METASAPIENS: Real-Time Neural Rendering with Efficiency-Aware Pruning and Accelerated Foveated Rendering

Weikai Lin*
University of Rochester
Rochester, NY, USA
wlin33@ur.rochester.edu

Yu Feng*
Shanghai Jiao Tong University
Shanghai, China
y-feng@sjtu.edu.cn

Yuhao Zhu
University of Rochester
Rochester, NY, USA
yzhu@rochester.edu

Abstract

Point-Based Neural Rendering (PBNR) is emerging as a promising class of rendering techniques, which are permeating all aspects of society, driven by a growing demand for real-time, photorealistic rendering in AR/VR and digital twins. Achieving real-time PBNR on mobile devices is challenging.

This paper proposes METASAPIENS, a PBNR system that for the first time delivers real-time neural rendering on mobile devices while maintaining human visual quality. METASAPIENS combines three techniques. First, we present an efficiency-aware pruning technique to optimize rendering speed. Second, we introduce a Foveated Rendering (FR) method for PBNR, leveraging humans' low visual acuity in peripheral regions to relax rendering quality and improve rendering speed. Finally, we propose an accelerator design for FR, addressing the load imbalance issue in (FR-based) PBNR. Our evaluation shows that our system achieves an order of magnitude speedup over existing PBNR models without sacrificing subjective visual quality, as confirmed by a user study. The code and demo are available at: https://horizon-lab.org/metasapiens/.

CCS Concepts: • **Computer systems organization → Architectures**; • **Human-centered computing → Ubiquitous and mobile computing**.

Keywords: Gaussian Splatting, Foveated Rendering, Hardware Accelerator

---

*Both authors contributed equally to this research.

1 Introduction

Rendering technologies are infiltrating every facet of society. For instance, rendering is critical to enabling digital twins [31] in emerging areas such as smart cities, digital healthcare, and telepresence. Reinvigorated interests in Augmented/Virtual Reality (AR/VR) further heighten the demand for real-time, photorealistic rendering.

Point-Based Neural Rendering (PBNR), i.e., the Gaussian Splatting-family algorithms [17, 18, 22, 34, 40, 50], is emerging as a new class of rendering solutions, which revitalizes the classic point-based rendering techniques [23, 42, 53, 79] using modern neural rendering methods [46]. PBNR, like previous neural rendering algorithms such as Neural Radiance Fields (NeRF), offers photorealistic rendering by learning scene radiance from data, but is significantly faster by replacing the compute-intensive Multilayer Perceptrons (MLPs) in NeRF with lightweight point-based rasterization.

Nevertheless, PBNR is still far from real-time on mobile devices, rendering generally below 10 Frames-Per-Second (FPS) on the mobile Volta GPU [2]. This paper introduces METASAPIENS, which, for the first time, delivers real-time PBNR on mobile devices while maintaining human visual quality. METASAPIENS combines three key ingredients.

Efficiency-Aware Pruning. Much of the recent efforts on optimizing PBNR models focus on pruning [17, 18, 22, 40, 50], which, while reducing the model size, does not bring significant speedups. This is because existing pruning methods are single-minded in reducing the sheer number of points while being agnostic to the actual computational cost. We find that different points in a PBNR model contribute differently to the overall computation. Instead, we propose an efficiency-aware pruning method that directly optimizes for the rendering/inference speed (Sec. 3).

Foveated PBNR. METASAPIENS also exploits characteristics of human vision to improve performance (Sec. 4). Human vision acuity is poor in the visual periphery [73], an opportunity that has long been exploited: one can speed up rendering by gradually reducing the rendering quality as the pixel eccentricity increases (i.e., as pixels are positioned more at

the visual periphery) with impunity, a technique known as Foveated Rendering (FR) [24, 51].

We introduce the first FR method for PBNR. We gradually reduce the number of points used for rendering as the pixel eccentricity increases. The key is a data representation, where points at higher eccentricies are purposely designed to be a strict subset of the points at lower eccentricies. That way, (most of the) parameters and computation are shared when rendering different eccentricity regions, improving performance and reducing storage requirements.

Equally important to improving performance is to maintain a high visual quality. To that end, we introduce a new training method, which guides both pruning and peripheral quality relaxation (in FR) by explicitly modeling human visual perception at different eccentricities. As a result, the subjective visual quality is consistent across the visual field and is aligned with the dense, non-FR model.

Architectural Support. While the two techniques above readily provide about an order of magnitude speedup on GPU, we co-design an accelerator architecture to further improve the performance (Sec. 5). Aside from providing hardware support for FR, the hardware addresses a key performance bottleneck in PBNR exacerbated by FR: low hardware utilization due to workload imbalance across tiles in a frame. Our hardware mitigates this issue via 1) a dynamic tile merging scheme that balances workloads across tiles and 2) incrementally pipelining adjacent stages through line-buffering.

Result. We evaluate our method using both subjective human studies and objective measurements of performance and quality. Across 12 participants, the subjective rendering quality of our method is statistically no-worse than that of Mini-Splatting-D [18], a state-of-the-art PBNR method in rendering quality. Compared to five state-of-the-art PBNR methods, we out-perform all of them in both objective rendering quality (by up to 0.4 dB in PSNR) and rendering speed (by up to 7.4× on a mobile Volta GPU and 20.9× with our hardware support).

The contributions of this paper are as follows:

- We propose an efficiency-aware pruning method for PBNR that directly targets computational efficiency rather than merely reducing point counts.
- We propose the first FR method tailored for PBNR; the method is centered around a new data/point representation, which improves rendering performance with little storage overhead.
- We introduce a training framework that incorporates both pruning and FR while maintaining subjective visual quality; the key is to explicitly model HVS and use the model to guide training.
- We co-design an accelerator architecture, which addresses the load imbalance issue in PBNR and further improves performance.
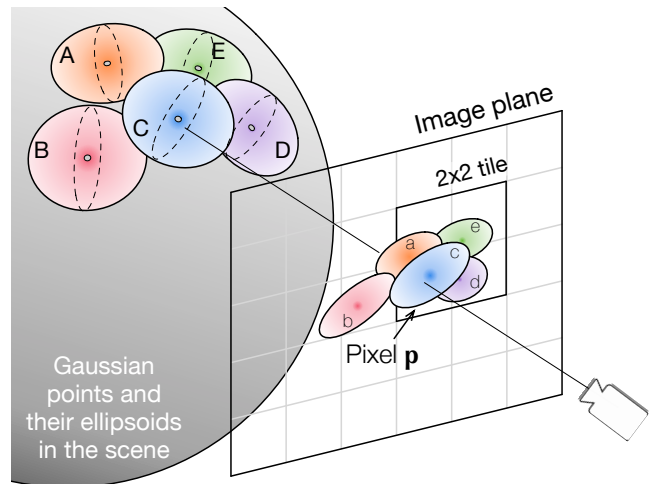

Fig. 1. Illustration of PBNR, which parameterizes the scene with a set of points, each associated with a 3D Gaussian distribution that gives rise to an ellipsoid. The ellipsoids are projected to ellipses on the image plane, where the ellipses are sorted (per tile, e.g., 2 × 2 pixels). The color of a pixel is calculated by integrating the contribution of each intersecting ellipse (e.g., a, c, d, e for **p**).

2 Background

We first introduce the necessary background in PBNR (Sec. 2.1), followed by the main characteristics of the Human Visual System (HVS) and how they are used by Foveated Rendering to improve rendering speed (Sec. 2.2).

2.1 Point-Based Neural Rendering

PBNR is a class of neural rendering techniques, exemplified by the 3D Gaussian Splatting (3DGS) algorithm [34] and its descendants [17, 18, 22, 40, 50]. Compared to previous neural rendering techniques, a.k.a., the NeRF-family algorithms [8, 11, 46, 49, 69], PBNR is fundamentally more efficient (e.g., usually over 1,000 times faster), because it parameterizes the scene with discrete points (rather than voxels) to avoid redundant computations and renders via a lightweight rasterization-based process called splatting [57, 62, 79] rather than the heavy MLP inference.

General PBNR Pipeline. We use 3DGS as a running example to explain the general pipeline of PBNR, which all PBNR algorithms follow. Fig. 1 illustrates the general idea. Rendering starts with an offline-trained model, which contains a set of discrete points (A–E) that represent the scene. Each point is associated with an ellipsoid, whose three-dimensional scales are determined by the σs of a 3D Gaussian distributions (hence the name Gaussian point). Each ellipsoid has a set of *trainable* parameters, including the scales, position, orientation, opacity, color distribution (which is parameterized through Spherical Harmonics; SH).

Given the trained points/ellipsoids, the online rendering follows three steps: *Projection*, *Sorting*, and *Rasterization*.

Projection. Each ellipsoid is first projected/splatted to an ellipse on the image plane[1]. In the example of Fig. 1, the ellipsoids A–E in the scene are splatted to ellipses a–e on the image plane. The goal is to identify, for each pixel tile (e.g., 2×2), which ellipses intersect with the tile and thus contribute to the pixel colors in the tile.

Sorting. For each tile, we sort all the intersecting ellipses based on their depths to the image plane; that way, closer ellipses can be integrated first when calculating pixel colors. For instance in Fig. 1, ellipse c would be the closest.

Rasterization. Finally, we calculate the intersections of all the ellipses in a tile with each pixel. The color of a pixel **p** is then computed using the classic volume rendering method [33, 41, 45], which integrates the contribution of all the intersecting ellipses from near to far:

$$\mathbf{p} = \sum_{i=0}^{N-1} T_i \alpha_i c_i, \ T_i = \prod_{j=0}^{i-1} (1 - \alpha_j) \quad (1a)$$

$$\alpha_i = f(\text{opacity}_i, ..., \text{pose}) \quad (1b)$$

$$c_i = g(\text{SH}_0, ..., \text{SH}_n) \quad (1c)$$

where N is the number of ellipses intersecting **p** (i.e., a, c, d, e in Fig. 1), α_i is a function f of various trainable parameters of the ith intersecting ellipse (e.g., opacity) and the camera pose, and c_i is the color of the ith ellipse at the position of intersection, which is calculated using the Spherical Harmonics (SH) function g with trainable coefficients SH_n. We refer readers to Kerbl et al. [34] for the details of f and g.

2.2 Human Visual System and Foveated Rendering

Foveated Rendering. It is well-known that human visual acuity drops as eccentricity increases, i.e., when objects are placed more toward the visual periphery [73]. This is due to a combination of larger pooling sizes [15, 58] and a sparser photoreceptor distribution [14, 67] on the retina as the eccentricity increases. Foveated Rendering (FR) [24, 51] leverages this natural fall-off in visual acuity to speed up rendering by relaxing the rendering quality in peripheral regions. Fig. 2 illustrates an example where the visual content in high-eccentricity regions could be altered without being noticeable by users.

While in classic FR the peripheral rendering quality is relaxed by lowering the resolution, neural rendering offers another dimension: reducing the computational *workload* of each pixel. This new dimension is possible because the rendering load of each pixel is controlled by inferencing a deep learning model, which offers many knobs for accuracy-vs-speed trade-offs that have been extensively studied [9, 25, 30, 75]. For instance, one can train a smaller model for

---

[1]We use "points", "ellipses", and "ellipsoids" interchangeably: there is a one-to-one mapping between them.

rendering the visual periphery [16]. This paper will explore FR knobs unique to PBNR.

Modeling HVS. A key question in FR is to determine how much quality to relax without introducing visual artifacts. It is well-established that commonly used visual quality metrics such as Peak Signal to Noise Ratio (PSNR) or Structural Similarity Index Measure (SSIM) [29] do not account for the eccentricity-dependent visual acuity drop in HVS [61, 68, 72] and, thus, are inadequate for FR: an image with a low PSNR at the visual periphery might not introduce visual artifacts. The altered image in Fig. 2, when placed in the visual periphery, is visually indiscriminable from the reference image.

This paper leverages an eccentricity-aware HVS Quality (HVSQ) metric [72] inspired by classic neuroscience studies about the human visual pathway [20]. Given a reference image, an altered image, and the eccentricity of each pixel (which depends on the display resolution and the eye-display distance), HVSQ quantifies how similar the two images are as viewed by humans; a lower HVSQ means more similar.

The principle behind the HVSQ metric is as follows. The retina aggregates photoreceptor outputs in spatial regions, called spatial poolings. In the image space, a spatial pooling corresponds to a set of adjacent pixels (e.g., SP in Fig. 2). The pooling size increases with eccentricity, usually quadratically. Computational models on HVS [72] show that as long as the statistics (mean and standard deviation) of the content in a spatial pooling between two images are close, humans can not discriminate between them. The statistics are calculated in a feature space (as opposed to the pixel space) to emulate the feature extraction in human's early visual processing.

Computationally, the HVSQ of an altered image with respect to a reference image is calculated as follows:

$$HVSQ = \frac{1}{N} \sum_{i=1}^{N} \left[\left(\mathcal{M}(\mathbf{I}_i^a) - \mathcal{M}(\mathbf{I}_i^r) \right)^2 + \left(\sigma(\mathbf{I}_i^a) - \sigma(\mathbf{I}_i^r) \right)^2 \right] \quad (2)$$

where N is the number of pixels in an image (each pixel has a unique spatial pooling), $\mathbf{I}_i^r$ and $\mathbf{I}_i^a$ denote the features of the ith spatial pooling in the reference and the altered image, respectively; $\mathcal{M}$ denotes arithmetic mean, and σ denotes standard deviation.

Intuitively, the HVSQ metric calculates the average distance between the two images' statistics across all the spatial poolings. HVSQ makes intuitive sense: as pixel eccentricities increase, the pooling sizes increase, which gives us more "wiggle room" within a spatial pooling to manipulate pixel values to match the feature statistics of the reference image.

3 Efficiency-Aware Pruning

This section introduces a pruning framework to speed up PBNR. We first identify the root-cause why existing pruning methods are ineffective (Sec. 3.1). We then propose two techniques to address the root-cause: intersection-aware pruning
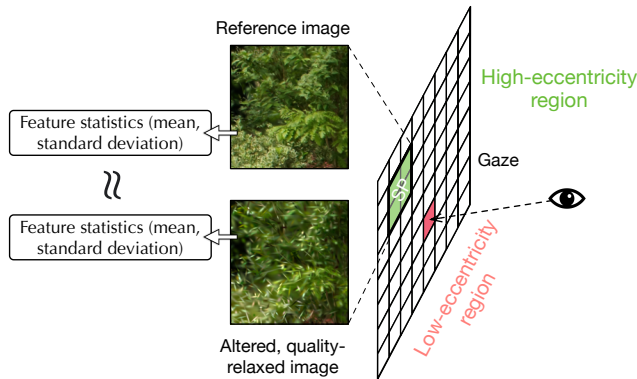
Fig. 2. Pixels under the user's gaze have low eccentricities, where the human visual quality is the highest; the peripheral pixels have high eccentricities where human visual acuity is low. In peripheral regions, the visual stimulus (image) can be altered without being discriminable from the reference stimulus if the statistics of the image features are close, as quantified by the HVSQ metric (Eqn. 2). SP: spatial pooling.

(Sec. 3.2) and scale decay (Sec. 3.3). Finally, we discuss how these two techniques are combined together (Sec. 3.4).

3.1 Motivations

Speed. The performance of recent PBNR models is far from real-time on mobile GPUs. Fig. 3 shows the FPS on the Mip-NeRF 360 [8], Tanks&Temples [35], and Deep Belending [26] dataset, measured on the mobile Volta GPU on Jetson Xavier across five recent PBNR models [17, 18, 34, 40]. The data is plotted as a standard boxplot to show the FPS distribution across the 13 traces within the datasets.

3DGS [34] and Mini-Splatting-D [18] are two dense models and generally are the slowest. Much of recent work focuses on pruning: reducing the number of points in a PBNR model [17, 18, 40]. While effective for reducing the model size, these methods do not significantly speed up rendering. For instance, CompactGS, LightGS, and Mini-Splatting in Fig. 3 are all pruned models; while generally faster than dense models, they are still below real-time, especially for immersive applications such as AR/VR, which would normally require an FPS of 75–90 [3, 4, 6].

Why is Existing Pruning Insufficient? Existing pruning methods focus on reducing the point count in a model, which is ineffective for improving speed in PBNR. To quantify this, Fig. 4 shows the inference latency (*x*-axis) vs. point count (left *y*-axis) of LightGS [17] (which prunes 3DGS [34]) trained on the `bicycle` trace in the Mip-NeRF 360 dataset at different pruning levels (between 75% and 97%). The latency reduction rate is slower than that of the point reduction rate.

The reason that reducing the point count is ineffective for acceleration is because the computational costs associated with different points vary. Fig. 5 shows the intuition,

where there are two ellipses projected onto the image plane. The smaller ellipse intersects with only two tiles whereas the larger one intersects with eight. As a result, the larger one is used in calculating more pixel colors and is naturally responsible for more computation.

Therefore, what *does* impact the inference speed is the number of tile-ellipse intersections. Fig. 4 shows the latency vs. the average number of intersections per tile (right *y*-axis) for each pruned LightGS model; the latency reduction rate and intersection reduction rate match.

3.2 Intersection-Aware Pruning

The goal of our pruning is to judiciously reduce tile-ellipse intersections without affecting the visual quality. The key to our pruning is a metric that we call *Computational Efficiency* (CE), which intuitively describes how much contribution a point makes to pixel values per unit cost of compute. Intuitively, we would like to prioritize pruning points with low CEs, as they consume a lot of computation without making much contribution to pixel values. For every point i in a dense model, its CE is defined as:

$$\text{CE}_i = \frac{\text{Val}_i}{\text{Comp}_i} \tag{3}$$

Val_i, contribution of a point i to pixel values, is defined as the number of pixels that are "dominated" by that point. A pixel is dominated by a point if and only if that point, among all the points, has the highest numerical contribution to the pixel value during rasterization (Sec. 2.1). The numerical contribution of a point i is quantified by $T_i\alpha_i$ in Eqn. 1a.

Comp_i, the compute cost of a point i, which is ignored in all existing pruning methods, is quantified by the number of tiles that intersect and use (the ellipse of) that point, which directly affects the rendering speed as established above.

In actual rendering, a point will be used in different frames based on the camera pose. Thus, a point's CE is frame-specific; in extreme cases, a point could be outside the camera's viewing frustum and thus makes no contribution to the image. We empirically find that the final CE of a point is adequately measured by the maximum CE across all poses (as opposed to the average, which is susceptible to dataset bias) in the training set.

With this metric, during pruning we sort all the points by their CEs and remove a certain portion of points with the lowest CEs. How many points to remove must be done in conjunction with controlling the quality of the pruned model, which we will discuss in Sec. 3.4.

3.3 Scale Decay

Orthogonal to pruning points, another way to reduce tile-ellipse intersections is to reduce the ellipse size/scale, which we call "scale decay." In particular, we want to focus on scaling ellipses that are both large and are used by a lot of tiles in rendering. To guide scale decay, we propose a metric
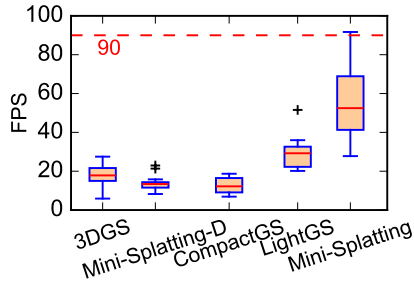
Fig. 3. FPS distribution of recent PBNR models on common datasets measured on mobile Volta GPU on Jetson Xavier.
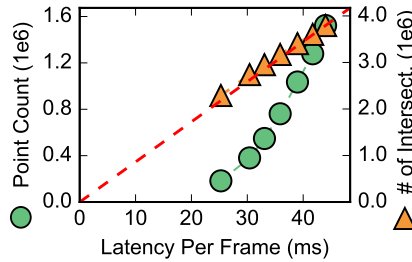
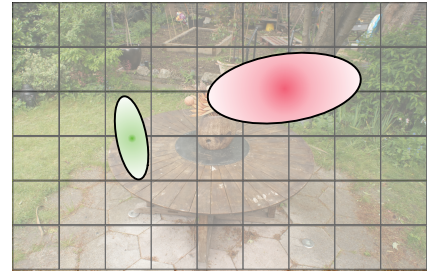**Fig. 4.** Point count vs. latency per frame and the number of tile-ellipse intersections vs. latency per frame.

Fig. 5. Two ellipses intersect different number of tiles so contribute to computation cost differently.
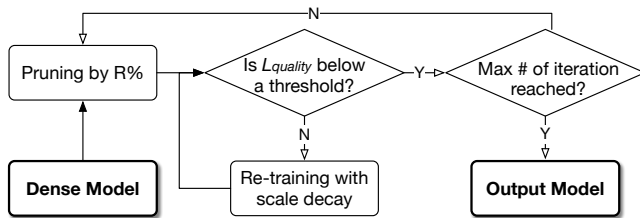

Fig. 6. The procedure to obtain an efficient PBNR model given a dense model. We iteratively apply pruning and re-training with scale decay (guided by $\mathcal{L}$ in Eqn. 6) while controlling for quality ($\mathcal{L}_{quality}$).

called *Weighted Scale* (WS) that weighs the point sizes with how often they are used in rendering:

$$WS = \frac{1}{N} \sum_{i=0}^{N-1} S_i G_i \qquad (4)$$

where N is the number of points, S_i is the scale of point i's ellipse (the maximum span of the ellipse in any direction). Without G_i, WS is simply the average scale of all points in a model. G_i weighs a point's scale by how often it is used in rendering, and is defined as:

$$G_i = (U_i > T) \cdot (U_i - T) \qquad (5)$$

where U_i is the number of tiles a point i is used in rendering and T is a threshold; intuitively, if a point i is used by fewer than T tiles, its scale is insignificant, in which case the G_i is 0 so i does not participate in calculating the average scale. That way, the G term helps suppressing the scale of points that are not only large but are also used often in rendering.

The WS metric is a general metric characterizing point/ellipse scales in PBNR. We empirically find that it is particularly effective when integrated into the training process as an additional term to the loss function $\mathcal{L}$, which ordinarily is concerned only with the rendering quality ($\mathcal{L}_{quality}$):

$$\mathcal{L} = \mathcal{L}_{quality} + \gamma \cdot WS \qquad (6)$$

where γ is a hyper-parameter governing how much scale decay to apply.

3.4 Putting It Together

Pruning and scale decay are conceptually orthogonal, but, importantly, scaling an ellipse's size also changes its CE. Thus, scale decay must be done in conjunction with pruning. Fig. 6 illustrates the general procedure.

Given a dense model, we first compute the CE for all the points, and repetitively prune a small percentage ($R = 10\%$ in our implementation) of the points with the lowest CEs until the quality loss ($\mathcal{L}_{quality}$ in Eqn. 6) is above a prescribed threshold. We then train the pruned model again to regain the quality, but using the composite loss $\mathcal{L}$ in Eqn. 6 in order to apply scale decay. The re-training continues until $\mathcal{L}_{quality}$ is once again below the threshold, at which point we apply intersection-aware pruning again. We iteratively apply pruning and scale decay in such a way until a certain number of iterations is reached.

Note that $\mathcal{L}_{quality}$ is usually PSNR or SSIM but can be any other quality metric of interest. In the next section we will show how we can use a human vision-inspired quality metric to account for the eccentricity dependence of visual quality.

Our iterative procedure has two advantages. First, it combines pruning and scale decay. Second, it does not require quality-specific hyper-parameter tuning to achieve a specific visual quality: monitoring and controlling for $\mathcal{L}_{quality}$ automatically yield a model at a given quality.

4 Foveated PBNR

This section introduces a Foveated Rendering (FR) method tailored to PBNR. We first describe the main idea and its main challenges (Sec. 4.1). We then discuss an efficient data representation that enables effective FR for PBNR (Sec. 4.2). Finally, we describe how to train FR models leveraging the efficiency-aware pruning discussed before (Sec. 4.3).

4.1 Main Idea and Challenges

We accelerate rendering by relaxing the rendering quality at the visual periphery, leveraging the low peripheral visual acuity in HVS. We illustrate the idea in Fig. 7, panel Ⓐ.
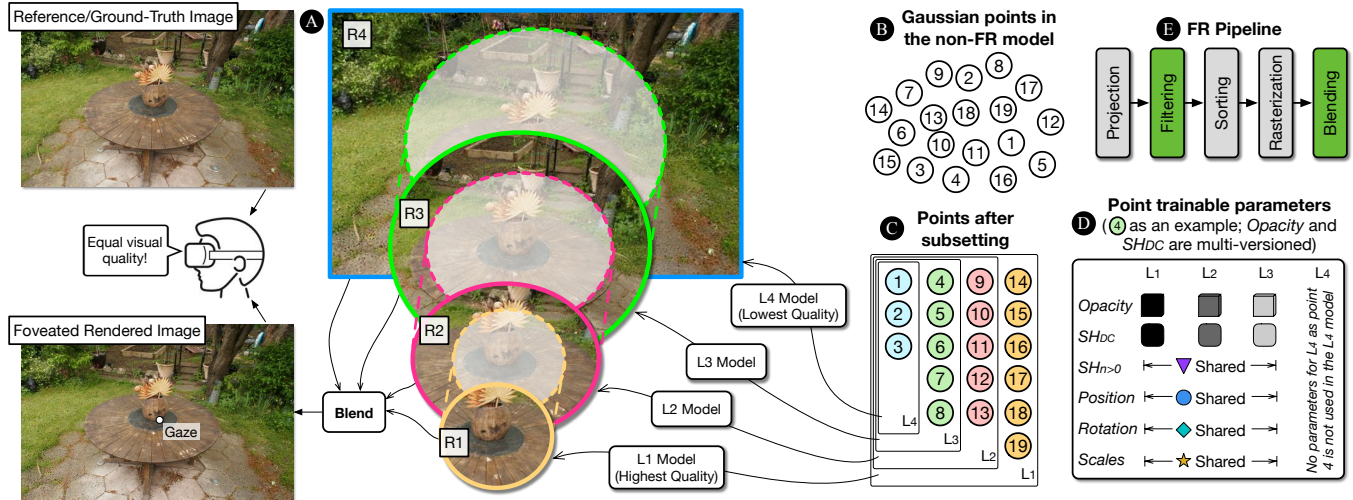
Fig. 7. The general idea of FR for PBNR. **A**: We train multiple models (four in this example), each with a different quality and is responsible for rendering a different quality region in the image ($R1$ – $R4$). The four quality regions are blended together to generate the final image. The goal is for the FR-rendered image to have the same visual quality as the reference image (e.g., generated by a dense model) when judged by humans. **B**: Points in the original non-FR model. **C**: Our hierarchical point representation to support compute- and data-efficient FR. We subset the points so that points used to train a higher-level (lower quality) model are strictly a subset of that used by a lower-level model. The *quality bound m* of a point is the highest level that uses the point (e.g., $m = 3$ for Point 4). **D**: To provide more flexibility for training, we selectively allow key trainable parameters to differ across levels; these parameters are the opacity of a point and the Direct Current (DC) component of the SH coefficients (SH_{DC}). Other (trainable) parameters of a point are shared across all the levels *that use the point* (e.g., no parameter in L_4 for Point 4). **E**: The rendering pipeline augmented to support FR (augmentations in green).

Main Pipeline. As with prior FR work [16, 24, 51], we divide an image into N regions (4 in the example), each corresponding to a quality level and is rendered by a separate model. The region currently under the user's gaze has the highest quality ($R1$ here). Lower-quality regions are rendered using lighter models, which are obtained by applying pruning and scale decay (Sec. 3) to a high-quality model.

Panel **E** shows the rendering pipeline augmented to support FR — with two new stages (green). First, after projection we must *filter* each model's points that are outside the model's quality region. Second, after each region is rendered, we must *blend* the results together to avoid aliasing.

Blending is required in all FR algorithms [24, 51]. Due to the quality difference across levels, there is a sharp, undesirable boundary between two adjacent levels in the rendered image (a form of aliasing). To eliminate the boundary, a common technique is for each model to render slightly beyond its assigned boundary; thus, pixels at the boundary will be rendered twice and then are interpolated/blended to provide a smooth transition between the two levels.

While this multi-model FR idea is conceptually simple, we must address three challenges.

Challenge 1: Performance Overhead. FR can potentially accelerate rendering because it reduces the amount of

rasterization work in low-quality regions. However, it has two sources of performance overhead.

First, all N models must go through the Projection and Filtering stages. In our profiling, these two stages can take up to 18% of the rendering time. Second, blending also adds overhead. Empirically we find that about 25% of the pixels are to be blended and, thus, rendered twice.

Challenge 2: Storage Overhead. FR could increase the model size due to the need to store multiple models, exacerbating the storage pressure of PBNR models. For instance, the `bicycle` scene in the Mip-NeRF 360 dataset [7] takes about 1.4 GB of space when trained with 3DGS [34]; recent pruning methods [17] reduce the model size of that scene to about 490 MB, which is still large for mobile devices.

We address the first two challenges using an efficient data representation, as we will discuss in Sec. 4.2.

Challenge 3: Controlling Quality. FR must be done in a way that guarantees human visual quality — how do we decide the amount of relaxation at each level? We describe a training strategy to guarantee consistent human visual quality across all levels, as described in Sec. 4.3.

4.2 Efficient Data Representation

To address both the performance and storage overhead, we propose an efficient data representation that allows models at
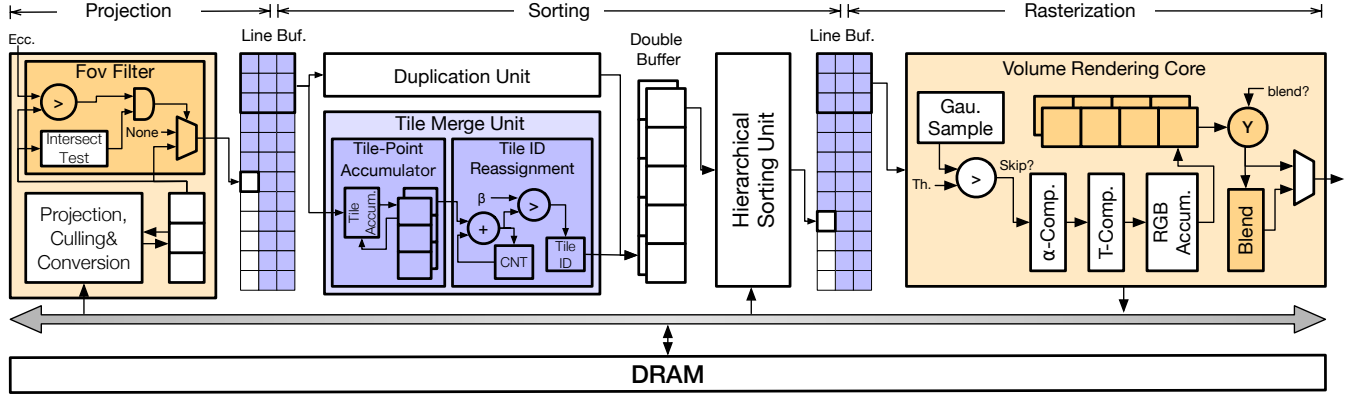
Fig. 8. The overall architecture design. The the basic pipeline is similar to that of GSCore [39], a recent PBNR accelerator. We augment the baseline to support FR (yellow-colored) and to address the workload imbalance issue in PBNR/FR (blue-colored).

different quality levels to share computation and parameters. The key idea is that points used to train and render a lower quality level are strictly a subset of the points used by a higher quality level. Panel **C** in Fig. 7 illustrates how the original points in Panel **B** are organized after subsetting. Level 1 (L_1) model is trained with the most points and thus would offer the highest quality, and Level 4 (L_4) model has the fewest points and lowest quality.

Subsetting mitigates both the performance and storage overhead, because the total number of points across all N models, P_{total}, is the same as that of the highest-quality model, P_1, rather than the sum of all N models. That is, $P_{total} = \max_{i=1}^N P_i = P_1 < \sum_{i=1}^N P_i$. As a result, there is no storage overhead. The compute overhead is small too, since the Projection and Filtering stages are executed only once, rather than once for each of the N models.

Under subsetting, each point is simultaneously used in models $[L_1, \cdots, L_m]$, where m is the highest level beyond which the point is not used and is called the *quality bound* of the point. For instance in Fig. 7, $m = 3$ for Points 4. During the Projection stage, each point is projected to a tile, which has a specific eccentricity and thus a corresponding quality level t. If $t > m$, the point does not participate in the rest of rendering. This is the Filtering stage in Panel **E**.

Selective Multi-Versioning. Practically, strict subsetting is likely too restrictive in controlling the rendering quality at different levels. This is because all the trainable parameters of a point would be fixed across all levels, so how a point participates in calculating pixel colors (the $\alpha_i c_i$ term in Eqn. 1a) is also fixed at any time. In reality, however, a point's contribution to pixel colors should vary depending on the quality region the point is projected to, which varies with the camera pose and the gaze position.

To relax this, we allow multi-versioning as illustrated in panel **D**: a point can maintain m (where m is the quality

bound of the point) versions of *some* of its trainable parameters, one version for each level the point is in. Empirically, we allow four such parameters, i.e., the Opacity and the Direct Current component of the SH coefficients (SH_{DC}); these four parameters are empirically found to impact the pixel colors the most. We will show in Sec. 7.4 that selective multi-versioning is critical to maintain high visual quality.

4.3 HVS-Guided Training

The discussion so far has focused on performance, but equally important to FR is the visual quality: how much weaker can higher-level models be while maintaining subjectively good visual quality across quality levels/eccentricies?

To answer this question, we turn to the HVSQ metric discussed in Sec. 2.2. The HVSQ metric quantifies the subjective visual quality between a reference image (e.g., rendered from a dense PBNR model) and an altered image (e.g., rendered from a pruned PBNR model), accounting for the eccentricity-dependent visual acuity of HVS. Conveniently, while the vanilla HVSQ metric in Eqn. 2 is applied to an entire image, it can be easily adapted to a selected region — by simply iterating over the spatial poolings (pixels) in the selected region rather than over the entire image.

That way, each quality region has a unique HVSQ measure, and our goal is to ensure the HVSQs across all quality levels are similar to the HVSQ of the baseline model. To that end, we first train the highest-quality, L_1 model, which itself can be pruned and scale-decayed from a dense model. We then prune a L_1 model to obtain a L_2 model, which is pruned down to obtain a L_3 model; this continues until the desired level is achieved. The way to obtain a L_{i+1} model follows the exact procedure as laid out in Sec. 3.4 (i.e., iteratively apply pruning and re-training to a L_i model while controlling for the quality loss $\mathcal{L}_{quality}$) — with two key differences.

First, instead of using the usual PSNR/SSIM metrics, we use HVSQ as $\mathcal{L}_{quality}$ in Eqn. 6. In particular, when obtaining

a L_i model we use the HVSQ corresponding to level i. We control for $\mathcal{L}_{quality}$ so that the HVSQ at all quality levels is the same as that of L_1 such that the human visual quality is consistent *across the entire visual field.* Second, during iterative re-training we do not apply scale decay, because an ellipse scale is not part of the multi-versioned parameters.

5 Hardware Support

Complementing pruning and FR techniques, we propose an accelerator to further improve the performance. We first provide an overview (Sec. 5.1) and discuss how the hardware addresses the low hardware utilization issue in PBNR, which is exacerbated by FR (Sec. 5.2).

5.1 Overview

Our architecture is built on top of GSCore [39], a recent PBNR accelerator without FR. The basic architecture is designed to support the three PBNR stages discussed in Sec. 2.1. The three stages are pipelined across different tiles in a frame. Fig. 8 shows the pipelined architecture, with the colored components denoting our augmentations. The top panel in Fig. 10 illustrates the pipelining process (omitting the Projection stage for simplicity).
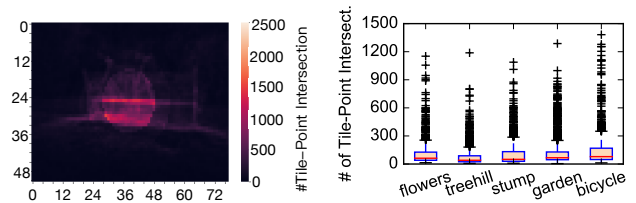
Supporting FR. We propose two sets of hardware augmentations to support the two new stages in FR (green in panel **E**). The augmentations are colored in yellow in Fig. 8. First, during the Projection stage we must filter out points that are not used to render a particular quality level. To that end, we augment the Projection hardware to include a filtering unit, which compares the quality bound m of a tile with the current quality level t of that tile, and pushes the tile to the output buffer only if $t > m$.

Second, FR requires blending across quality levels (Sec. 4.1). We augment the Rasterization stage with a blending unit, which takes the two colors of a pixel (rendered by two models corresponding to the quality levels) and performs an interpolation. A small buffer is also needed to temporarily store pixels before blending.

5.2 Addressing Load Imbalance

The Issue. With the augmentations above, the hardware can functionally support FR, but faces low hardware utilization. This is because the amount of work each tile requires is severely imbalanced (recall a tile, like an instruction in a conventional processor, is a basic unit for pipelining; Fig. 10).

The amount of work a tile involves can be quantified by the number of tile-ellipse intersections. To quantify this imbalance, Fig. 9a is a heatmap plotting the number of intersections each tile (16×16 pixels) has when rendering an image in the Mip-NeRF 360 dataset using our FR model (with four quality levels). The amount of intersections can vary by over three orders of magnitude. In particular, most of the intersections are concentrated at the center, because the peripheral

(a) Heatmap showing the number of intersections per tile in `bicycle`.

(b) Boxplot of the intersection distribution in five traces (clipped at 1,500).

Fig. 9. Workload imbalance, quantified by the number of intersections per tile, on the Mip-NeRF 360 dataset [8]. The top and bottom notches in the boxplot represent data points that are 1.5 Interquartile Range (IQR) above the third quartile and below the first quartile, respectively.

tiles are rendered using pruned models, which have fewer points. Fig. 9b is the boxplot showing large the distribution of intersections across all traces in the Mip-NeRF 360 dataset [8]; the imbalance issue is universal.

Load imbalance leads to frequent pipeline stalls. This is illustrated by the top panel in Fig. 10, which shows the pipeline dynamics when pipelining Sorting and Rasterization across four imbalanced tiles. We propose two techniques to address this issue: tile merging and incremental pipelining. The corresponding hardware components are blue-colored in Fig. 8.

Tile Merging. One way to balance the workload across tiles is to merge tiles that have few intersections. This is dealt with by the Tile Merge Unit (TMU) in the Sorting stage. Conceptually, the TMU merges incoming tiles into a single tile if the cumulative intersections is below a threshold β. The second panel in Fig. 10 shows an example where the second and third tiles are merged, which reduces pipeline stalls and improves performance.

The TMU has two stages. The first stage processes a Gaussian point by incrementing its counter associated with a specific tile ID, storing the result in a temporal buffer. When the accumulation for one tile is complete, its total count is forwarded to the second stage, which continuously aggregates incoming tiles, accumulates the intersection counts, and compares the cumulative count against β. If this threshold is exceeded, a merged-tile is formed, where each constituting tile is augmented with a merged-tile ID, which is sent along with the native tile ID to the sorting unit.

Incremental Pipelining. While Tile Merging reduces the workload imbalances, it does not completely eliminate pipeline stalls, because it is unlikely all the merged tiles have exactly the same number of intersections.

To further enhance pipelining efficiency, we propose to incrementally pipeline data between adjacent stages. In our baseline PBNR accelerator, a double buffer is placed between two adjacent stages; the consumer stage does not start until the producer stage has finished processing the entire tile. Our
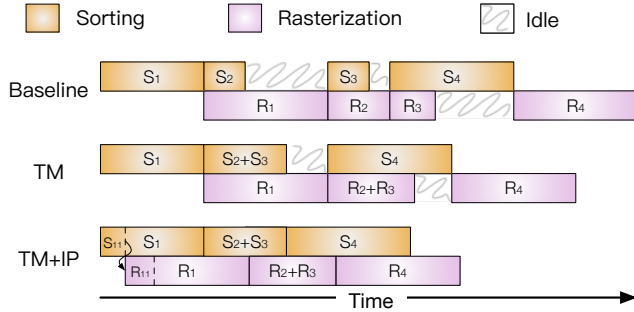
Fig. 10. The baseline pipeline faces frequent stalls when the workload is imbalanced across tiles. Tile Merging (TM) and Incremental Pipelining (IP) mitigate the workload imbalanced issue and improves the pipelining efficiency. With IP, when the first sub-tile S_{11} in the S_1 tile is available by the Sorting stage it can be processed by the Rasterization stage.

idea is to break the workload of a tile into smaller sub-tiles so that the consumer stage can start working on available sub-tiles before the entire tile is ready from the producer stage. This works because the workload of each pixel is independent. This is akin to classic superpipelining in processor design [47, 64]. The last panel in Fig. 10 illustrates the benefit of incremental pipelining.

To support such an incremental computation, we replace the double buffers between stages with line buffers (LB) [27, 28, 56, 71], which, under the surface, is a set of small SRAMs, each of which buffers one row of pixels. Assume a 16×16 tile size, once 16 rows are produced in the LB, the consumer can starting using them. The line buffer can be small, since it has to buffer only sub-tiles rather than entire tile.

6 Experimental Setup

FR Training Procedure. We use four quality regions whose eccentricity starts at 0°, 18°, 27°, and 33°, respectively, corresponding to about 13%, 17%, 21%, 49% of image pixels in these four regions, respectively.

We use Mini-Splatting-D [18], the current-best in quality, as the baseline dense PBNR model. The L_1 model is obtained from the dense model through pruning and scale decay as described in Sec. 3.4, with an iteration budget of 50,000, followed by another 5,000 iterations of fine-tuning with HVSQ loss. The three lower-quality models are obtained from their immediately higher-quality model as described in Sec. 4.3, with a 7,500 iteration budget. Our training time is roughly three times as much as that of the original, dense model. Much of the slow down is because the HVS loss is calculated using an open-source Python implementation [5], which could be accelerated with a more efficient implementation in, e.g., CUDA.

Variants. We design three variants of our method, namely MetaSapiens-H, MetaSapiens-M, and MetaSapiens-L, with

decreasing rendering quality. The L_1 model in the three variants is pruned to have a PSNR of 99%, 98%, and 97% of that of the dense model. The total model size of the three variants is 16%, 12%, and 10%, respectively, of that of the dense model.

Datasets. We evaluate three real-world datasets: Mip-Nerf360 [8], Tanks & Temple [35], and DeepBlending [26], which amounts to 13 traces in total. The camera poses in the datasets are usually very sparsely populated, which is not representative of the continuous rendering scenario (e.g., VR). We interpolate between the poses in the dataset to create smooth trajectories, producing approximately 1,440 poses for each trace, corresponding to a 16-second video at 90 FPS.

Hardware Implementation. We develop a RTL implementation of the accelerator, where the basic pipeline (the uncolored in Fig. 8) is similar to GScore [39], with the resource allocation adjusted for a more balanced pipeline for our workloads (8 Culling and Conversion Units, a single Hierarchical Sorting Unit, and a 16×16 Volume Rendering Core array). Our RTL design is implemented via Synposys synthesis and Cadence layout tools in TSMC 16nm FinFET technology. Each line buffer has a capacity of 1 KB, and the double buffer before the sorting unit is 64 KB. The SRAMs are generated by an Arm compiler. The DRAM is modeled after four channels of Micron 16 Gb LPDDR3-1600 memory [1].

Overall, we have an area of 2.73 mm^2. The volume Rendering Core takes 63% of the total area, other stages occupy the rest 30 %; the SRAMs comprise 7% of the total area. Our area is larger than that of GScore (1.45 mm^2), whose area is scaled to 16nm using the DeepScaleTool [63]. We will show in Sec. 7.5 that we outperform GScore even under the same area when the latter is scaled up.

Baselines. We compare against five recent PBNR models:

- Dense PBNR models: 3DGS [34], which is the earliest PBNR model, and Mini-Splatting-D [18], Mip-Splatting [77], StopThePop [54], which are state-of-the-art work that improve upon 3DGS.
- Pruned PBNR models: LightGS [17], CompactGS [40], and Mini-Splatting [18]. The first two are pruned from 3DGS and the last one is from Mini-Splatting-D.

We also compare with two FR methods applied to PBNR. Both methods use the same quality-region division as in our method. The first one is SMFR (Single-Model FR), which uses a single dense PBNR model, which is the L_1 model in MetaSapiens-H, and randomly samples the points when rendering lower-quality regions. It is effectively a strict sub-setting version of our model without selective multi-versioning. The second one is MMFR (Multi-Model FR) [16], whose L_1 model is the same as that of MetaSapiens-H and whose higher-level models are pruned from its L_1 model separately (without subsetting). The number of points used in each level in both methods matches that used in our method.
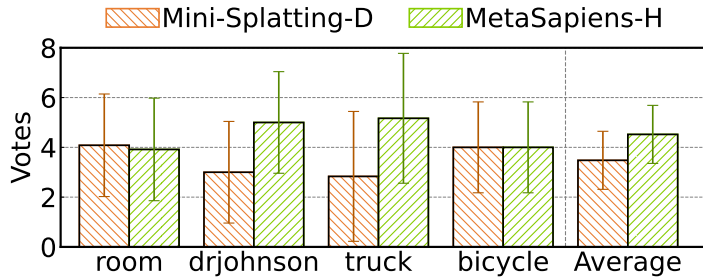
Weikai Lin, Yu Feng, and Yuhao Zhu

Fig. 11. The average number of times the two methods are preferred by users (a tie would be 4-vs-4). Error bars indicate the standard deviation within the participants. Users either have no preference or prefer our method (binomial test on the average result; $p < 0.01$).

Fig. 12. Ablation study teasing apart the impact of various techniques. The FPS results are obtained on Jetson Xavier and averaged over all traces.

User Study Procedure. To assess the subjective rendering quality of our method, we perform a user study; the procedure is approved by our Internal Review Board (IRB). We recruited 12 participants (8 males and 4 females between 20 and 30 years old), all with normal or corrected-to-normal vision. The scale of our user study is comparable with other research in the field [16, 60, 65, 76]. We select four scenes from the three datasets: bicycle, room, drjohnson, and truck, which vary in both content and complexity. We then render the scenes using our MetaSapiens-H and Mini-Splatting-D, which, recall, is the current-best in rendering quality.

Since Mini-Splatting-D does not render in real-time on a mobile device, we use a workstation with an RTX 4090 GPU to execute both models, both of which render smoothly at 90 FPS. The workstation streams, in real-time, the rendering to a Meta Quest Pro headset, which has an eye tracker to track the user's real-time gaze.

We use the classic Two-Interval Forced Choice (2IFC) psychophysical procedure [12, 52], which is commonly used for evaluating the subjective quality of foveated rendering [16, 24, 60, 65, 72, 74, 76, 78]. For each trace, we display its rendering by the two methods on the headset in a random order to each participant, with a 5-second rest interval in-between. Each participant is then asked to pick which of the two versions they prefer. Each trace is repeated eight times, and the repetitions across traces are also randomized. The entire experiment lasts about one hour for each participant.

7 Evaluation

We first show that the subjective rendering quality of our method is statistically no-worse than Mini-Splatting-D (Sec. 7.1). We then show that we provide a better speed-quality trade-off than virtually all baselines on a mobile GPU (Sec. 7.2); hardware acceleration improves the speed even further (Sec. 7.3). We out-perform other FR methods (Sec. 7.4) and a prior PBNR accelerator (Sec. 7.5).
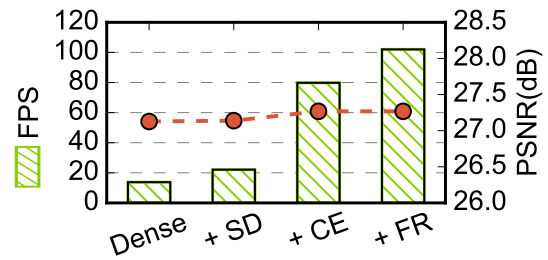
7.1 Subjective Experiments.

Fig. 11 shows the average number of participants who prefer the two methods for each video. A tie would be 4-vs-4, since each video is watched eight times by each user. We find that users either have no preference or prefer our method over Mini-Splatting-D. The results are statistically significant through a binomial test with $p < 0.01$; the null hypothesis is "users prefer Mini-Splatting-D more than 50% of the time".

It might initially look surprising that we have equal or better subjective quality than Mini-Splatting-D, a dense model from which we prune and build our FR model. Further inspection and interviewing participants show two reasons. First, our HVS-aware fine-tuning (Sec. 6) better aligns the statistics of human-sensitive features with the ground truth. Second, some points in the dense model are trained with inconsistent information across camera poses, leading to incorrect luminance changes over time; pruning those points helps alleviate this inconsistency.

7.2 GPU Results

We now show the performance results on the mobile Volta GPU on Nvidia Jetson AGX Xavier [2], a representative mobile device for use-cases such as VR. Fig. 13 shows the results. We execute each model five times for each camera pose in each scene in all the datasets, and report the average FPS.

To put the performance results in context, we compare our variants with the baselines using three objective metrics, PSNR, SSIM, and LPIPS, which provide good quality measures for the region under the user's gaze and are commonly reported in prior work. Using the objective metrics also allows us to scale up the study to more traces.

Our three variants provide better speed-quality trade-offs in virtually all metrics. Our slowest variant MetaSapiens-H is 1.9× faster than the fastest baseline while having better or similar objective quality. The fastest variant MetaSapiens-L is 7.9× faster than 3DGS, and can be up to 19.8× on the largest bicycle trace.
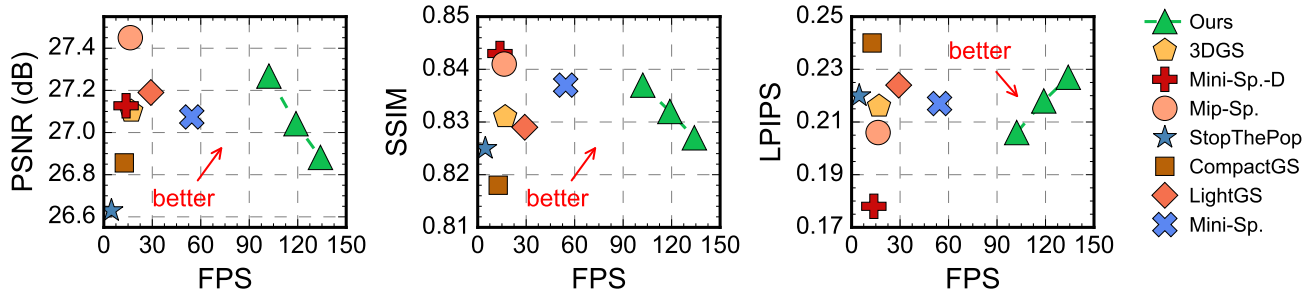
Fig. 13. Performance and objective rendering quality (PSNR, SSIM, LPIPS) comparison across the seven baselines and the three MᴇᴛᴀSᴀᴘɪᴇɴs variants on the mobile Volta GPU. 3DGS, Mɪɴɪ-Sᴘʟᴀᴛᴛɪɴɢ-D, Mɪᴘ-Sᴘʟᴀᴛᴛɪɴɢ, and SᴛᴏᴘTʜᴇPᴏᴘ are dense models, and the other three baselines are pruned models.
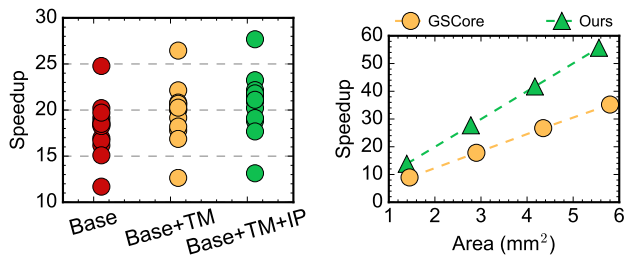

Fig. 14. Speed-ups of different accelerator variants over the GPU baseline; each marker is a dataset trace.

Fig. 15. Speedup and area comparison between our hardware and GSCore under different configurations.

Table 1. Comparison of FR methods.

Methods	FPS ↑	Storage (MB) ↓	HVS Quality (×10⁻⁵)↓			
			L1	L2	L3	L4
SMFR	125.9 (1×)	161.6 (1×)	2.12	10.1	21.7	28.3
MMFR	52.6 (0.42×)	311.0 (1.92 ×)	2.12	1.87	1.79	1.76
MetaSapiens-H	102.2 (0.81×)	171.8 (1.06×)	2.12	2.10	2.09	2.08

Ablation Studies. We now ablate the contribution of various performance-enhancing techniques. Fig. 12 shows the FPS (left y-axis) and PSNR (right y-axis) under: 1) the dense Mɪɴɪ-Sᴘʟᴀᴛᴛɪɴɢ-D model, 2) MᴇᴛᴀSᴀᴘɪᴇɴs with only scale decay (SD; Sec. 3.3), 3) MᴇᴛᴀSᴀᴘɪᴇɴs with SD and CE-based pruning (Sec. 3.2), and 4) MᴇᴛᴀSᴀᴘɪᴇɴs with SD, CE, and FR (Sec. 4). We use the MᴇᴛᴀSᴀᴘɪᴇɴs-H model and obtain the FPS/PSNR results on Xavier averaged over all traces.

The PSNRs for all the variants are similar. With a similar quality, our SD implementation achieves 1.6× speedup compared to original dense model; CE-based pruning and FR bring the speedup to 5.8× and 7.4×, respectively. CE reduces the model size by 85%, and FR diminishes the pruning rate only marginally to 84% owing to selective multi-versioning,.

7.3 Results with Hardware Support

Speedup. Our hardware support provides further performance improvements. Fig. 14 shows the speedups over GPU of: 1) the base accelerator (the uncolored in Fig. 8), 2) the accelerator with only Tile Merging, and 3) one with both Tile Merging and Incremental Pipelining. Each marker represents one of the 13 dataset traces. We use the MᴇᴛᴀSᴀᴘɪᴇɴs-H model for evaluation. Overall, even the base accelerator achieves a 18.5× speedup (geomean), up to 24.8×, compared to the GPU baseline across different datasets.

The introduction of Tile Merging consistently improves performance, because tile merging mitigates the load imbalance across tiles. On top of that, Incremental Pipelining completely addresses the load imbalance in FR. Overall, MᴇᴛᴀSᴀᴘɪᴇɴs-TM-IP combines both techniques and achieves an average 20.9× (up to 27.7×) speedup.

Energy Savings. We summarize the energy results. Our base accelerator achieves a 54.4× energy reduction compared to the GPU baseline. MᴇᴛᴀSᴀᴘɪᴇɴs-TM-IP improves the energy saving to 56.8×; this is primarily because with incremental pipelining we can afford to smaller SRAMs as line buffers, which reduces the energy consumption of SRAMs.

7.4 Comparison with Other FR Methods

MᴇᴛᴀSᴀᴘɪᴇɴs also out-performs the two FR baselines. Tbl. 1 compares the FPS (on the mobile Volta GPU), storage requirement, and the HVSQ metric across different quality regions/layers. The results are averaged across all the datasets.

SMFR has been seen as a variant of our FR with strict subsetting (no multi-versioning). Thus, it is the fastest, but has excessively low visual quality, because it simply sub-samples pre-trained points to render low-quality regions. Its HVSQ in L_4 is over 10× worse than the other two methods. Users confirm that subjectively this gives the worst quality.

Our method selectively multi-versioned four trainable parameters out of about 60 (Sec. 4.2 and ⓓ in Fig. 7), so the additional storage requirement is small (about 6%). Note that MᴇᴛᴀSᴀᴘɪᴇɴs-H already reduces the dense model size to 16% as shown in Sec. 6.

MMFR can be seen as a variant of our FR that multi-versions *all* parameters. It is thus the slowest of the three —

its FPS is way below a 90 FPS real-time requirement, and has the largest storage requirement. This is due to the compute and storage overhead discussed in Sec. 4.1. Its HVSQ metrics in higher levels (lower-quality regions) are better than that of ours. Given that our method is already subjectively no-worse than even a dense model (Sec. 7.1), this suggests that MMFR unnecessarily optimizes for details that are imperceptible to users, which we confirm with users.

7.5 Comparison with GSCore

Our accelerator is based on GSCore [39], but has a larger area (Sec. 6). This is because our baseline hardware (uncolored in Fig. 8) has 4× more Volume Rendering Cores compared to that of GSCore with 2× fewer sorting unit to balance the latency of different stages.

Fig. 15 compares the speedup over GPU and area between our architecture (with TM and IP) and GSCore; both hardware execute the MetaSapiens-H model on the flowers scene. We proportionally scale both GSCore and ours based on their own resource ratio. We consistently achieve higher speedups with a slightly smaller area against GSCore. For example, at an area of around 6 mm², MetaSapiens outperforms GSCore by 1.6×. The higher performance of MetaSapiens is from the effectiveness of TM and IP that avoid stalling when pipelining over tiles. Our performance gain is more significant as the area increases, because the workload imbalance is more pronounced with a large amount of idle resource.

8 Related Work

Foveated Rendering. The graphics community has long exploited FR for real-time rendering [10, 13, 24, 32, 36, 37, 51, 66, 70]. Particularly relevant to our work, researchers have started applying FR to neural rendering [16, 59, 60], such as Fov-NeRF [16]. Our work differs from them in two key ways. First, these methods exclusively focus on NeRF, whereas we focus on PBNR, which is shown to be fundamentally more efficient that NeRF. Second, some [16] use the multi-model approach, similar to our MMFR baseline, which we out-perform (Sec. 7.4). Our FR method uses subsetting with selectively multi-versioning, addressing the performance overhead of evaluating multiple models (Sec. 4.1).

While conventional FR uses heuristics (e.g., blurring) to guide quality relaxation, recently researchers have investigated more principled ways to model human perception for quality relaxation [20, 72]. This work leverages such theoretical work and integrates it into the training framework to demonstrate its practical utility.

Efficient PBNR. Almost all existing work optimizing PBNR focuses on pruning, based on the observation that a considerable amount of points can be pruned without impacting the rendering quality. They usually do so by, e.g., explicitly training a mask to remove points [40] or sorting points by their numerical contribution to pixel colors followed by removing low-contributing points [17, 18, 22, 50]. People have also investigated non-pruning methods, such as vector quantization [17] and distillation [40] techniques, to compress PBNR models.

Our work differs from them in two key ways. First, we show that point count is not indicative of performance; tile intersections are (Sec. 3.1). We propose an intersection-aware metric to guide pruning (Sec. 3.2). Second, we show an orthogonal technique, scale decay, that complements pruning (Sec. 3.3) and can be performed in conjunction with pruning to further achieve improve performance (Sec. 3.4).

Neural Rendering Accelerators. Significant work has been done on accelerating neural rendering [19, 21, 38, 43, 44, 48, 55], but most of them focuses on NeRF, a variant of neural rendering that PBNR aims to out-perform. GSCore [39] accelerates 3DGS, a particular PBNR model.

Our work differs from prior PBNR accelerators in two ways. First, our hardware supports FR with minimal hardware augmentations. Second, we identify and address the load imbalance issue in PBNR, which is exacerbated by FR.

9 Conclusions

We achieve over an order of magnitude speedup over existing PBNR models with no subjective quality loss through a user study. The speedup comes from: 1) a pruning techniques that directly optimizes for the compute-cost of PBNR, 2) FR specialized for PBNR, and 3) hardware support addressing the load imbalanced issue in FR-based PBNR.

Acknowledgments

References

[1] 2014. Micron 178-Ball, Single-Channel Mobile LPDDR3 SDRAM Features. https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/178b_8-16gb_2c0f_mobile_lpddr3.pdf

[2] 2018. NVIDIA Reveals Xavier SOC Details. https://www.forbes.com/sites/moorinsights/2018/08/24/nvidia-reveals-xavier-soc-details/amp/

[3] 2021. VIVE Pro 2 Headset Specifications. https://www.vive.com/us/product/vive-pro2/specs/.

[4] 2022. Meta Quest Pro Specifications. https://vr-compare.com/headset/metaquestpro. Accessed: 2024-06-24.

[5] 2022. Metameric Loss. https://github.com/kaanaksit/odak/blob/master/odak/learn/perception/metameric_loss.py.

[6] 2024. Apple Vision Pro Specifications. https://www.apple.com/apple-vision-pro/specs/. Accessed: 2024-06-24.

[7] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. 2021. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5855–5864.

[8] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. 2022. Mip-nerf 360: Unbounded anti-aliased neural

radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5470–5479.

[9] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? *Proceedings of machine learning and systems* 2 (2020), 129–146.

[10] P. Chakravarthula, Z. Zhang, O. Tursun, P. Didyk, Q. Sun, and H. Fuchs. 2021. Gaze-Contingent Retinal Speckle Suppression for Perceptually-Matched Foveated Holographic Displays. *IEEE Transactions on Visualization and Computer Graphics* 27, 11 (2021), 4194–4203.

[11] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision*. Springer, 333–350.

[12] K. Chen, T. Wan, N. Matsuda, A. Ninan, A. Chapiro, and Q. Sun. 2024. PEA-PODs: Perceptual Evaluation of Algorithms for Power Optimization in XR Displays. *ACM Transactions on Graphics* 43, 4 (July 2024), 67.

[13] S. Chen, B. Duinkharjav, X. Sun, L.-Y. Wei, S. Petrangeli, J. Echevarria, C. Silva, and Q. Sun. 2022. Instant Reality: Gaze-Contingent Perceptual Optimization for 3D Virtual Reality Streaming. *IEEE Transactions on Visualization and Computer Graphics* 28, 5 (2022), 2157–2167.

[14] Christine A Curcio, Kenneth R Sloan, Robert E Kalina, and Anita E Hendrickson. 1990. Human photoreceptor topography. *Journal of comparative neurology* 292, 4 (1990), 497–523.

[15] Dennis M Dacey. 1993. The mosaic of midget ganglion cells in the human retina. *Journal of Neuroscience* 13, 12 (1993), 5334–5355.

[16] Nianchen Deng, Zhenyi He, Jiannan Ye, Budmonde Duinkharjav, Praneeth Chakravarthula, Xubo Yang, and Qi Sun. 2022. Fov-nerf: Foveated neural radiance fields for virtual reality. *IEEE Transactions on Visualization and Computer Graphics* 28, 11 (2022), 3854–3864.

[17] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2024. LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS. In *Advances in Neural Information Processing Systems*. To appear.

[18] Guangchi Fang and Bing Wang. 2024. Mini-Splatting: Representing Scenes with a Constrained Number of Gaussians. In *Proceedings of the European Conference on Computer Vision (ECCV)*. To appear.

[19] 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 *Proceedings of the 50th Annual International Symposium on Computer Architecture*.

[20] Jeremy Freeman and Eero P. Simoncelli. 2011. Metamers of the Ventral Stream. *Nature Neuroscience* 14, 9 (Sept. 2011), 1195–1201.

[21] Yonggan Fu, Zhifan Ye, Jiayi Yuan, Shunyao Zhang, Sixu Li, Haoran You, and Yingyan Lin. 2023. Gen-NeRF: Efficient and Generalizable Neural Radiance Fields via Algorithm-Hardware Co-Design. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–12.

[22] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. 2024. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In *Proceedings of the European Conference on Computer Vision (ECCV)*. To appear.

[23] Markus Gross and Hanspeter Pfister. 2011. *Point-based graphics*. Elsevier.

[24] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D graphics. *ACM transactions on Graphics (ToG)* 31, 6 (2012), 1–10.

[25] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015).

[26] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. 2018. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)* 37, 6 (2018), 1–15.

[27] James Hegarty, John Brunhaver, Zachary DeVito, Jonathan Ragan-Kelley, Noy Cohen, Steven Bell, Artem Vasilyev, Mark Horowitz, and Pat Hanrahan. 2014. Darkroom: compiling high-level image processing code into hardware pipelines. *ACM Trans. Graph.* 33, 4 (2014), 144–1.

[28] John L Hennessy and David A Patterson. 2019. Chapter 7.7 Pixel Visual Core, a Personal Mobile Device Image Processing Unit. In *Computer Architecture: a Quantitative Approach*. Elsevier, 579–592.

[29] Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*. IEEE, 2366–2369.

[30] Qing Jin, Linjie Yang, and Zhenyu Liao. 2020. Adabits: Neural network quantization with adaptive bit-widths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2146–2156.

[31] Maria G Juarez, Vicente J. Botti, and Adriana S. Giret. 2021. Digital Twins: Review and Challenges. *Journal of Computing and Information Science in Engineering* 21 (2021), 030802:1–030802:23.

[32] A. S. Kaplanyan, A. Sochenov, T. Leimkühler, M. Okunev, T. Goodall, and G. Rufo. 2019. Deepfovea: Neural Reconstruction for Foveated Rendering and Video Compression Using Learned Statistics of Natural Videos. *ACM Transactions on Graphics* 38, 6 (Nov. 2019). https://doi.org/10.1145/3355089.3356557

[33] Arie Kaufman, Daniel Cohen, and Roni Yagel. 1993. Volume graphics. *Computer* 26, 7 (1993), 51–64.

[34] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (Aug. 2023), 1–14. https://doi.org/10.1145/3592433

[35] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *ACM Transactions on Graphics* 36, 4 (2017).

[36] R. Konrad, A. Angelopoulos, and G. Wetzstein. 2020. Gaze-Contingent Ocular Parallax Rendering for Virtual Reality. *ACM Transactions on Graphics* 39 (2020).

[37] B. Krajancich, P. Kellnhofer, and G. Wetzstein. 2020. Optimizing Depth Perception in Virtual and Augmented Reality through Gaze-Contingent Stereo Rendering. *ACM Transactions on Graphics* 39 (2020).

[38] 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*. 1–13.

[39] 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*. 497–511.

[40] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. 2024. Compact 3d gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 21719–21728.

[41] Marc Levoy. 1988. Display of surfaces from volume data. *IEEE Computer graphics and Applications* 8, 3 (1988), 29–37.

[42] Marc Levoy and Turner Whitted. 1985. The use of points as a display primitive. (1985).

[43] 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*. 1–9.

[44] Sixu Li, Chaojian Li, Wenbo Zhu, Boyang Yu, Yang Zhao, Cheng Wan, Haoran You, Huihong Shi, and Yingyan 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*. 1–13.

[45] Nelson Max. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 1, 2 (1995), 99–108.

[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] Sunil Mirapuri, Michael Woodacre, and Nader Vasseghi. 1992. The MIPS R4000 processor. *IEEE Micro* 12, 2 (1992), 10–22.

[48] Muhammad Husnain Mubarik, Ramakrishna Kanungo, Tobias Zirr, and Rakesh Kumar. 2023. Hardware Acceleration of Neural Graphics. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–12.

[49] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)* 41, 4 (2022), 1–15.

[50] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. 2024. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. *arXiv preprint arXiv:2403.13806* (2024).

[51] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016. Towards foveated rendering for gaze-tracked virtual reality. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–12.

[52] Maria Perez-Ortiz, Aliaksei Mikhailiuk, Emin Zerman, Vedad Hulusic, Giuseppe Valenzise, and Rafał K Mantiuk. 2019. From pairwise comparisons and rating to a unified quality scale. *IEEE Transactions on Image Processing* 29 (2019), 1139–1151.

[53] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. 2000. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 335–342.

[54] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. 2024. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. *ACM Transactions on Graphics* 4, 43, Article 64 (2024).

[55] Chaolin Rao, Huangjie Yu, Haochuan Wan, Jindong Zhou, Yueyang Zheng, Minye Wu, Yu Ma, Anpei Chen, Binzhe Yuan, Pingqiang Zhou, et al. 2022. ICARUS: A Specialized Architecture for Neural Radiance Fields Rendering. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–14.

[56] Jason Redgrave, Albert Meixner, Nathan Goulding-Hotta, Artem Vasilyev, and Ofer Shacham. 2018. Pixel Visual Core: Google's Fully Programmable Image, Vision, and AI Processor For Mobile Devices. In *Proc. IEEE Hot Chips Symp.(HCS)*. 1–18.

[57] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. 2002. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum*, Vol. 21. Wiley Online Library, 461–470.

[58] RW Rodieck, KF Binmoeller, and J Dineen. 1985. Parasol and midget ganglion cells of the human retina. *Journal of Comparative Neurology* 233, 1 (1985), 115–132.

[59] Tim Rolff, Ke Li, Julia Hertel, Susanne Schmidt, Simone Frintrop, and Frank Steinicke. 2023. Interactive VRS-NeRF: Lightning fast Neural Radiance Field Rendering for Virtual Reality. In *Proceedings of the 2023 ACM Symposium on Spatial User Interaction*. 1–3.

[60] Tim Rolff, Susanne Schmidt, Ke Li, Frank Steinicke, and Simone Frintrop. 2023. VRS-NeRF: Accelerating Neural Radiance Field Rendering with Variable Rate Shading. In *2023 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 243–252.

[61] Ruth Rosenholtz. 2016. Capabilities and limitations of peripheral vision. *Annual review of vision science* 2 (2016), 437–457.

[62] Szymon Rusinkiewicz and Marc Levoy. 2000. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 343–352.

[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)*. IEEE, 1–5.

[64] John Paul Shen and Mikko H Lipasti. 2013. *Modern processor design: fundamentals of superscalar processors*. Waveland Press.

[65] Xuehuai Shi, Lili Wang, Xinda Liu, Jian Wu, and Zhiwen Shao. 2024. Scene-aware Foveated Neural Radiance Fields. *IEEE Transactions on Visualization and Computer Graphics* (2024).

[66] Rahul Singh, Muhammad Huzaifa, Jeffrey Liu, Anjul Patney, Hashim Sharif, Yifan Zhao, and Sarita Adve. 2023. Power, performance, and image quality tradeoffs in foveated rendering. In *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. IEEE, 205–214.

[67] Hongxin Song, Toco Yuen Ping Chui, Zhangyi Zhong, Ann E Elsner, and Stephen A Burns. 2011. Variation of cone photoreceptor packing density with retinal eccentricity and age. *Investigative ophthalmology & visual science* 52, 10 (2011), 7376–7384.

[68] Hans Strasburger, Ingo Rentschler, and Martin Jüttner. 2011. Peripheral vision and pattern recognition: A review. *Journal of vision* 11, 5 (2011), 13–13.

[69] Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5459–5469.

[70] Q. Sun, F.-C. Huang, J. Kim, L.-Y. Wei, D. Luebke, and A. Kaufman. 2017. Perceptually-Guided Foveation for Light Field Displays. *ACM Transactions on Graphics* 36, 6 (Nov. 2017). https://doi.org/10.1145/3130800.3130807

[71] Nisarg Ujjainkar, Jingwen Leng, and Yuhao Zhu. 2023. ImaGen: A general framework for generating memory-and power-efficient image processing accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.

[72] David R Walton, Rafael Kuffner Dos Anjos, Sebastian Friston, David Swapp, Kaan Akşit, Anthony Steed, and Tobias Ritschel. 2021. Beyond blur: Real-time ventral metamers for foveated rendering. *ACM Transactions on Graphics* 40, 4 (2021), 1–14.

[73] Brian A Wandell. 1995. *Foundations of vision*. sinauer Associates.

[74] Yue Wang, Yan Zhang, Xuanhui Yang, Hui Wang, Dongxu Liu, and Xubo Yang. 2024. Foveated Fluid Animation in Virtual Reality. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. IEEE, 535–545.

[75] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. 2019. Quantization networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 7308–7316.

[76] Jiannan Ye, Anqi Xie, Susmija Jabbireddy, Yunchuan Li, Xubo Yang, and Xiaoxu Meng. 2022. Rectangular mapping-based foveated rendering. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 756–764.

[77] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2024. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 19447–19456.

[78] Yan Zhang, Keyao You, Xiaodan Hu, Hangyu Zhou, Kiyoshi Kiyokawa, and Xubo Yang. 2024. Retinotopic Foveated Rendering. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. IEEE, 903–912.

[79] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2001. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 371–378.