment on resource limited edge devices. These results highlight the complementary strengths of the three methods: ray tracing provides the highest fidelity for fine details, whereas 3DGS and NeRF are better suited for backgrounds or regions with lower precision requirements, consistent with observations from prior work [12, 20]. To better support diverse applications with different visual priorities, a versatile VR system should be able to accelerate all of these rendering approaches and enable dynamic, region specific rendering policies, since each method has its own bottlenecks. For example, a foveated rendering system can employ high fidelity ray tracing at the gaze center while using NeRF or 3DGS for peripheral regions to maintain high overall frame rates. Similarly, a complex VR scene might represent vast static environments with NeRF but render interactive avatars using 3DGS to ensure low latency responsiveness.

Additionally, while Figure 3b demonstrates the potential of these rendering techniques, it also reveals a key limitation: none of them achieve real time performance on the edge GPU, which prior work suggests is 72–90 FPS range commonly targeted by interactive VR systems [8, 43, 44]. This gap further motivates the need for specialized acceleration. Yet, as illustrated in Figure 3b, building an efficient accelerator is nontrivial because each rendering method has a fundamentally different bottleneck. Vanilla NeRF is dominated by its MLP, which accounts for 95.2% of total latency. Grid-based methods such as Instant NGP and TensoRF distribute their time between encoding (31.2% and 42.2%) and the MLP (43.6% and 40.6%). The 3DGS pipeline is limited by sorting (23.2%) and blending (69.2%), whereas conventional ray tracing is overwhelmingly bottlenecked by BVH traversal and intersection (79.0%). This pronounced diversity indicates that **an accelerator tailored to only one rendering paradigm would be ineffective for the others**, making a unified accelerator architecture essential for supporting the full spectrum of rendering workloads.

3.2 Why CGRA?

As shown in Table 2, CGRAs combine the efficiency of specialized hardware with the programmability of processors, making them well-suited for the evolving rendering demands of VR edge devices [39, 56]. Standalone VR systems operate under tight power and thermal budgets, and rendering is both compute-intensive and power-hungry, requiring hardware that delivers high energy efficiency at low absolute power. Owing to their efficiency, CGRAs have been widely adopted in IoT and edge-class applications, with some designs achieving sub-milliwatt power for ultra-low-energy workloads [14, 15]. Leveraging this architecture as a rendering accelerator is therefore highly promising.

Table 2: Comparison of CGRA and Other Architectures

	CPU	GPU	FPGA	CGRA	ASIC
Energy Eff.	Low	Medium	High	High	Very High
Power	High	Very High	Medium	Low	Low
Flexibility	Very High	High	High	Medium	Low

Programmability is equally important because VR rendering is rapidly evolving, with methods such as NeRF and 3DGS introducing diverse and shifting computational patterns. CGRAs naturally support this variability through PE-level reconfigurability, enabling efficient execution across heterogeneous algorithms [36, 39]. Finally, while CGRAs provide general flexibility, they also allow domain-specific customization. By examining shared structures and bottlenecks across rendering methods, the CGRA can be tuned to optimize common kernels, achieving a balanced design that offers both programmability and specialized performance.

3.3 Common Patterns Between Rendering Methods

NeRF, 3DGS, and ray tracing rely on different scene representations, but they all follow a broadly similar pipeline structure. In the front end, each ray or sample gathers a small amount of data through highly irregular memory accesses. In the back end, intensive computation is performed for intersection, neural inference, or composition. As a result, the front end is often bottlenecked by memory access, while the back end offers substantial parallelism that is not fully utilized. The key insight, however, is that nearby rays tend to access many of the same primitives repeatedly. This access irregularity is therefore not arbitrary; instead, there is considerable latent sharing across samples.

3.3.1 Hash Grid NeRF. As illustrated in Figure 4a, in Hash Grid NeRF, samples from nearby rays often query nearly identical grid vertices across multiple levels, leading to reuse of the same vertex features as well as the same interpolation and neural computations.

3.3.2 Low Rank NeRF. As illustrated in Figure 4b, in Low Rank NeRF, samples within a ray group typically lie in a compact region, corresponding to small patches on the plane factors and short segments on the line factors. As a result, many samples reuse the same factor vectors and differ only in their interpolation weights.

3.3.3 3DGS. As shown in Figure 4c, each Gaussian typically covers a block of pixels and often extends across multiple tiles. Consequently, the same Gaussian appears in many tile lists, causing its parameters and projected conic to be loaded and evaluated repeatedly. This results in redundant memory traffic and repeated blending computations.

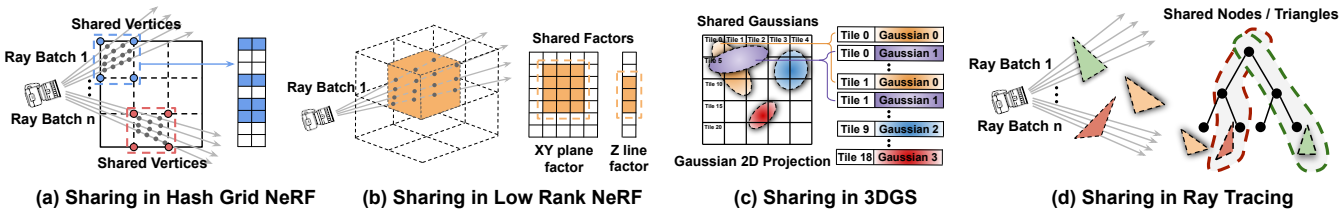

Figure 4: Common data sharing patterns across modern rendering methods. (a) In Hash Grid NeRF, rays within the same batch query highly shared sets of hash grid vertices. (b) In Low Rank NeRF, samples repeatedly reuse localized patches of plane factors and contiguous segments of line factors. (c) In 3DGS, projected Gaussians are reused by many pixels and often by multiple tiles. (d) In Ray Tracing, rays in a batch traverse largely the same BVH nodes and hit shared triangles.

3.3.4 Ray Tracing. As illustrated in Figure 4d, during ray tracing, nearby rays often traverse highly similar paths through the bounding volume hierarchy. They visit many of the same internal nodes and frequently intersect the same triangles. However, conventional traversal is still performed on a per ray basis, leading to repeated node loads and redundant intersection computations.

Building on these observations, reshaping irregular accesses into shared batches allows the compute stage to better match the strengths of a coarse grained reconfigurable array, enabling its parallel units to be utilized much more effectively. This insight suggests a decoupled architecture that separates memory orchestration from parallel computation into different pipeline stages. By isolating the collection of shared primitives from the main computation, the hardware can operate in a streaming manner. This enables a macro-pipeline in which the system prepares and reshapes the next data batch while the current batch is still being processed. Such producer-consumer coordination hides the latency of irregular memory accesses and turns the traditionally serial rendering process into a high-throughput streaming workflow.

3.4 Design Insights, Tradeoffs, and Challenges

The observations above reveal the key insight behind RAICHU: despite their different scene representations, NeRF variants, 3D Gaussian Splatting, and ray tracing share a memory-compute execution structure. Their front-end stages perform irregular but shareable accesses to spatial primitives, while their back-end stages consume these primitives through dense parallel computation. Based on this insight, RAICHU makes three key design choices. First, it decouples memory orchestration and arithmetic computation using a heterogeneous **M-CGRA/C-CGRA** organization. Second, it employs the **Data Access Engine** (DAE) and shared buffers to bridge irregular memory accesses and regular compute execution. Third, it co-optimizes the compiler and architecture to efficiently map rendering pipelines onto the heterogeneous substrate.

This design makes an explicit tradeoff. RAICHU is not a general-purpose CGRA for arbitrary workloads; instead, it specializes the reconfigurable fabric for rendering memory-compute pipelines. Compared with fixed-function accelerators, RAICHU preserves programmability across NeRF variants, 3DGS, ray tracing, and related pipelines; compared with conventional CGRAs, it sacrifices some generality for higher efficiency on this workload class

This specialization shifts the main design challenge to compilation. The compiler must partition high-level rendering kernels across two heterogeneous CGRAs, lower irregular accesses into M-CGRA primitives, fuse arithmetic patterns for the C-CGRA, and generate the shared buffer layout that connects the two lowering paths. Section 4 describes how RAICHU addresses this challenge through hardware-software co-design.

4 Methodology

The core idea behind RAICHU is a unified and reconfigurable framework that supports multiple rendering pipelines, including NeRF, 3DGS, and ray tracing. As discussed in Section 3.3, although these workloads differ in mathematical formulation, they share a common structure: dense computation combined with irregular, memory-bound dataflow. RAICHU exploits this commonality through a tightly integrated hardware-software co-design. In this section, we describe the RAICHU system architecture, compiler support, and mapping workflow.

4.1 Hardware Architecture Overview

RAICHU operates as a plug-in unified rendering accelerator integrated into a heterogeneous VR SoC, as shown in Figure 5a. The SoC CPU controls RAICHU through a scheduler interface. While CPU, GPU, and MPU subsystems collectively manage tracking, display, and general purpose computation, RAICHU offloads the rendering kernels that dominate frame latency and exhibit strong structural regularities. RAICHU does not alter the existing VR pipeline, but replaces the GPU execution of supported rendering kernels with accelerator execution.

To support this role, RAICHU adopts a dual-tier reconfigurable architecture. It consists of a **Memory CGRA** (M-CGRA) for programmable data orchestration and filtering, and a **Compute CGRA** (C-CGRA) for executing batched, regularized kernels.

The methodological core of RAICHU is a unified tile template that supports the creation of diverse reconfigurable fabrics. By freely selecting configuration parameters such as array dimension, network topology, PE specialization, memory hierarchy, and shared buffering, we can realize different types and scales of CGRA instances to meet specific application requirements. Figure 5b illustrates the full RAICHU accelerator featuring a dual tier fabric composed of these specialized instances.

As illustrated in Figure 5b, M-CGRA is the first tier of the fabric, designed to manage irregular data movement directly on the DRAM path. It is instantiated from the unified template with four memory

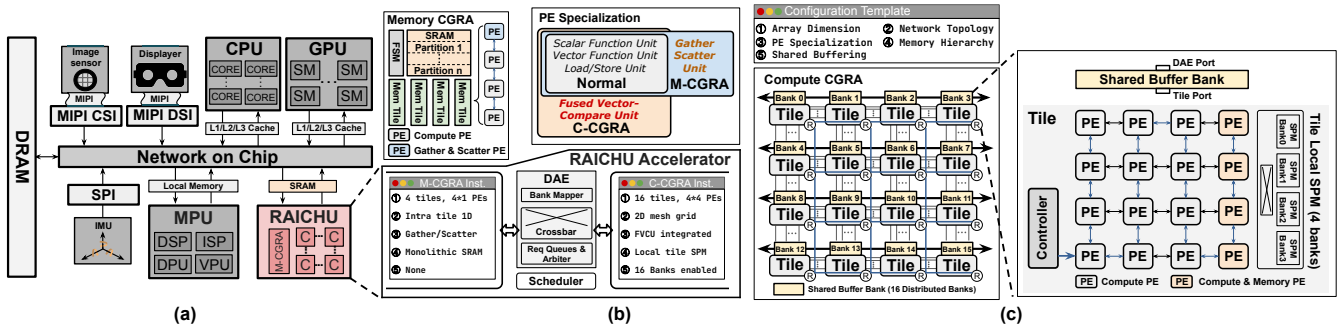

Figure 5: (a) VR SoC with RAICHU accelerator. (b) RAICHU accelerator microarchitecture. (c) Tile microarchitecture.

tiles. Each tile is configured as a 1 by 4 vertical PE array with linear interconnects between adjacent elements. Unlike a distributed memory setup, the M-CGRA features a large monolithic SRAM shared across all memory tiles. This centralized buffer sits parallel to the tiles, providing a vast contiguous space for complex data reorganization and orchestration tasks.

As illustrated in Figure 8, the M-CGRA features a four-stage programmable pipeline optimized for decoupled address generation and data gathering. While the lower three PEs use general-purpose ALUs for algorithm-specific address computations, the topmost PE integrates a Gather/Scatter (GA) unit to collect shared primitives. Through a 32×32 global crossbar, the four M-Tiles connect to a 32-bank shared SRAM. This enables each GA unit to simultaneously issue 8-wide requests to multiple banks, achieving a peak aggregate bandwidth of 32 data elements per cycle. Hardware arbiters transparently serialize any bank conflicts. This vertical topology and multi-banked SRAM reshape irregular memory fetches into dense data batches, effectively masking memory latency via its deep pipeline and highly concurrent memory subsystem.

The C-CGRA is the second tier of the fabric, instantiated from the unified template as a 4×4 grid of tiles. As shown in Figure 5c, each tile is configured with a 4 by 4 PE array, totaling 256 processing elements across the entire fabric. The network topology is established as a 2D mesh grid to support high density data movement and spatial reuse during rendering. In terms of memory hierarchy, each tile contains a multi banked tile local scratchpad memory (SPM) for near data computation.

Furthermore, the C-CGRA enables the shared buffering knob by connecting each tile to a corresponding bank of the shared buffer system. These banks feature dual ports to mediate streaming dataflow from the M-CGRA and local access from the tile itself. The internal PE specialization of the C-CGRA is tailored for intensive rendering arithmetic. As detailed in Figure 7, each PE features specialized functional units including scalar units, vector units, and a fused vector compare unit (FVCU). The FVCU is specifically designed to execute core arithmetic and geometric primitives common across modern rendering pipelines. This organization allows RAICHU to maintain tightly coupled dataflow between the M-CGRA driven preprocessing and tile level computation.

The Data Access Engine (DAE) manages the communication between the orchestration tier and the compute tier. The processed data from the M-CGRA is placed into a distributed shared buffer system consisting of 16 banks. A lightweight crossbar within the DAE ensures efficient routing based on bank locality and dependency constraints. This partitioned architecture creates a macro pipeline where the M-CGRA proactively prepares and reshapes data while the compute tier simultaneously processes the current workload. This streaming synergy effectively hides the long latency of irregular DRAM access and ensures that the hardware remains highly utilized across diverse rendering modalities.

4.2 Compiler and Programming Model

The RAICHU compiler translates high level rendering kernels into representations that can be executed efficiently by the M-CGRA and C-CGRA. As Figure 6a shows, the process begins by lowering Python kernels into RAICHU IR, a pipeline oriented form where major rendering stages such as hash grid encoding, Gaussian accumulation, MLP and intersection test appear as explicit operations. This level preserves the structure of the rendering pipeline without binding computation to hardware.

Next, the compiler classifies operators according to their algorithmic properties and user specified configuration. Stages with irregular data access or memory dominated behavior are mapped to the M-CGRA, while compute-intensive regions that benefit from parallel arithmetic are assigned to the C-CGRA. As a result, the RAICHU IR naturally serves as the point where the rendering pipeline splits into two coordinated lowering paths.

4.2.1 ***Co-design with Memory CGRA.*** The M-CGRA manages irregular data access patterns prevalent in rendering pipelines. To support this specialized unit, the compiler follows a structured workflow comprising Pattern Matching, Primitive Mapping, Dataflow graphs (DFG) Generation, DFG Manipulation, and Mapping. While this sequence mirrors the compute path, the primitive mapping stage is uniquely critical for translating high level algorithmic intent into efficient hardware memory operations. The overall position of this process within the compiler is shown in Figure 6a.

The Primitive Mapping stage serves as the bridge between Python programming abstractions and hardware capabilities. As illustrated in Figure 6b, programmers can express irregular memory operations directly through explicit gather and scatter interfaces. In a standard lowering flow, a feature extraction loop might be translated into eight discrete load operations which incurs significant

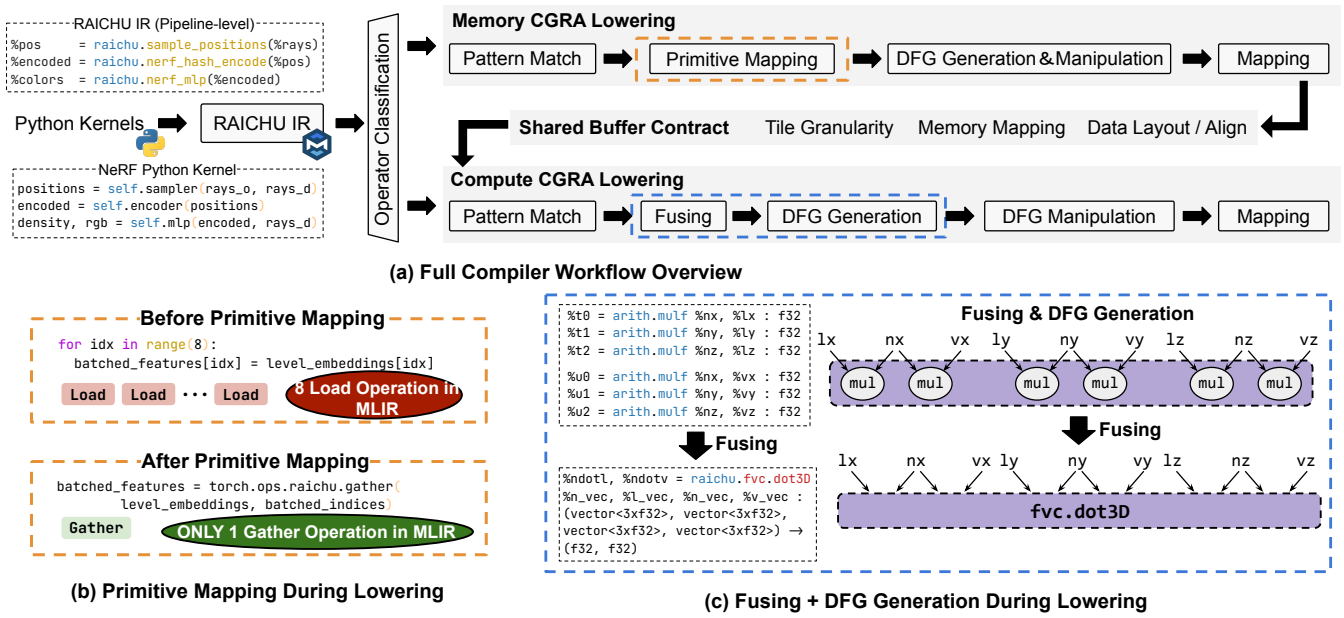

Figure 6: (a) E2E compiler workflow. (b) M-CGRA side primitive mapping. (c) C-CGRA side operator fusing and DFG generation.

instruction overhead and memory latency. Through Primitive Mapping, the compiler identifies these patterns and consolidates them into a single hardware gather operation. This transformation effectively leverages the specialized address generation logic within the M-CGRA to achieve superior throughput.

Following this, the compiler performs DFG Generation and Manipulation to optimize the data flow between memory primitives. The resulting DFG reflects the batch level structure required for efficient rendering. The compiler validates batch sizes against hardware limits and shared buffer capacity while rewriting index expressions into batch and inner coordinates. This ensures that the irregular accesses are transformed into a uniform and structured format before they reach the C-CGRA.

After DFG manipulation, the compiler materializes the shared buffer by assigning each batch to a specific bank and base address. It emits the address computation logic that places each encoded element in its correct location within the distributed shared buffer. The resulting layout is bank aware so that C-CGRA tiles can read input batches directly without further reformatting or synchronization.

4.2.2 ***Co-design with C-CGRA.*** For RAICHU IR regions that the algorithm and configuration assign to the C-CGRA, the compiler follows the C-CGRA path illustrated in Figure 6a. This path reshapes arithmetic code into compact vector kernels that align with the parallel structure of each PE. The process includes pattern matching, operation fusion, dataflow graph construction, and final mapping onto the C-CGRA array.

The key transformation is operation fusion for the FVCU. The compiler identifies arithmetic patterns that naturally map onto the six-lane fused unit. For example, dot-product and matrix-vector computations in Low Rank NeRF are rewritten into partial dot-product and reduction operations. In 3DGS and ray-based shading, paired 3D dot products such as $n \times l$ and $n \times v$ are fused into a single parallel operation, as illustrated in Figure 6c. Ray-tracing operations like axis-aligned box checks are converted into vectorized forms that expose their parallel structure. Single 3D dot products and six-element fused multiply-accumulate chains are likewise collapsed into compact FVCU primitives. Thus, scalar multiply-add chains are replaced by a few fused operations, supported by vector loads matching the FVCU layout.

After fusion, the compiler constructs a dataflow graph that captures the dependencies among vector loads, fused operations, and the remaining scalar steps. This graph is then simplified into a compact kernel block that can be instantiated across all C-CGRA tiles without modification. During mapping, this block is placed and scheduled on the C-CGRA. The shared buffer ensures that each tile receives a coherent batch of data, allowing the compiler to map the same fused graph to every tile while selecting FVCU modes and aligning vector loads with the buffer banks. The regularized batches produced by the M-CGRA eliminate imbalance across tiles and enable the C-CGRA to function as a set of parallel compute units executing an identical fused kernel.

4.3 Software-Hardware Co-design Pipeline

The proposed co-designed pipeline organizes diverse rendering workloads into a unified execution flow across the M-CGRA and C-CGRA. The M-CGRA maintains multiple batches in flight, advancing each through idle, fetching, and pending states while storing ray state, stage tags, and per-batch data blocks in its local SRAM. In doing so, it absorbs irregular memory accesses, gathers the working set required for each rendering stage, and performs lightweight preprocessing to produce batch-organized job lists.

These job lists are then forwarded to the DAE, which maps them onto shared-buffer banks and distributes them across the C-CGRA

tiles. Each tile executes a fused compute kernel on its assigned jobs and returns intermediate results, completion tags, and pixel colors to the M-CGRA. Together, the DAE and the Memory and Compute CGRAs form a continuous producer-consumer pipeline that exposes locality, balances memory and computation, and sustains high throughput across heterogeneous units.

Figure 8 illustrates the generalized hardware-software co-design pipeline of our architecture. Batches advance through *Idle, Fetching*, and *Pending* states under the M-CGRA's control. During the fetching stage, the M-CGRA gathers required spatial primitives to construct coalesced job lists, effectively masking memory latency. The DAE then dispatches these jobs to the C-CGRA. Here, compiled kernels execute on the PE array, leveraging local SPM for intermediate results. Once complete, C-CGRA tiles return updated data and a completion tag to the M-CGRA, establishing a continuous, high-throughput execution flow.

4.3.1 ***Hash Grid NeRF.*** Hash grid NeRF maps to the M-CGRA by tracking sample positions and partial MLP activations within the ray state. State transitions drive the M-CGRA to gather voxel-corner features across active hash levels, merging them into dense embeddings. The data access engine feeds these embeddings to the C-CGRA, where fused kernels update the MLP stages to maximize on-chip compute utilization.

4.3.2 ***Low Rank NeRF Pipeline.*** Low rank neural radiance fields map to the M-CGRA for coherent embedding generation and task mapping. State transitions enable extensive tile reuse to smooth memory traffic and maximize data locality. The data access engine feeds these tasks to the C-CGRA for high utilization execution.

4.3.3 ***3DGS Pipeline.*** 3D gaussian splatting maps to the M-CGRA to compute conic terms and generate gaussian centric job lists. State transitions facilitate loading each gaussian once while filtering pixels to maximize reuse across neighboring spatial tiles. The data access engine distributes tasks to the C-CGRA for fused accumulation and efficient scratchpad memory utilization.

4.3.4 ***Ray Tracing Pipeline.*** Ray tracing maps to the M-CGRA by grouping spatially neighboring rays to share bounding volume hierarchy nodes. State transitions enable coalesced memory requests and node reuse across overlapping ray paths to smooth memory traffic. The data access engine distributes bundled intersection jobs to the C-CGRA for fused kernel evaluation and high throughput execution.

4.4 Fused Vector Compute Unit

The PE incorporates a compact FVCU that consolidates the short vector patterns pervasive in NeRF, 3DGS and ray tracing. These workloads repeatedly generate 3–6 element Fused Multiply Adds (FMAs), min/max chains, and small dot product groups that are too fine grained to efficiently occupy a C-CGRA tile on their own. The FVCU provides a dedicated datapath to fuse these fragments into a single, high utilization pipeline.

As shown in Figure 7b, each FVCU lane consists of an FP32 FMA followed by a configurable network of min/max units and forwarding switches, coordinated by four gate modes. Gate mode 1 (G1) performs six independent multiply–add operations, directly

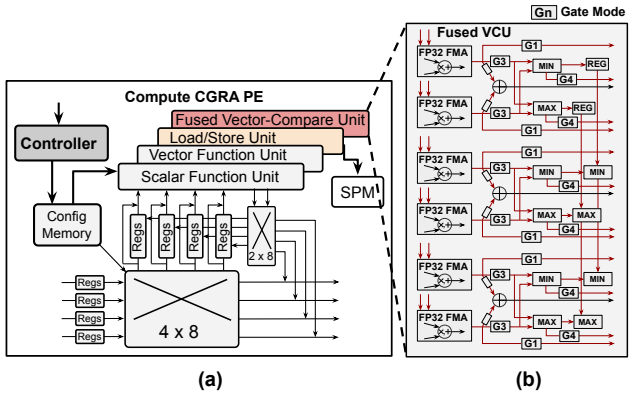

Figure 7: (a) Compute CGRA PE-level architecture overview. (b) Fused Vector-Compare Unit (FVCU) microarchitecture.

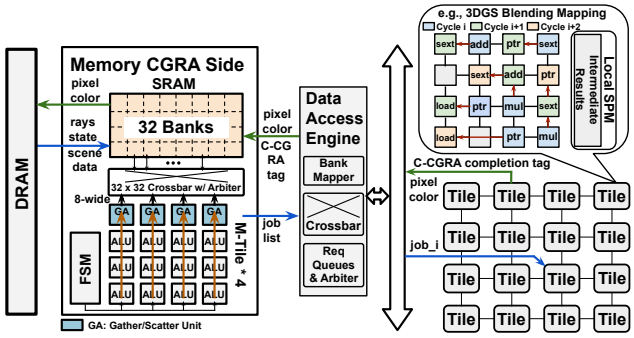

Figure 8: SW-HW co-designed pipeline on RAICHU.

supporting patterns such as partial_dot6. G2 extends this by pairing and summing the six FMA results, enabling the reduce stage commonly used in NeRF and low-rank NeRF. G3 implements the comparison-and-interval updates required for AABB slab tests in ray traversal. G4 performs pairwise compare–and–swap, which is useful for ordering the dual 3D dot products in 3DGS or tightening interval bounds in ray queries. By switching among these modes, the FVCU flexibly captures the dominant vector motifs in rendering pipelines, allowing the compiler to pack heterogeneous small operators into a single, regularized compute path.

By hardening these recurrent motifs, the FVCU keeps PE throughput high even when kernels contain many tiny vector operations. Combined with the M-CGRA's data reordering, it ensures that the C-CGRA remains compute-bound and fully utilized across diverse rendering workloads.

4.5 Foveated Rendering Implementation

RAICHU also supports foveated rendering. Specifically, we adopt a unified foveated sampling scheme, consistent with the structure that applies naturally across all rendering methods considered in this work, including NeRF-style volume accumulation, 3D Gaussian Splatting, and ray-tracing-based intersection pipelines. The image plane is divided into 3×3 tiles, and each tile selects a sampling mask

Table 3: Performance comparison of proposed RAICHU on five typical rendering pipelines for scenes from the NeRF Synthetic dataset [47] and LumiBench dataset [38]: (a) performance speedup and (b) energy efficiency improvement over edge GPU platforms, versatile rendering accelerators, and application-specialized rendering accelerators. EGP denotes Edge GPU Platform, VRA denotes Versatile Rendering Accelerator, and ASRA denotes Application-Specialized Rendering Accelerator.

Platform	Work	M-NeRF	G-NeRF	L-NeRF	3DGS	RT
EGP	Orin NX [52]	33.92×	109.65×	5.67×	16.68×	7.18×
	Nano [53]	135.59×	175.40×	7.43×	19.36×	288.02×
VRA	Uni-Render [30]	4.85×	1.32×	4.36×	1.85×	-
	FlexNeRFer [49]	1.88×	6.09×	0.31×	-	-
ASRA	-	MetaVRain [18]	Instant-3D [35]	RT-NeRF [31]	GSCore [29]	AQB8 [21]
		0.44×	7.92×	13.08×	1.01×	3.95×

(a) Performance Speedup

Platform	Work	M-NeRF	G-NeRF	L-NeRF	3DGS	RT
EGP	Orin NX [52]	87.87×	151.60×	10.83×	20.78×	17.58×
	Nano [53]	428.78×	375.70×	19.67×	38.54×	883.01×
VRA	Uni-Render [30]	7.77×	2.12×	6.99×	2.97×	-
	FlexNeRFer [49]	3.81×	12.33×	0.63×	-	-
ASRA	-	MetaVRain [18]	Instant-3D [35]	RT-NeRF [31]	GSCore [29]	AQB8 [21]
		0.16×	4.17×	27.18×	0.25×	11.43×

(b) Energy Efficiency Improvement

Table 4: Rendering quality comparison. Entries marked with an asterisk indicate our reproduced results.

Implementation	M-NeRF	G-NeRF	L-NeRF	3DGS
Baseline	31.01	30.85	31.61	33.32
RAICHU	**30.81**	**30.72**	**31.23**	**33.48**

Note: M-NeRF, G-NeRF, and L-NeRF denote MLP-based NeRF, Hash Grid NeRF, and Low-Rank Decomposition NeRF, respectively.

based only on its distance from the gaze point. Tiles in the foveal region retain all nine pixels. Tiles in the parafoveal region shade six structured pixels. Perifoveal tiles keep only three diagonal samples, while peripheral tiles are reduced to a single anchor sample.

M-CGRA drives this process by determining the region of each tile, assigning the appropriate sampling mask, and emitting shading tasks only for the marked pixels. Pixels that are not sampled are reconstructed by copying the value of the nearest sampled point inside the same tile, which keeps the reconstruction lightweight while preserving spatial coherence. Because the sampling rule depends only on tile geometry, it forms a shared front end for all rendering pipelines. Whether the downstream stage blends Gaussians, marches rays through a scene, or evaluates hash based encodings, CGRA receives a uniformly thinned set of pixel tasks. This reduces shading cost across all methods while maintaining perceptual fidelity, setting the stage for the co-designed optimizations that follow.

5 Evaluation

Our evaluation uses PyTorch and CUDA implementations for NeRF and 3DGS, and a Vulkan [62] implementation for ray tracing. We study three representative NeRF variants, KiloNeRF [60], Instant-NGP [48], and TensoRF [5], along with the official 3DGS implementation [23]. To establish baselines and characterize workloads, we profile rendering kernels on two NVIDIA edge devices, Jetson Orin NX [52] and Jetson Nano [53], both used in prior work as VR edge platforms [13, 89]. RAICHU is evaluated using a dedicated simulation framework built on a custom MLIR-based toolchain, with specialized compiler passes for mapping rendering kernels onto the CGRA and enabling detailed latency analysis. A cycle-accurate simulator is used to measure end-to-end pipeline performance.The simulator is a custom accelerator level cycle simulator driven by compiler-generated CGRA mappings. It executes mapped kernels cycle by cycle while accounting for data dependencies, resource availability, communication latency, SRAM bandwidth and bank constraints. We validate functional outputs against the original PyTorch, CUDA, Vulkan implementations and set timing parameters and resource constraints according to the RTL microarchitecture. The simulator models off-chip memory with the assumption of a 102.4 GB/s DRAM bandwidth, which is the same as the typical bandwidth of the LPDDR5 memory subsystem used in the baseline Jetson Orin NX device [52].

We also implement the DAE and CGRA in RTL, with synthesizable CGRA RTL generated using VecPAC [68]. We verify the RTL functionality using module-level and integration-level testbenches that cover DAE request scheduling, arbitration, shared-SRAM access, CGRA PE/FVCU execution, routing, memory operations, and end-to-end data movement. The design is synthesized in Synopsys Design Compiler targeting a 45 nm library to report area, power, and maximum frequency. On-chip SRAM area and energy are modeled with CACTI-7.0 [33], and DeepScaleTool [63] is used to normalize power across process nodes for fair comparison.

5.1 Algorithm Evaluation

In this section, we assess the effect of our proposed algorithmic optimizations on rendering quality. Specifically, we integrate the pipeline reordering techniques described in Section 3.3 into all the rendering pipelines. We also apply FP8 quantization to the exponential function in the 3DGS blending stage and to the MLPs in the NeRF pipeline. The effect of the reordered pipeline presented in Section 3.3 on image quality is quantitatively evaluated using the NeRF Synthetic Dataset, with Peak Signal-to-Noise Ratio (PSNR) as the metric. As summarized in Table 4, the results show that across all evaluated scenes and methods, the PSNR degradation remains consistently below 1 dB relative to baselines. Since a difference of less than 1 dB is typically imperceptible in visual quality and falls well below established human perceptual thresholds [81], this confirms that RAICHU algorithm preserves rendered image fidelity with negligible impact.

5.2 Hardware Evaluation

5.2.1 ***Power & Area Breakdown.*** We synthesize the RAICHU accelerator using Synopsys Design Compiler with a FreePDK 45nm library [67] at a target frequency of 1 GHz to obtain detailed power

	Power	Area
PE Array	3.22 W	7.42 mm2
SRAM	0.93 W	2.95 mm2
CGRA Ctrl	0.17 W	0.28 mm2
M-CGRA	0.70 W	1.82 mm2
Total	5.02 W	12.5 mm2
Process	45 nm	

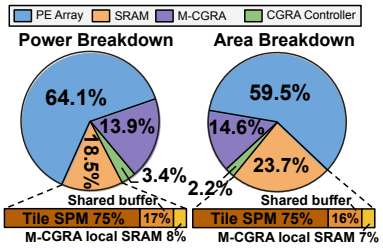

Figure 9: Power and area breakdown of RAICHU accelerator

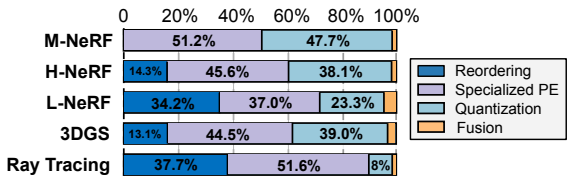

Figure 10: Contribution of each component to the speedup of different rendering methods. M-NeRF, H-NeRF, and L-NeRF denote MLP-based, Hash Grid, and Low-Rank NeRF.

and area estimates. For energy, we model the off-chip DRAM access energy as 8 pJ/bit [54] multiplied by the per-method DRAM traffic from the simulator. On chip SRAM area and energy are characterized separately using CACTI 7.0. As shown in Figure 9, the accelerator consumes 5.02 W in total. The C-CGRA array is the primary contributor, accounting for 64.1% of overall power, followed by SRAM at 18.5% and the M-CGRA plus DAE at 13.9%. The total chip area is 12.5 mm^2, with the C-CGRA array occupying 59.5%, SRAM 23.7%, and the M-CGRA plus DAE 14.6%.

We compare RAICHU with commercial edge devices [52, 53], versatile rendering accelerators [30, 49], and application-specific accelerators [18, 21, 29, 31, 35], evaluating latency and energy efficiency across different rendering methods, as summarized in Table 3. NeRF and 3DGS workloads are profiled using the NeRF Synthetic dataset [47], while ray tracing is evaluated with LumiBench [38].

5.2.2 ***Comparison over Edge GPU.*** We use Jetson Orin NX and Jetson Nano as representative edge platforms in our evaluation. For energy comparison, RAICHU power refers to the synthesized accelerator subsystem, while Jetson energy is measured as idle-subtracted module-level dynamic energy [79, 90] using the onboard power monitors, since a GPU-only power rail is not exposed on these platforms [50]. To match the module level Jetson measurements, which already include off-chip DRAM power, RAICHU energy in this section also includes off-chip DRAM access energy, as described in Section 5.2.1. Using NeRF Synthetic dataset [47] for neural radiance fields and 3D gaussian splatting alongside LumiBench [38] for ray tracing, RAICHU achieves 5.67× to 288.02× speedup and 10.83× to 883.01× energy efficiency gains over both edge GPU baselines across all five methods. For ray tracing workloads, the Orin includes dedicated RT Cores, whereas the Nano lacks specialized ray tracing hardware and must execute highly divergent kernels on CUDA Cores. As a result, RAICHU achieves a much larger speedup over the Nano than over the Orin on ray tracing tasks.

5.2.3 ***Comparison with Other Versatile Rendering Accelerators.*** We select two existing versatile rendering accelerators including Uni Render [30] and FlexNeRFer [49] to compare with the proposed RAICHU. As shown in Table 3, compared with UniRender, RAICHU achieves 1.32×–4.85× speedup and 2.12×–7.77× higher energy efficiency, while providing additional acceleration support for the ray tracing pipeline. Compared with FlexNeRFer, RAICHU delivers 0.31×–6.09× speedup and 0.63×–12.33× higher energy efficiency, while additionally supporting both 3DGS and ray tracing. Overall, RAICHU provides higher performance and energy efficiency than existing versatile designs while supporting a broader set of rendering pipelines. This advantage comes from its heterogeneous CGRA architecture and specialized compiler, jointly exploiting specific computation patterns through SW-HW co-design.

5.2.4 ***Comparison with Application Specific Rendering Accelerators.*** As shown in Table 3, RAICHU achieves 0.44×–13.08× speedup and 0.16×–27.18× higher energy efficiency. RAICHU shows lower performance than MetaVRain on MLP-based NeRF, partly because MetaVRain is specifically optimized for this pipeline. In addition, RAICHU is mainly designed to accelerate pipelines that contain a memory-bound stage followed by a compute-bound stage. However, as discussed in Section 3.1, the latency of MLP-based NeRF is dominated almost entirely by MLP computation, which limits RAICHU's ability to fully exploit its architectural advantages. For the other specialized accelerators, RAICHU achieves better results on nearly all pipelines, demonstrating strong competitiveness across Hash Grid NeRF, Low Rank NeRF, 3DGS, and ray tracing.

5.2.5 ***Speedup Mechanisms.*** RAICHU's speedups come from matching each workload bottleneck to the M-CGRA/C-CGRA execution path in Section 4.3. MLP-based NeRF is compute-dominated, so its gain mainly comes from C-CGRA/FVCU fused arithmetic; this also explains why MetaVRain [18], which is fully specialized for MLP-based NeRF, can outperform RAICHU on this single pipeline. For Hash Grid NeRF and Low-Rank NeRF, the mechanisms follow the mappings in Sections 4.3.1 and 4.3.2: the M-CGRA batches shared hash vertices or plane/line factors across nearby samples before the C-CGRA performs interpolation and MLP execution, thereby reducing redundant feature traffic. For 3DGS, the mapping in Section 4.3.3 leverages its natural Gaussian-centric structure: the M-CGRA loads each Gaussian once per job batch, filters its affected pixels or tiles, and emits compact jobs to C-CGRA, reducing repeated Gaussian parameter movement through the pipeline. However, these gains rely on spatial locality; lower locality reduces reusable data sharing and therefore limits the performance gain.

For ray tracing, the Section 4.3.4 mapping exploits rays locality: spatially neighboring rays are usually adjacent in the ray order and thus likely fall into the same batch. The M-CGRA coalesces shared BVH node and triangle fetches along overlapping paths without an extra grouping stage, while the C-CGRA evaluates bundled intersection jobs from shared buffers. However, at deeper BVH levels, ray paths diverge and node sharing drops, making performance more M-CGRA bandwidth-bound.

Overall, these mechanisms reflect a deliberate tradeoff in RAICHU. The M-CGRA/C-CGRA split and FVCU specialization, as discussed in Section 3.4, reduce generality and make RAICHU less suitable for arbitrary workloads, while improving efficiency for rendering pipelines with shareable irregular accesses and dense arithmetic.

5.3 Ablation Study

5.3.1 ***M-CGRA SRAM Capacity vs. C-CGRA Array Size.*** We sweep M-CGRA buffering and bandwidth together with C-CGRA array size, as shown in Figure 11a. M-CGRA resources are scaled to 0.5×, 1×, and 2× the baseline, while the C-CGRA array ranges from 8×8 (64 PEs) to 32×32 (1024 PEs). Performance is normalized to the configuration with 1× SRAM and a 16×16 C-CGRA. Sensitivity to hardware scaling varies substantially across workloads. MLP-based NeRF is purely compute-bound and scales linearly with C-CGRA size, with little dependence on M-CGRA capacity. Hash Grid NeRF and Low Rank NeRF are also mainly C-CGRA bound, but show modest gains from larger M-CGRA resources at bigger array sizes due to improved feature sharing. In contrast, 3DGS requires balanced scaling of both units: while small configurations are compute-limited, scaling the C-CGRA to 32×32 yields significant benefits only when M-CGRA resources are also doubled to relieve memory pressure. Ray tracing, by contrast, is strictly memory-bound. Its performance scales almost linearly with M-CGRA capacity, while larger C-CGRA arrays provide little benefit. These results show that area allocation between M-CGRA and C-CGRA should be guided by the dominant rendering characteristics of the target workload.

5.3.2 ***PE Array Scaling within C-CGRA Tiles.*** We further study the effect of scaling the PE array size within each C-CGRA tile while fixing the total number of tiles at 4×4. Using the 4×4 PE array as the baseline, we evaluate 2×2, 3×3, 5×5, and 4×8 configurations across the five rendering pipelines, as shown in Figure 11c.

For MLP-based NeRF, performance scales almost proportionally with PE array size, confirming that this workload is strongly compute-bound and dominated by MLP execution. In contrast, the other four pipelines do not show proportional gains as the PE array grows, suggesting that the baseline already lies near an effective design point for these workloads. Increasing C-CGRA compute resources alone, without matching improvements in memory access, disrupts the balance between data orchestration and computation. Ray tracing benefits the least from this scaling. As a representative memory-bound workload, it sees little improvement from additional compute units once memory bandwidth becomes the main bottleneck. These results highlight the need to balance M-CGRA and C-CGRA resources to avoid underutilizing larger compute arrays.

5.3.3 ***FVCU versus a Four Lane Vector Unit.*** We evaluate the benefits of the specialized FVCU by comparing it with a standard four-lane SIMD vector unit while keeping the M-CGRA and C-CGRA configurations at their baseline values. As shown in Figure 11b, the specialized FVCU is important for compute-intensive rendering tasks. MLP-based NeRF achieves a 2.2× speedup because its matrix-vector layers map directly to the FVCU's specialized partial dot modes. Hash Grid NeRF and Low Rank NeRF similarly gain 1.6×–1.7× from more efficient dot-product support in their encoding and MLP layers. 3DGS improves by 1.8×, as Gaussian blending kernels benefit from specialized dot2x3 and fused multiply-add modes. In contrast, ray tracing is less sensitive to FVCU scaling because it is mainly limited by M-CGRA traffic. Replacing the FVCU with a four-lane unit reduces ray tracing performance by only 1.3×, mainly due to slower AABB tests and comparison operations. Overall, these results justify our design choice: NeRF and 3DGS benefit most from specialized C-CGRA compute, while ray tracing depends more on M-CGRA resources. This supports the inclusion of a modestly larger but more efficient FVCU in each PE.

Table 5: Throughput of RAICHU under Foveated Setting.

Pipeline	MLP	Low-rank NeRF	Hash Grid NeRF	3DGS	Ray Tracing
Ref.	[47]	[5]	[48]	[23]	[38]
FPS	359	134	428	313	195

Table 6: Exposed frame-level reconfiguration latency.

Workload	Ideal	No Pref.	Prefetch	Fusion
RT + NeRF	2.664 ms	169.34 μs	3.22 μs	0.03 μs
		6.355%	0.121%	0.00113%
RT + 3DGS	3.430 ms	4.24 μs	1.31 μs	0.03 μs
		0.124%	0.038%	0.00087%
RT + 3DGS + NeRF	3.038 ms	171.92 μs	3.23 μs	0.04 μs
		5.659%	0.106%	0.00132%

5.3.4 ***Speedup Breakdown.*** We evaluate the individual contributions of Pipeline Reordering, Specialized PE, Quantization, and Operation Fusion. As shown in Figure 10, we measure each contribution using logarithmic speedup values. The heterogeneous M-CGRA/C-CGRA architecture together with Pipeline Reordering accounts for 13.1%–37.7% of the total speedup, confirming that the data orchestration optimizations in Section 3.3 effectively reduce memory bottlenecks. Although MLP-based NeRF does not use reordering because it is compute-dominated, the other optimizations still provide substantial gains. Specialized PE modules contribute the most across most workloads. By accelerating common rendering operations with dedicated PE support and compiler-level fusion, RAICHU achieves high execution efficiency. Quantization, from FP8 to FP16, further contributes 8%–47.7% through MLP computation, blending, and intersection tests. The contribution of general Operation Fusion is smaller because much of the rendering-specific fusion is already captured in the Specialized PE category. Overall, these results show that the combination of reconfigurable memory orchestration and specialized compute support is the main source of RAICHU's performance gains.

5.4 User Study

To evaluate our RAICHU-based foveated 3DGS, we conduct a user study to determine whether foveated rendering introduces perceivable visual artifacts. The goal is to assess visual quality under gaze-contingent viewing and verify that RAICHU maintains high-quality VR experiences. We recruit seven participants, each seated and viewing stimuli through a Meta Quest Pro HMD [41], using the controllers to switch between images across trials. The stimuli are

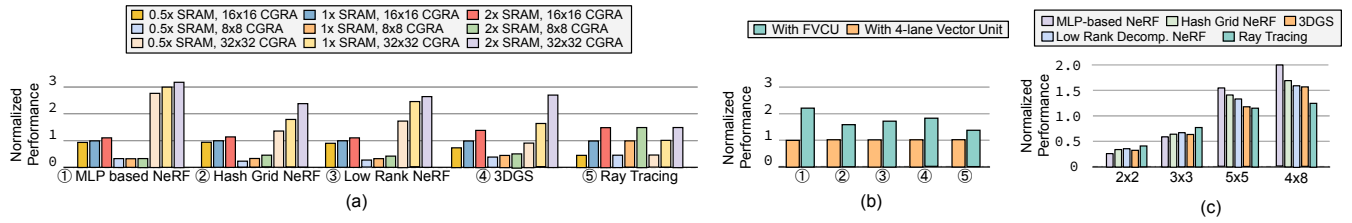

Figure 11: (a) Normalized performance of five rendering workloads: ① MLP-based NeRF, ② Hash-Grid NeRF, ③ Low-Rank NeRF, ④ 3DGS, and ⑤ Ray Tracing, when sweeping M-CGRA side SRAM/bandwidth (0.5×, 1×, 2×) and CGRA array size (8×8, 16×16, 32×32). Performance is normalized to the baseline configuration with 1× M-CGRA SRAM and a 16×16 CGRA. (b) Performance with the proposed FVCU versus a design where the FVCU is replaced by a conventional 4-lane vector unit. Bars are normalized to the 4-lane vector unit design (larger is better). (c) Speedup of different rendering methods while scaling C-CGRA.

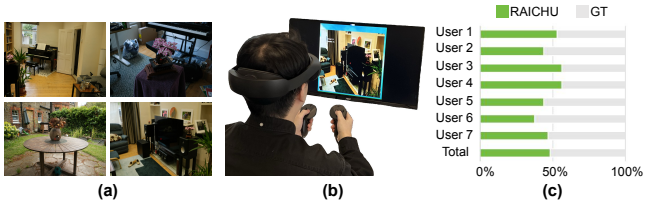

Figure 12: (a) Four test images rendered by RAICHU 3DGS. (b) Participants taking part in the study. (c) Selection results.

generated by rendering multiple scenes with 3DGS. We first select four representative viewpoints from four scenes (Figure 12a). For each view, we use the full-resolution 3DGS image as the ground-truth (GT) reference and render a corresponding foveated image with RAICHU using a 3×3 upsampling layout centered on the screen, emulating a central gaze position. Each viewpoint therefore yields a GT/RAICHU image pair.

Participants then complete a two-interval forced-choice (2IFC) task [86]. In each trial, two images of the same scene and viewpoint were shown (t1 and t2), representing the ground-truth and RAICHU outputs in randomized order. Participants freely toggle between the two using the controllers and were instructed to keep their gaze near a central fixation point while selecting the image with higher perceived quality. Each participant complete 32 trials, covering four image pairs with two presentation orders and four repetitions, providing a compact yet diverse set of comparisons across scenes and randomized conditions. The aggregated results are shown in Figure 12c. Across all trials, the RAICHU foveated images were chosen in 48.2% ± 7.2% of cases, essentially matching chance. This near-equal preference indicates that participants could not reliably distinguish RAICHU from GT, suggesting that the foveated output preserves visual quality and is suitable for practical VR deployment.

5.5 Throughput under Foveated Setting

With the combined hardware-software optimizations described above, together with our foveated rendering strategy, RAICHU's throughput is summarized in Table 5. Most evaluated pipelines exceed the 120 FPS target for high-end immersive experiences. Specifically, MLP-based NeRF reaches 359 FPS, hash-grid NeRF achieves 428 FPS, 3DGS reaches 313 FPS, and ray tracing delivers 195 FPS, while low-rank NeRF reaches 134 FPS. Overall, these results show that RAICHU delivers real-time performance across a diverse range of modern rendering methods.

5.6 Reconfiguration Analysis

5.6.1 ***Reconfiguration overhead.*** RAICHU supports different rendering stages through compile time generated context programs. We distinguish intra-kernel context sequencing from inter-kernel configuration loading. During kernel execution, each PE selects a context already resident in its local context bank; this cost is included in the reported kernel latency. Inter-kernel configuration loading is modeled separately and can be overlapped with execution through double-buffered context banks and prefetching.

Table 6 reports the exposed frame level reconfiguration latency for hybrid rendering workloads. Without prefetching, stage level switching can add up to 6.355% latency. With double-buffered prefetching, the exposed overhead is reduced to 0.038%–0.121%. When stages in the same rendering method are fused into one context program per CGRA, the exposed overhead further drops to 0.00087%–0.00132%, corresponding to only 30–40 cycles per frame.

5.6.2 ***Configurability and support for new variants.*** RAICHU uses the same hardware template for the evaluated NeRF, 3DGS, and ray tracing workloads. Its configurability comes from compile-time dataflow generation and CGRA reconfiguration, where the compiler configures the M-CGRA, C-CGRA, and DAE.

RAICHU-supported micro operations cover common rendering computations such as transformation, interpolation, filtering, comparison, accumulation, and blending. Therefore, many new rendering variants can be decomposed into these micro operations and supported by recompilation. As summarized in Table 7, compiler extensions are needed when a variant requires new dataflow/access pattern, while hardware extensions are needed only for new specialized primitives that are inefficient on existing RAICHU units.

6 Conclusion

We introduced RAICHU, a unified CGRA-based accelerator for NeRF, 3DGS, and ray tracing. Through hardware–software co-design, RAICHU transforms irregular rendering workloads into efficient compute batches, achieving large gains in performance

Table 7: Support paths for rendering algorithm variants.

Variant Type	Existing Compiler	Existing Hardware
Evaluated kernels	✓	✓
Supported micro operators	✓	✓
New dataflow/access pattern	✗	✓
New specialized primitive	✗	✗

and energy efficiency over edge GPUs while preserving visual quality. These results show that a programmable spatial architecture can effectively support diverse real-time VR rendering pipelines.

References

[1] Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. 2017. Latency requirements for foveated rendering in virtual reality. *ACM Transactions on Applied Perception (TAP)* 14, 4 (2017), 1–13.

[2] Thilini Kaushalya Bandara, Dan Wu, Rohan Juneja, Dhananjaya Wijerathne, Tulika Mitra, and Li-Shiuan Peh. 2023. FLEX: Introducing FLEXible Execution on CGRA with Spatio-Temporal Vector Dataflow. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. https://doi.org/10.1109/ICCAD57390.2023.10323612

[3] BASIL M BIJU, Sulfath P M, and Sheena K M. 2025. NVIDIA Ray Tracing: Revolutionizing Real-Time Computer Graphics. *TechRxiv* 2025, 0410 (2025). https://doi.org/10.36227/techrxiv.174431738.81337106/v1 arXiv:https://www.techrxiv.org/doi/pdf/10.36227/techrxiv.174431738.81337106/v1

[4] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M. M. Montiel, and Juan D. Tardos. 2021. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics* 37, 6 (Dec. 2021), 1874–1890. https://doi.org/10.1109/tro.2021.3075644

[5] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII* (Tel Aviv, Israel). Springer-Verlag, Berlin, Heidelberg, 333–350. https://doi.org/10.1007/978-3-031-19824-3_20

[6] Yuzhou Chen, Zhenyu Li, Dongxu Lyu, Yansong Xu, and Guanghui He. 2025. Neural Rendering Acceleration With Deferred Neural Decoding and Voxel-Centric Data Flow. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 44, 7 (2025), 2725–2737. https://doi.org/10.1109/TCAD.2024.3524918

[7] Yuan Hsi Chou and Tor M. Aamodt. 2025. Treelet Accelerated Ray Tracing on GPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 1334–1347. https://doi.org/10.1145/3676641.3716279

[8] Gyorgy Denes, Kuba Maruszczyk, Matthew Ashby, Gordon Wetzstein, Karol Myszkowski, and Piotr Didyk. 2019. Temporal Resolution Multiplexing: Exploiting the Limitations of Spatio-Temporal Vision for More Efficient VR Rendering. *IEEE Transactions on Visualization and Computer Graphics* 25, 5 (2019), 2116–2128. https://doi.org/10.1109/TVCG.2019.2898741

[9] Carl Ebeling, Darren C. Cronquist, and Paul Franklin. 1996. RaPiD - Reconfigurable Pipelined Datapath. In *Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers (FPL '96)*. Springer-Verlag, Berlin, Heidelberg, 126–135.

[10] Yu Feng, Weikai Lin, Zihan Liu, Jingwen Leng, Minyi Guo, Han Zhao, Xiaofeng Hou, Jieru Zhao, and Yuhao Zhu. 2024. Potamoi: Accelerating Neural Rendering via a Unified Streaming Architecture. *ACM Trans. Archit. Code Optim.* 21, 4, Article 80 (Nov. 2024), 25 pages. https://doi.org/10.1145/3689340

[11] Yu Feng, Zihan Liu, Jingwen Leng, Minyi Guo, and Yuhao Zhu. 2024. Cicero: Addressing Algorithmic and Architectural Bottlenecks in Neural Rendering by Radiance Warping and Memory Optimizations. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 1293–1308. https://doi.org/10.1109/ISCA59077.2024.00096

[12] Linus Franke, Laura Fink, and Marc Stamminger. 2025. VR-Splatting: Foveated Radiance Field Rendering via 3D Gaussian Splatting and Neural Points. *Proc. ACM Comput. Graph. Interact. Tech.* 8, 1, Article 18 (May 2025), 21 pages. https://doi.org/10.1145/3728302

[13] Antonin Gilles, Pierre Le Gargasson, Grégory Hocquet, and Patrick Gioia. 2023. Holographic near-eye display with real-time embedded rendering. In *SIGGRAPH Asia 2023 Conference Papers*. 1–10.

[14] Graham Gobieski, Ahmet Oguz Atli, Kenneth Mai, Brandon Lucia, and Nathan Beckmann. 2021. Snafu: An Ultra-Low-Power, Energy-Minimal CGRA-Generation Framework and Architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1027–1040. https://doi.org/10.1109/ISCA52012.2021.00084

[15] Graham Gobieski, Souradip Ghosh, Marijn Heule, Todd Mowry, Tony Nowatzki, Nathan Beckmann, and Brandon Lucia. 2022. RipTide: A Programmable, Energy-Minimal Dataflow Compiler and Architecture. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 546–564. https://doi.org/10.1109/MICRO56248.2022.00046

[16] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 3, 15 pages. https://doi.org/10.1145/3579371.3589038

[17] Dongho Ha, Lufei Liu, Yuan Hsi Chou, Seokjin Go, Won Woo Ro, Hung-Wei Tseng, and Tor M. Aamodt. 2024. Generalizing Ray Tracing Accelerators for Tree Traversals on GPUs. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1041–1057. https://doi.org/10.1109/MICRO61859.2024.00080

[18] Donghyeon Han, Junha Ryu, Sangyeob Kim, Sangjin Kim, Jongjun Park, and Hoi-Jun Yoo. 2024. MetaVRain: A Mobile Neural 3-D Rendering Processor With Bundle-Frame-Familiarity-Based NeRF Acceleration and Hybrid DNN Computing. *IEEE Journal of Solid-State Circuits* 59, 1 (2024), 65–78. https://doi.org/10.1109/JSSC.2023.3291871

[19] David Heaney. 2023. Quest 3 Teardown Reveals It's Mostly Battery Inside. https://www.uploadvr.com/quest-3-teardown-battery/. Accessed: 2026-06-16.

[20] Horácio Henriques, Eder Oliveira, Esteban Clua, and Daniela Trevisan. 2024. A mixed path tracing and NeRF approach for optimizing rendering in XR Displays. In *Proceedings of the 25th Symposium on Virtual and Augmented Reality* (Rio Grande, Brazil) *(SVR '23)*. Association for Computing Machinery, New York, NY, USA, 123–130. https://doi.org/10.1145/3625008.3625027

[21] Yen-Chieh Huang, Chen-Pin Yang, and Tsung Tai Yeh. 2025. AQB8: Energy-Efficient Ray Tracing Accelerator through Multi-Level Quantization. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 374–387. https://doi.org/10.1145/3695053.3731104

[22] Joongho Jo and Jongsun Park. 2025. PS-GS: Group-Wise Parallel Rendering with Stage-Wise Complexity Reductions for Real-Time 3D Gaussian Splatting. In *2025 Design, Automation Test in Europe Conference (DATE)*. 1–7. https://doi.org/10.23919/DATE64628.2025.10992921

[23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4, Article 139 (July 2023), 14 pages. https://doi.org/10.1145/3592433

[24] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.* 42, 4 (2023), 139–1.

[25] Wonung Kim, Yubin Lee, Yoonsung Kim, Jinwoo Hwang, Seongryong Oh, Jiyong Jung, Aziz Huseynov, Woong Gyu Park, Chang Hyun Park, Divya Mahajan, and Jongse Park. 2025. Pimba: A Processing-in-Memory Acceleration for Post-Transformer Large Language Model Serving. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 292–307. https://doi.org/10.1145/3725843.3756121

[26] Philippe Lancheres and Mohamed Hafed. 2019. The MIPI C-PHY standard: A generalized multiconductor signaling scheme. *IEEE Solid-State Circuits Magazine* 11, 2 (2019), 69–77.

[27] Junseo Lee, Kwanseok Choi, Jungi Lee, Seokwon Lee, Joonho Whangbo, and Jaewoong Sim. 2023. NeuRex: A Case for Neural Rendering Acceleration. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 21, 13 pages. https://doi.org/10.1145/3579371.3589056

[28] Junseo Lee, Jaisung Kim, Junyong Park, and Jaewoong Sim. 2025. VR-Pipe: Streamlining Hardware Graphics Pipeline for Volume Rendering. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 217–230. https://doi.org/10.1109/HPCA61900.2025.00027

[29] Junseo Lee, Seokwon Lee, Jungi Lee, Junyong Park, and Jaewoong Sim. 2024. GSCore: Efficient Radiance Field Rendering via Architectural Support for 3D Gaussian Splatting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 497–511. https://doi.org/10.1145/3620666.3651385

[30] Chaojian Li, Sixu Li, Linrui Jiang, Jingqun Zhang, and Yingyan Celine Lin. 2025. Uni-Render: A Unified Accelerator for Real-Time Rendering Across Diverse Neural Renderers. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 246–260. https://doi.org/10.1109/HPCA61900.2025.00029

[31] Chaojian Li, Sixu Li, Yang Zhao, Wenbo Zhu, and Yingyan Lin. 2022. RT-NeRF: Real-Time On-Device Neural Radiance Fields Towards Immersive AR/VR Rendering. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design* (San Diego, California) *(ICCAD '22)*. Association for Computing Machinery, New York, NY, USA, Article 132, 9 pages. https://doi.org/10.1145/3508352.3549380

[32] Leshu Li, Jiayin Qin, Jie Peng, Zishen Wan, Huaizhi Qu, Ye Han, Pingqing Zheng, Hongsen Zhang, Yu Cao, Tianlong Chen, and Yang (Katie) Zhao. 2025. RTGS: Real-Time 3D Gaussian Splatting SLAM via Multi-Level Redundancy Reduction. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 1838–1851. https://doi.org/10.1145/3725843.3756099

[33] Sheng Li, Ke Chen, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. 2011. CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 694–701. https://doi.org/10.1109/ICCAD.2011.6105405

[34] Sixu Li, Ben Keller, Yingyan Celine Lin, and Brucek Khailany. 2025. GauRast: Enhancing GPU Triangle Rasterizers to Accelerate 3D Gaussian Splatting. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. https://doi.org/10.1109/DAC63849.2025.11132449

[35] Sixu Li, Chaojian Li, Wenbo Zhu, Boyang (Tony) Yu, Yang (Katie) Zhao, Cheng Wan, Haoran You, Huihong Shi, and Yingyan (Celine) Lin. 2023. Instant-3D: Instant Neural Radiance Field Training Towards On-Device AR/VR 3D Reconstruction. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 6, 13 pages. https://doi.org/10.1145/3579371.3589115

[36] Zhaoying Li, Dhananjaya Wijerathne, and Tulika Mitra. 2024. Coarse-Grained Reconfigurable Array (CGRA). In *Embedded Computing*. Springer. https://doi.org/10.1007/978-981-97-9314-3_50

[37] Lufei Liu, Wesley Chang, Francois Demoullin, Yuan Hsi Chou, Mohammadreza Saed, David Pankratz, Tyler Nowicki, and Tor M. Aamodt. 2021. Intersection Prediction for Accelerated GPU Ray Tracing. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 709–723. https://doi.org/10.1145/3466752.3480097

[38] Lufei Liu, Mohammadreza Saed, Yuan Hsi Chou, Davit Grigoryan, Tyler Nowicki, and Tor M. Aamodt. 2023. LumiBench: A Benchmark Suite for Hardware Ray Tracing. In *2023 IEEE International Symposium on Workload Characterization (IISWC)*. 1–14. https://doi.org/10.1109/IISWC59245.2023.00011

[39] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. 2019. A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications. *ACM Comput. Surv.* 52, 6, Article 118 (Oct. 2019), 39 pages. https://doi.org/10.1145/3357375

[40] Wenxuan Liu, Budmonde Duinkharjav, Qi Sun, and Sai Qian Zhang. 2025. FovealNet: Advancing AI-Driven Gaze Tracking Solutions for Efficient Foveated Rendering in Virtual Reality. *IEEE Transactions on Visualization and Computer Graphics* 31, 5 (May 2025), 3183–3193. https://doi.org/10.1109/TVCG.2025.3549577

[41] Meta. 2022. Meta Quest Pro. https://www.meta.com/quest/quest-pro/. Accessed: 2025-07-21.

[42] Meta. 2026. Missed Frames and Frame Recovery. https://developers.meta.com/horizon/documentation/native/android/os-missed-frames/. [Online; accessed 2026-04-01].

[43] Meta. 2026. Rendering. https://developers.meta.com/horizon/documentation/unreal/unreal-advanced-rendering/. [Online; accessed 2026-04-01].

[44] Meta. 2026. Set Display Refresh Rates. https://developers.meta.com/horizon/documentation/native/android/mobile-display-refresh-rate/. [Online; accessed 2026-04-01].

[45] Meta. n.d.. Meta Quest 3: Next-Gen Virtual Reality Headset. https://www.meta.com/quest/quest-3/. Accessed: 2026-06-16.

[46] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

[47] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (Dec. 2021), 99–106. https://doi.org/10.1145/3503250

[48] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

[49] Seock-Hwan Noh, Banseok Shin, Jeik Choi, Seungpyo Lee, Jaeha Kung, and Yeseong Kim. 2025. FlexNeRFer: A Multi-Dataflow, Adaptive Sparsity-Aware Accelerator for On-Device NeRF Rendering. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 1894–1909. https://doi.org/10.1145/3695053.3731107

[50] NVIDIA. 2024. Jetson Orin Nano Series, Jetson Orin NX Series and Jetson AGX Orin Series. *NVIDIA Jetson Linux Developer Guide*. https://docs.nvidia.com/jetson/archives/r36.4.4/DeveloperGuide/SD/PlatformPowerAndPerformance/JetsonOrinNanoSeriesJetsonOrinNxSeriesAndJetsonAgxOrinSeries.html Jetson Linux r36.4.4. Accessed: 2026-06-12.

[51] NVIDIA. n.d.. NVIDIA Jetson Thor: Advanced AI for Physical Robotics. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-thor/. Accessed: 2026-06-16.

[52] NVIDIA Corporation. 2022. NVIDIA Jetson Orin NX Series Data Sheet. Online. https://developer.nvidia.com/downloads/jetson-orin-nx-series-data-sheet DS-10712-001. Accessed: 2026-06-18.

[53] NVIDIA Corporation. n.d.. Jetson Nano Brings the Power of Modern AI to Edge Devices. Online. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/ Accessed: 2026-06-18.

[54] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W. Keckler, and William J. Dally. 2017. Fine-grained DRAM: energy-efficient DRAM for extreme bandwidth systems. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, Massachusetts) *(MICRO-50 '17)*. Association for Computing Machinery, New York, NY, USA, 41–54. https://doi.org/10.1145/3123939.3124545

[55] Hyunchul Park, Kevin Fan, Scott Mahlke, Taewook Oh, Heeseok Kim, and Hongseok Kim. 2008. Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. In *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 166–176.

[56] Artur Podobas, Kentaro Sano, and Satoshi Matsuoka. 2020. A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective. *IEEE Access* 8 (2020), 146719–146743. https://doi.org/10.1109/ACCESS.2020.3012084

[57] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2017. Plasticine: A reconfigurable architecture for parallel patterns. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 389–402. https://doi.org/10.1145/3079856.3080256

[58] Jiajun Qin, Tianhua Xia, Cheng Tan, Jeff Zhang, and Sai Qian Zhang. 2025. PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 845–861. https://doi.org/10.1145/3676641.3716013

[59] Akshat Ramachandran, Souvik Kundu, and Tushar Krishna. 2025. MicroScopiQ: Accelerating Foundational Models through Outlier-Aware Microscaling Quantization. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 1193–1209. https://doi.org/10.1145/3695053.3730989

[60] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. 2021. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 14315–14325. https://doi.org/10.1109/ICCV48922.2021.01407

[61] Junha Ryu, Hankyul Kwon, Wonhoon Park, Zhiyong Li, Beomseok Kwon, Donghyeon Han, Dongseok Im, Sangyeob Kim, Hyungnam Joo, Minsung Kim, and Hoi-Jun Yoo. 2025. NeuGPU: An Energy-Efficient Neural Graphics Processing Unit for Instant Modeling and Real-Time Rendering on Mobile Devices. *IEEE Journal of Solid-State Circuits* 60, 1 (2025), 99–111. https://doi.org/10.1109/JSSC.2024.3447701

[62] Mohammadreza Saed, Yuan Hsi Chou, Lufei Liu, Tyler Nowicki, and Tor M. Aamodt. 2022. Vulkan-Sim: A GPU Architecture Simulator for Ray Tracing. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 263–281. https://doi.org/10.1109/MICRO56248.2022.00027

[63] Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401196

[64] Zhijing Shao, Zhaolong Wang, Zhuang Li, Duotun Wang, Xiangru Lin, Yu Zhang, Mingming Fan, and Zeyu Wang. 2024. SplattingAvatar: Realistic Real-Time Human Avatars With Mesh-Embedded Gaussian Splatting. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1606–1616. https://doi.org/10.1109/CVPR52733.2024.00159

[65] Dongjoo Shin, Jinmook Lee, Jinsu Lee, Juhyoung Lee, and Hoi-Jun Yoo. 2018. DNPU: An Energy-Efficient Deep-Learning Processor with Heterogeneous Multi-Core Architecture. *IEEE Micro* 38, 5 (Sept. 2018), 85–93. https://doi.org/10.1109/MM.2018.053631145

[66] H. Singh, Ming-Hau Lee, Guangming Lu, F.J. Kurdahi, N. Bagherzadeh, and E.M.C. Filho. 1998. MorphoSys: a reconfigurable architecture for multimedia applications. In *Proceedings. XI Brazilian Symposium on Integrated Circuit Design (Cat. No.98EX216)*. 134–139. https://doi.org/10.1109/SBCCI.1998.715427

[67] James E. Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W. Rhett Davis, Paul D. Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, and Ravi Jenkal. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. 173–174.

https://doi.org/10.1109/MSE.2007.44

[68] Cheng Tan, Deepak Patil, Antonino Tumeo, Gabriel Weisz, Steve Reinhardt, and Jeff Zhang. 2023. VecPAC: A Vectorizable and Precision-Aware CGRA. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. https://doi.org/10.1109/ICCAD57390.2023.10323910

[69] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B Goldman, and Michael Zollhöfer. 2020. State of the Art on Neural Rendering. arXiv:2004.03805 [cs.CV] https://arxiv.org/abs/2004.03805

[70] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, Tomas Simon, Christian Theobalt, Matthias Niessner, Jonathan T. Barron, Gordon Wetzstein, Michael Zollhoefer, and Vladislav Golyanik. 2022. Advances in Neural Rendering. arXiv:2111.05849 [cs.GR] https://arxiv.org/abs/2111.05849

[71] Yavuz Selim Tozlu and Huiyang Zhou. 2025. CoopRT: Accelerating BVH Traversal for Ray Tracing via Cooperative Threads. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 166–179. https://doi.org/10.1145/3695053.3731118

[72] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. 2022. Mega-NeRF: Scalable Construction of Large-Scale NeRFs for Virtual Fly- Throughs. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12912–12921. https://doi.org/10.1109/CVPR52688.2022.01258

[73] K. Vaidyanathan, S. Woop, and C. Benthin. 2022. Wide BVH traversal with a short stack. In *Proceedings of the Conference on High-Performance Graphics* (Strasbourg, France) *(HPG '19)*. Eurographics Association, Goslar, DEU, 15–19. https://doi.org/10.2312/hpg.20191190

[74] J. M. P. van Waveren. 2016. The asynchronous time warp for virtual reality on consumer hardware. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology* (Munich, Germany) *(VRST '16)*. Association for Computing Machinery, New York, NY, USA, 37–46. https://doi.org/10.1145/2993369.2993375

[75] VR/AR Wiki. 2026. Motion-to-photon latency. https://vrarwiki.com/wiki/Motion-to-photon_latency. [Online; last edited 2026-03-16; accessed 2026-04-01].

[76] D. Wagner. 2018. Motion To Photon Latency In Mobile AR And VR. https://medium.com/@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926. [Online; accessed 2026-04-01].

[77] Haochuan Wan, Linjie Ma, Antong Li, Pingqiang Zhou, Jingyi Yu, and Xin Lou. 2024. ZeroTetris: A Spacial Feature Similarity-based Sparse MLP Engine for Neural Volume Rendering. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '24)*. Association for Computing Machinery, New York, NY, USA, Article 224, 6 pages. https://doi.org/10.1145/3649329.3655684

[78] Haiyu Wang, Wenxuan Liu, Kenneth Chen, Qi Sun, and Sai Qian Zhang. 2025. Process Only Where You Look: Hardware and Algorithm Co-optimization for Efficient Gaze-Tracked Foveated Rendering in Virtual Reality. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 344–358. https://doi.org/10.1145/3695053.3731110

[79] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 97–110. https://doi.org/10.1109/HPCA51647.2021.00018

[80] Yuanfang Wang, Yu Li, Jianli Chen, Jun Yu, and Kun Wang. 2025. FAMERS: An FPGA Accelerator for Memory-Efficient Edge-Rendered 3D Gaussian Splatting. In *2025 Design, Automation Test in Europe Conference (DATE)*. 1–7. https://doi.org/10.23919/DATE64628.2025.10993091

[81] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

[82] Lizhou Wu, Haozhe Zhu, Jiapei Zheng, Mengjie Li, Yinuo Cheng, Qi Liu, Xiaoyang Zeng, and Chixiao Chen. 2024. Hi-NeRF: A Multicore NeRF Accelerator With Hierarchical Empty Space Skipping for Edge 3-D Rendering. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 12 (2024), 2315–2326. https://doi.org/10.1109/TVLSI.2024.3458032

[83] Tianhua Xia, Haiyu Wang, and Sai Qian Zhang. 2026. Segment Only Where You Look: Leveraging Human Gaze Behavior for Efficient Computer Vision Applications in Augmented Reality. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (USA) *(ASPLOS '26)*. Association for Computing Machinery, New York, NY, USA, 1693–1710. https://doi.org/10.1145/3779212.3790216

[84] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. 2024. PhysGaussian: Physics-Integrated 3D Gaussians for Generative Dynamics . In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 4389–4398. https://doi.org/10.1109/CVPR52733.2024.00420

[85] Zhifan Ye, Yonggan Fu, Jingqun Zhang, Leshu Li, Yongan Zhang, Sixu Li, Cheng Wan, Chenxi Wan, Chaojian Li, Sreemanth Prathipati, and Yingyan Celine Lin. 2025. Gaussian Blending Unit: An Edge GPU Plug-in for Real-Time Gaussian-Based Rendering in AR/VR. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 353–365. https://doi.org/10.1109/HPCA61900.2025.00036

[86] Yaffa Yeshurun, Marisa Carrasco, and Laurence T Maloney. 2008. Bias and sensitivity in two-interval forced choice procedures: Tests of the difference model. *Vision research* 48, 17 (2008), 1837–1851.

[87] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2024. Mip-Splatting: Alias-Free 3D Gaussian Splatting . In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 19447–19456. https://doi.org/10.1109/CVPR52733.2024.01839

[88] Yipu Zhang, Jiawei Liang, Jian Peng, Jiang Xu, and Wei Zhang. 2025. SpNeRF: Memory Efficient Sparse Volumetric Neural Rendering Accelerator for Edge Devices. arXiv:2505.08191 [cs.AR] https://arxiv.org/abs/2505.08191

[89] Ziliang Zhang, Zexin Li, Hyoseung Kim, and Cong Liu. 2024. BOXR: Body and head motion Optimization framework for eXtended Reality. *arXiv preprint arXiv:2410.13084* (2024).

[90] Zhekai Zhang, Hanrui Wang, Song Han, and William J. Dally. 2020. SpArch: Efficient Architecture for Sparse Matrix Multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 261–274. https://doi.org/10.1109/HPCA47549.2020.00030

[91] Zhongyuan Zhao, Weiguang Sheng, Qin Wang, Wenzhi Yin, Pengfei Ye, Jinchao Li, and Zhigang Mao. 2020. Towards Higher Performance and Robust Compilation for CGRA Modulo Scheduling. *IEEE Transactions on Parallel and Distributed Systems* 31, 9 (2020), 2201–2219. https://doi.org/10.1109/TPDS.2020.2989149