

Real-Time Neural Rasterization for Large Scenes

Jeffrey Yunfan Liu^{1,3†} Yun Chen^{1,2*} Ze Yang^{1,2*} Jingkang Wang^{1,2} Sivabalan Manivasagam^{1,2}
Raquel Urtasun^{1,2}

¹Waabi ²University of Toronto ³University of Waterloo
{jliu, ychen, zyang, jwang, siva, urtasun}@waabi.ai

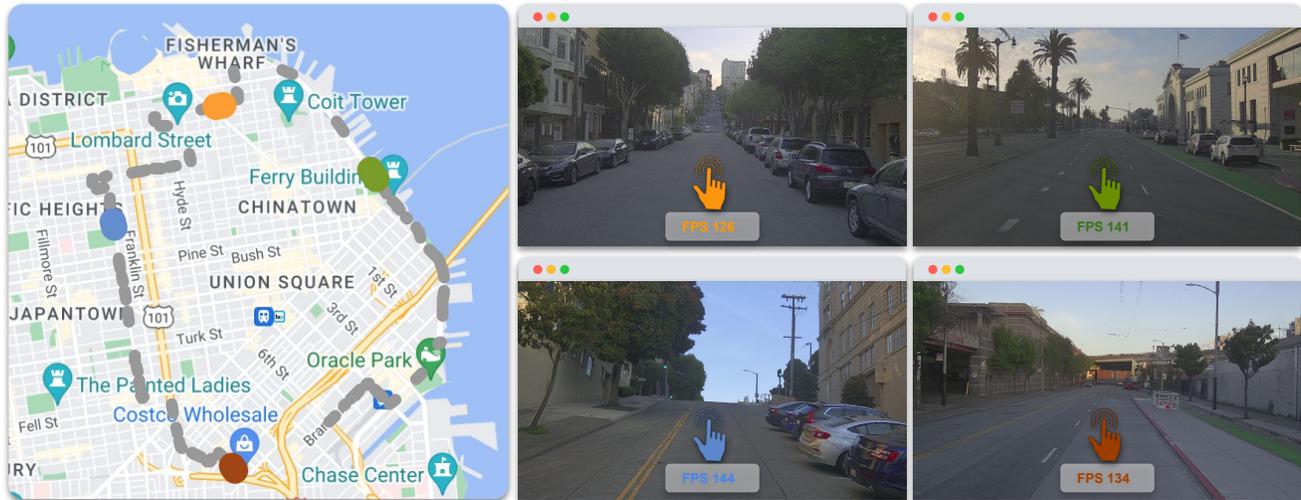


Figure 1. **Neural Scene Rasterization.** Our method renders urban driving scenes (1920×1080) at high quality and >100 FPS by leveraging neural textures and fast rasterization. We reconstruct driving scenes in Bay Area and show the rendering at four streets on the map. Please refer to our project page <https://waabi.ai/neuras/> for more results on driving scenes as well as drone scenes.

Abstract

We propose a new method for realistic real-time novel view synthesis (NVS) of large scenes. Existing neural rendering methods generate realistic results, but primarily work for small scale scenes ($< 50m^2$) and have difficulty at large scale ($> 10000m^2$). Traditional graphics-based rasterization rendering is fast for large scenes but lacks realism and requires expensive manually created assets. Our approach combines the best of both worlds by taking a moderate-quality scaffold mesh as input and learning a neural texture field and shader to model view-dependant effects to enhance realism, while still using the standard graphics pipeline for real-time rendering. Our method outperforms existing neural rendering methods, providing at least $30\times$ faster rendering with comparable or better realism for large self-driving and drone scenes. Our work is the first to enable real-time rendering of large real-world scenes.

1. Introduction

Synthesizing and rendering images for large-scale scenes, such as city blocks, holds significant value in fields such as robotics simulation and virtual reality (VR). In these fields, achieving a high level of realism and speed is of utmost importance. VR requires photorealistic renderings at interactive frame rates for an immersive and seamless user experience. Similarly, robot simulation development requires high-fidelity image quality for real world transfer and high frame rates for evaluation and training at scale, especially for closed-loop sensor simulation [51].

Achieving both speed and realism in large-scale scene synthesis has been a long-standing challenge. Recently, a variety of neural rendering approaches [34, 44, 45] have shown impressive realism results in novel view synthesis (NVS). These methods fall into two primary paradigms: implicit and explicit-based approaches. Implicit-based approaches [34, 6, 37, 65] represent scene geometry and appearance with multi-layer-perceptrons (MLPs) and render novel views by evaluating these MLPs hundreds of thou-

*Indicates equal contribution. †Work done while an intern at Waabi.

sands of times via volume rendering. Explicit-based approaches [44, 45] reconstruct a geometry scaffold (e.g., mesh, point cloud), and then learn image features that are blended and refined with neural networks (NN) to render a novel view. Both implicit and explicit methods require large amounts of NN computation to perform NVS. As a consequence, these approaches have primarily focused on reconstructing objects or small-scale scenes ($< 50m^2$), and typically render at non-interactive frame rates (< 5 FPS).

Several recent methods have enabled rendering at higher frame rates (> 20 FPS) while maintaining realism through several strategies such as sub-dividing the scene [35, 69, 50, 54, 41], caching mechanisms [17, 21, 69, 22], and optimized sampling [60, 36, 25]. However, these approaches still focus primarily on single objects or small scenes. They either do not work on large scenes ($> 10000m^2$) or far-away regions due to memory limitations when learning and rendering such a large volume, or have difficulty maintaining both photorealism and high speed. One area where large scenes are rendered at high speeds is in computer graphics through rasterization-based game engines [23]. However, to render with high quality, these engines typically require accurate specifications of the exact geometry, lighting, and material properties of the scene, along with well-crafted shaders to model physics. This comes at the cost of tedious and time-consuming manual efforts from expert 3D artists and expensive and complex data collection setups for dense capture [15, 19]. While several recent methods have leveraged rasterization-based rendering in NVS [9, 13, 3, 40, 31, 24], they have only been demonstrated on small scenes.

In this paper, we introduce NeuRas, a novel neural rasterization approach that combines rasterization-based graphics and neural texture representations for realistic real-time rendering of large-scale scenes. Given a sequence of sensor data (images and optionally LiDAR), our key idea is to first build a moderate quality geometry mesh of the scene, easily generated with existing 3D reconstruction methods [64, 46, 70, 35]. Subsequently, we perform rasterization with learned feature maps and Multi-Layer Perceptrons (MLPs) shaders to model view-dependent effects. Compared to computationally expensive neural volume rendering, leveraging an approximate mesh enables high-speed rasterization, which scales well for large scenes. Compared to existing explicit-based geometry methods that use large neural networks to perform blending and image feature refinement, we use light-weight MLPs that can be directly exported as fragment shaders in OpenGL for real-time rendering. We also design our neural rasterization method with several enhancements to better handle large scenes. First, inspired by multi-plane and multi-sphere image representations [73, 4], we model far-away regions with multiple neural skyboxes to enable rendering of distant buildings and

sky. Additionally, most NVS methods focus on rendering at target views that are close to the source training views. But for simulation or VR, we need NVS to generalize to novel viewpoints that deviate from the source views. To ensure our approach works well at novel viewpoints, we utilize vector quantization [20, 55] to make neural texture maps more robust and to store them efficiently.

Experiments on large-scale self-driving scenes and drone footage demonstrate that NeuRas achieves the best trade-off between speed and realism compared to existing SoTA. Notably, NeuRas can achieve comparable performance to NeRF-based methods while being at least $30\times$ faster. To the best of our knowledge, NeuRas is the first method of its kind that is capable of realistically rendering large scenes at a resolution of 1920×1080 in real-time, enabling more scalable and realistic rendering for robotics and VR.

2. Related Work

Explicit-based synthesis: Classical 3D reconstruction methods such as structure-from-motion and multi-view stereo [47, 48, 2] have been extensively utilized to reconstruct large-scale 3D models of real-world scenes. However, texture mapping applied during the reconstruction process often produces unsatisfactory appearance due to resolution limitations and lack of view-dependence modelling. To address the challenges of NVS, image-based rendering methods employ the reconstructed geometry as a proxy to re-project target views onto source views, and then blend the source views heuristically or by using a convolutional network [8, 44, 12, 9, 56]. These methods typically require a large amount of memory and may still have visual aberrations due to errors in image blending. Alternatively, point-based methods [26, 3, 40] use per-point feature descriptors and apply convolutional networks to produce images. However, these methods are inefficient in rendering due to large post-processing networks and often exhibit visual artifacts when the camera moves. Similarly, multi-plane images [14, 73, 16, 33] or multi-sphere images [4] are applied for outdoor-scenes rendering in real-time, but they can only be rendered with restricted camera motion. Our work builds on these techniques by leveraging an explicit geometry, but then utilizes UV neural texture maps and fast rasterization to boost speed and realism.

Implicit-based synthesis: In recent years, implicit neural field methods, especially NeRF [34], have become the foundation for many state-of-the-art NVS techniques. NeRF represents the scene as an MLP that is optimized based on camera observations. To accelerate reconstruction time, several methods have been proposed [35, 68, 50, 11, 28], but cannot achieve real-time rendering for high resolution. To accelerate rendering speed, “baking” methods [41, 17, 69, 21, 60] typically pre-compute and store the neural field

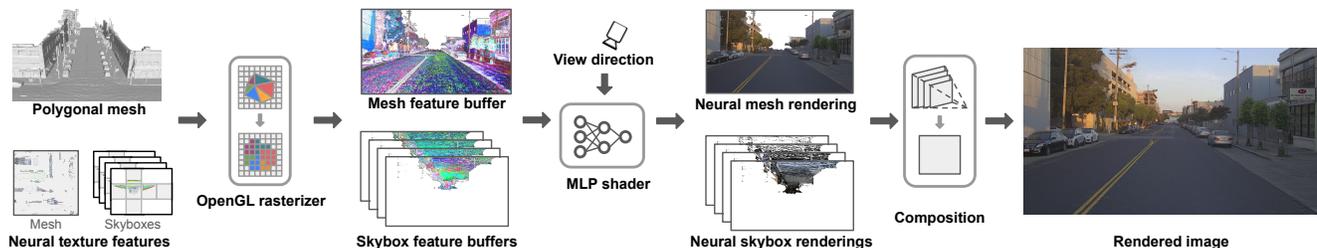


Figure 2. **NeuRas pipeline.** We first rasterize screen-space feature buffers from the scene representation. Then a learned MLP shader takes the rasterized feature buffers and view-direction as inputs and predicts a set of rendering layers. Finally the rendered layers are composited to synthesize the RGB image.

properties for rendering. However, these methods require a large amount of memory and disk storage, limiting their applicability mainly to small objects. Several recent works share similar ideas with our approach of utilizing the graphics pipeline to accelerate neural rendering while maintaining realism. MobileNeRF [13] also uses an explicit mesh with UV feature representations and an MLP baked in GLSL. However, its grid-mesh representation limits its scalability to large scenes due to memory constraints, and its hand-crafted mesh configurations fail to adapt to complex outdoor scenes. Similarly, BakedSDF [67] requires a high-resolution mesh that occupies a significant amount of space even for smaller scenes, and MeRF [42] bakes NeRF to tri-plane images, which reduces memory consumption but limits its scalability to large scenes. In contrast, our approach only requires a moderate-quality mesh as a proxy and uses a quantized UV texture feature, which can be easily extended to large scenes to achieve high realism in real-time.

Large-scene synthesis: Several methods have been proposed to extend NeRF for large-scene synthesis. BlockNeRF [52] and MegaNeRF [54] divide large scenes into multiple spatial cells to increase the model capacity. Recent work GP-NeRF [72] further improves the training speed by using hybrid representations. READ [26] uses LiDAR point cloud descriptors followed by a convolutional network to synthesize autonomous scenes. BungeeNeRF [62] employs a learnable progressive growing model for city-scale scenes. These methods typically struggle with high computational costs, especially for real-time rendering. For interactive visualization, URF [43] bakes mesh while MegaNeRF applies techniques like caching and efficient sampling, but the realism drops significantly. To the best of our knowledge, our approach is the first to demonstrate realistic rendering in real-time for such large-scale scenes.

3. Neural Scene Rasterization

In this section, we describe NeuRas, which aims to perform real-time rendering of large-scale scenes. Given a set of posed images and a moderate-quality reconstructed

mesh, our method generates a scene mesh with neural texture maps and view-dependent fragment shaders. Using the initial mesh, we first generate a UV parameterization to learn neural textures on. We then jointly learn a discrete texture feature codebook and view-dependent lightweight MLPs that can effectively represent scene appearance. Finally, we bake the texture feature codebook and the MLPs into a set of neural texture maps and a fragment shader that can be run in real time with existing graphics pipelines. We now first introduce our approach for representing large-scale scenes (Sec. 3.1), then describe how we render and learn the scene (Sec. 3.2-3.3), and finally how we export our model into real time graphics pipelines (Sec. 3.4).

3.1. Scene Representation

In this paper our focus is on rendering large-scale outdoor scenes. In order to handle potentially infinite depth ranges (*e.g.*, sky, vegetation, mountain, *etc.*) as well as nearby regions, we utilize a hybrid approach. The entire 3D scene is partitioned into two regions: an inner cuboid region (*foreground*) modelled by a polygonal mesh textured with neural features, and an outer cuboid region (*background*) modelled by neural skyboxes. Such a hybrid scene representation allows us to model fine-grained details in both close-by regions and far-away regions, and enables rendering with a remarkable degree of camera movement.

Foreground representation: For the *foreground* region, we leverage an explicit geometry mesh scaffold to learn and render neural textures. Our approach can be applied on various sources of mesh, for example we can leverage existing neural reconstruction methods [64, 35] or SfM [47] (see supp. materials for details). Initially, the reconstructed mesh could contain over tens of millions of triangle faces, which represents the geometry well, but may have self-intersections and duplicate vertices. We thus preprocess it to reduce computational cost and improve UV mapping quality. We first cluster nearby vertices together and perform quadric mesh decimation [18] to simplify the mesh while preserving essential structure, and then perform face culling to remove non-visible triangle faces (w.r.t. source

camera views). Finally we utilize a UV map generation tool [39] to unfold the mesh to obtain the UV mappings for each of its vertices. The resulting triangle mesh $\mathcal{M} = \{\mathbf{v}, \mathbf{t}, \mathbf{f}\}$ consists of vertex positions $\mathbf{v} \in \mathbb{R}^{N \times 3}$, vertex UV coordinates $\mathbf{t} \in \mathbb{R}^{N \times 2}$, and a set of triangle faces \mathbf{f} . Based on the generated UV mapping, we initialize a learnable UV feature map $\mathbf{T} \in \mathbb{R}^{V \times U \times D}$ to represent the scene appearance covered by the mesh. Using neural features instead of a color texture map enables modelling view-dependent effects during rendering (see Sec. 3.2).

Background representation: It is challenging to model the far-away *background* regions with polygonal mesh because of the complexity and scale of that region.

As an alternative, we draw inspiration from the concept of multi-plane images [73, 16, 33] and multi-sphere images [4] to represent the *background* region using neural skyboxes.

The neural skyboxes represents the scene as a set of cuboid layers, and each layer $\mathcal{S}_i = \{\mathbf{S}_i^1, \dots, \mathbf{S}_i^6\}$ contains 6 individual feature maps that each represents one plane of the cuboid. The feature maps represent both geometry and view-dependent appearance of the scene. Our neural skyboxes can represent a wide range of depths and can be integrated in existing graphics pipelines [23], enabling efficient rendering.

3.2. Rendering Large Scenes

Fig. 2 shows an overview of our rendering pipeline. Our NeuRas framework is inspired by the deferred shading pipeline from real-time graphics [53]. We first rasterize the foreground mesh and neural skyboxes with neural texture maps to the desired view point, producing a set of image feature buffers. The feature buffers are then processed with MLPs to produce a set of rendering layers, which are composited to synthesize the final RGB image. We now describe this rendering process in more detail.

Foreground rendering: Given the camera pose and intrinsics, we first rasterize the mesh into screen space, obtaining a UV coordinate (u, v) for each pixel (x, y) on the screen. We then sample the UV feature map $\mathbf{T} \in \mathbb{R}^{V \times U \times D}$ using the rasterized UV coordinates and obtain a feature buffer $\mathbf{F}_0 \in \mathbb{R}^{H \times W \times D}$:

$$\mathbf{F}_0(x, y) = \text{BilinearSample}(u, v, \mathbf{T}), \quad (1)$$

where $V \times U$ is the UV feature map resolution, $H \times W$ is the rendering resolution, and D is the feature dimension. In addition to a feature buffer, the rasterization also generates a opacity mask $\mathbf{O}_0 \in \mathbb{R}^{H \times W}$ to indicate if a pixel is covered by the polygonal mesh. To render the RGB image,

we concatenate the rendered feature with the view direction $\mathbf{d}(x, y)$ and pass through a learnable MLPs shader f_{θ^T} :

$$\mathbf{I}_0(x, y) = f_{\theta^T}(\mathbf{F}_0(x, y), \mathbf{d}(x, y)), \quad (2)$$

where θ^T is the MLP parameters, $\mathbf{I}_0(x, y)$ is the rendered RGB color for pixel (x, y) .

Background rendering: To render the neural skybox feature layers $\{\mathcal{S}_i\}_{i=1}^L$ representing distant *background* regions, we project camera ray shooting each pixel (x, y) and compute its intersection points $\{\mathbf{p}_i\}_{i=1}^L$ with layers 1 to L , from near to far. Next, we sample the features $\{\mathbf{f}_i\}_{i=1}^L$ corresponding to the intersection points on the neural skybox feature map at each layer, generating a set of feature buffers $\mathbf{F}_i \in \mathbb{R}^{H \times W \times D}$, where $i = 1, \dots, L$. This step can be efficiently performed with the OpenGL rasterizer. We then use a learnable MLPs shader f_{θ^S} to process the feature buffers and outputs the opacity map $\mathbf{O}_i \in \mathbb{R}^{H \times W}$ and RGB color map $\mathbf{I}_i \in \mathbb{R}^{H \times W \times 3}$ for each layer:

$$\mathbf{O}_i(x, y), \mathbf{I}_i(x, y) = f_{\theta^S}(\mathbf{F}_i(x, y), \mathbf{d}(x, y)), \quad (3)$$

where θ^S represents the parameters of the MLPs. The MLPs shader first processes the input feature $\mathbf{F}_i(x, y)$, and outputs opacity $\mathbf{O}_i(x, y)$ and an intermediate feature vector. The feature vector is then concatenated with $\mathbf{d}(x, y)$, the view direction of camera ray, and passed to the last layers that output the view-dependent color $\mathbf{I}_i(x, y)$.

Compositing foreground and background: To synthesize the final RGB image, we composite the rendered layers from the foreground mesh $\{\mathbf{I}_0(x, y), \mathbf{O}_0(x, y)\}$ and neural skyboxes $\{\mathbf{I}_i(x, y), \mathbf{O}_i(x, y)\}_{i=1}^L$ by repeatedly compositing the RGB and opacity layers, from near to far:

$$\mathbf{I}(x, y) = \sum_{i=0}^L \mathbf{I}_i(x, y) \cdot \mathbf{O}_i(x, y) \cdot \prod_{j=0}^{i-1} (1 - \mathbf{O}_j(x, y)). \quad (4)$$

The term $\mathbf{I}_i(x, y) \cdot \mathbf{O}_i(x, y)$ represents the color contribution of the current layer i , and $\prod_{j=1}^{i-1} (1 - \mathbf{O}_j(x, y))$ denotes the fraction of the color that will remain after attenuation through the layers in front. This compositing process ensures that the RGB values are correctly blended.

Quantized texture representation: To encourage the sharing of latent features in visually similar regions such as roads and sky, we apply vector quantization (VQ) to regularize the neural texture maps. This allows these features to be supervised from a large range of view directions which improves view-point extrapolation performance. Furthermore, it also significantly compacts the feature representations, reducing the offline storage space. We follow [55]

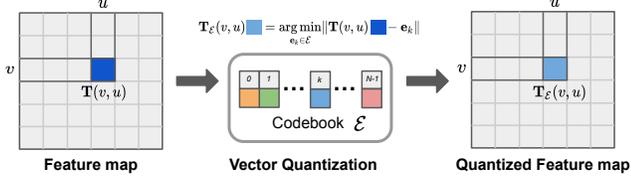


Figure 3. **Quantized feature representation.** For each entry in the feature map, we map it to the closest learnable latent code \mathbf{e}_k in the codebook \mathcal{E} .

and maintain two codebooks $\mathcal{E}^{\mathbf{T}}$ and $\mathcal{E}^{\mathbf{S}}$, each consists of K learnable latent code $\mathbf{e}_k \in \mathbb{R}^D$, with $k = 1, \dots, K$. In the forward pass, we quantize the UV feature map $\mathbf{T}_{\mathcal{E}} \in \mathbb{R}^{V \times U \times D}$ and neural skyboxes feature maps $\mathbf{S}_{\mathcal{E}} \in \mathbb{R}^{L \times V \times U \times D}$ by mapping each feature to its closest latent code in the codebook:

$$\begin{aligned} \mathbf{T}_{\mathcal{E}}(v, u) &= \arg \min_{\mathbf{e}_k \in \mathcal{E}^{\mathbf{T}}} \|\mathbf{T}(v, u) - \mathbf{e}_k\|, \\ \mathbf{S}_{\mathcal{E}}(l, v, u) &= \arg \min_{\mathbf{e}_k \in \mathcal{E}^{\mathbf{S}}} \|\mathbf{S}(l, v, u) - \mathbf{e}_k\|, \end{aligned} \quad (5)$$

where v, u are the spatial coordinates of the feature map, and l is the layer index of the neural skyboxes. Fig. 3 shows the feature map quantization process. We use the quantized features to compute the synthesized image in Eqn. 4.

3.3. Learning NeuRas

We jointly optimize the feature map \mathbf{T} , \mathbf{S} , the codebook $\mathcal{E}^{\mathbf{T}}$, $\mathcal{E}^{\mathbf{S}}$, as well as the parameters $\theta^{\mathbf{T}}$, $\theta^{\mathbf{S}}$ of the MLP shaders by minimizing the photometric loss and perceptual loss between our rendered images and camera observations, as well as the VQ regularizer. Our full objective is:

$$\mathcal{L} = \mathcal{L}_{\text{rgb}} + \lambda_{\text{perc}} \mathcal{L}_{\text{perc}} + \lambda_{\text{vq}} \mathcal{L}_{\text{vq}}. \quad (6)$$

In the following, we discuss each loss term in more detail.

Photometric loss: \mathcal{L}_{rgb} measures the ℓ_2 distance between the rendered and the observed images, the loss is defined as:

$$\mathcal{L}_{\text{rgb}} = \|\mathbf{I} - \hat{\mathbf{I}}\|_2, \quad (7)$$

where \mathbf{I} is the rendered image from Eqn. 4 and $\hat{\mathbf{I}}$ is the corresponding observed camera image.

Perceptual loss: We use an additional perceptual loss [71, 57] to enhance the rendered image quality. This loss measures the ‘‘perceptual similarity’’ that is more consistent with human visual perception:

$$\mathcal{L}_{\text{perc}} = \sum_{i=1}^M \frac{1}{N_i} \left\| V_i(\mathbf{I}) - V_i(\hat{\mathbf{I}}) \right\|_1, \quad (8)$$

where V_i denotes the i -th layer with N_i elements of the pre-trained VGG Network [49].

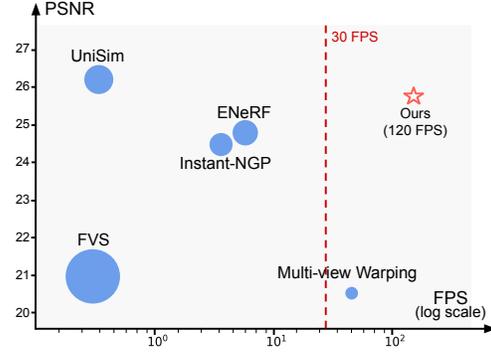


Figure 4. **Rendering realism vs efficiency.** Our method achieves the best tradeoff between realism and speed. The size of the markers indicates the memory consumption required for rendering.

VQ loss: To update the codebook \mathcal{E} , we follow [55] and define the VQ loss term as:

$$\begin{aligned} \mathcal{L}_{\text{vq}} &= \|\text{sg}[\mathbf{T}] - \mathbf{T}_{\mathcal{E}}\|_2^2 + \|\text{sg}[\mathbf{S}] - \mathbf{S}_{\mathcal{E}}\|_2^2 \\ &+ \beta \|\text{sg}[\mathbf{T}_{\mathcal{E}}] - \mathbf{T}\|_2^2 + \beta \|\text{sg}[\mathbf{S}_{\mathcal{E}}] - \mathbf{S}\|_2^2, \end{aligned} \quad (9)$$

where $\text{sg}[\cdot]$ denotes the stop-gradient operator that behaves as the identity map at forward pass and has zero partial derivatives at backward pass. The first two terms form the alignment loss and encourage the codebook latents to follow the feature maps. The last two terms form the commitment loss which stabilizes training by discouraging the features from learning much faster than the codebook. It is noted that the quantization step in Eqn. 5 is non-differentiable. We approximate the gradient of the feature maps \mathbf{T}, \mathbf{S} using the straight-through estimator [7], which simply passes the gradient from the quantized feature to the original feature unaltered during back-propagation.

3.4. Real Time Rendering

To enable NeuRas to render in real time, we convert our scene representations and MLPs to be compatible with the graphics rendering pipeline. The mesh, skyboxes, and texture representations are all directly compatible with the OpenGL, while the learned MLPs $f_{\theta^{\mathbf{T}}}$ and $f_{\theta^{\mathbf{S}}}$ are converted to fragment shaders in OpenGL. During each rendering pass, the triangle mesh \mathcal{M} and the skyboxes $\{\mathcal{S}_i\}_{i=1}^L$ are rasterized to the screen as a set of fragments, and each fragment is associated with a feature vector that is bilinearly sampled from the neural texture maps. The fragment shader then maps each fragment’s features to RGB color and opacity. To ensure correct alpha compositing, the scene mesh and cuboids are sorted depth-wise and rendered from back to front, following the procedure outlined in Eqn. 4.

4. Experiments

In this section, we introduce our experimental setting, and then compare our approach with state-of-the-art NVS



Figure 5. **Qualitative results on driving scenes.** Compared to existing novel view synthesis approaches, NeuRas produces competitive realism and achieves real-time rendering (> 100 FPS).

Methods	Interpolation				Lane Shift		Resources	
	MSE↓	PSNR↑	SSIM↑	LPIPS↓	FID↓ @2m	FID↓ @3m	Memory (MB) ↓	FPS↑
Instant-NGP [35]	0.0035	24.76	0.79 ●	0.40	134.20	135.75	7491	3.2
ENeRF [28]	0.0034 ●	24.92 ●	0.77	0.34	190.96	243.93	7285 ●	5.1 ●
UniSim [64]	0.0030 ●	26.28 ●	0.78 ●	0.26 ●	92.78 ●	100.63 ●	7623	1.4
Multi-View Warping [10]	0.0095	20.55	0.64	0.39	164.09	177.19	1441 ●	50 ●
FVS [44]	0.0089	20.87	0.71	0.29 ●	116.36 ●	122.87 ●	15243	0.3
Ours	0.0030 ●	25.45 ●	0.75 ●	0.31 ●	105.64 ●	111.63 ●	4538 ●	120 ●

Table 1. **State-of-the-art comparison on PandaSet.** Our method synthesizes novel views (1920×1080) in real time with high visual quality on urban driving scenes. We mark the methods with best performances using gold ●, silver ●, and bronze ● medals.

methods on large-scale driving scenes and drone footages. We demonstrate that our neural rendering system achieves the best balance between photorealism and rendering efficiency. We then ablate our design choices, showing the value of the neural shader and vector quantization for improved realism and extrapolation robustness. Finally, we show NeuRas can speed up various NeRF approaches in a plug-and-play fashion by leveraging their extracted meshes for real-time NVS of large scenes.

4.1. Experimental Setup

Datasets: We conduct experiments primarily on two public datasets with large-scale scenes: PandaSet [63] and BlendedMVS [66]. PandaSet is a real-world driving dataset that contains 103 urban scenes captured in San Francisco, each with a duration of 8 seconds (80 frames, sampled at 10Hz) and a coverage of around $300 \times 80m^2$ meters. At each frame, a front-facing camera image (1920×1080) and 360 degree point cloud are provided. We select 9 scenes

on Pandaset that have few dynamic actors for NVS evaluation. Since our focus is on static scenes, the dynamic actor regions are masked out during evaluation by projecting the 3D bounding boxes into images. The BlendedMVS dataset offers a collection of large-scale outdoor scenes captured by a drone and also provides reconstructed meshes generated from a 3D reconstruction pipeline [1]. We select 5 large scenes for evaluation which are diverse and range in size from $200 \times 200m^2$ to over $500 \times 500m^2$.

Baselines: We compare our approach with two types of novel view synthesis (NVS) methods: (1) *Implicit-based neural fields*: Instant-NGP [35], ENeRF [29], UniSim [64]. Instant-NGP introduces a multi-resolution hashing grid and tiny MLP for fast training and inference. UniSim and ENeRF leverage the LiDAR or depth information for efficient ray generation (sparse grids) and efficient sampling along each ray. (2) *Explicit-based approaches*: multi-view warping [10] uses the mesh to project nearby source images to

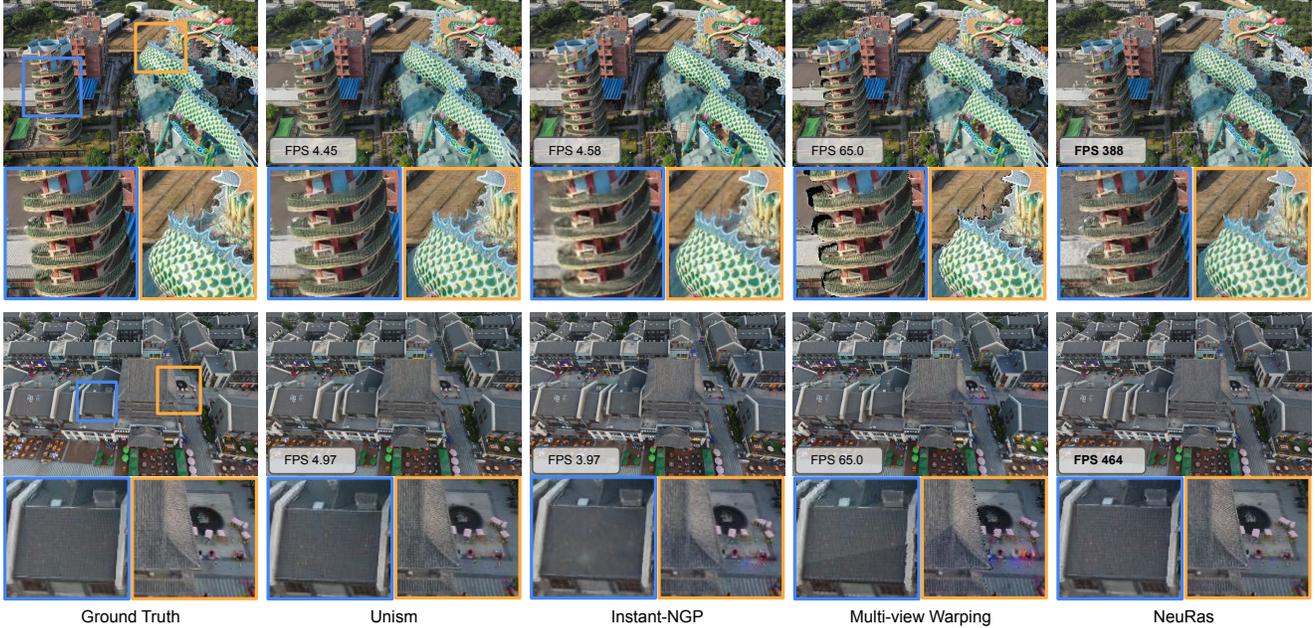


Figure 6. **Qualitative results on drone scenes.** Compared to existing novel view synthesis approaches, NeuRas produces competitive realism results and achieves real-time rendering (> 400 FPS) on drone scenes.

Methods	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	FPS \uparrow
Instant-NGP [35]	24.40	0.780	0.241	8.7
UniSim [64]	24.37	0.792	0.204	4.9
Multi-View Warping [10]	21.11	0.738	0.248	65
Ours	24.19	0.790	0.176	462

Table 2. **State-of-the-art comparison on BlendedMVS.**

target views, and FVS [44] further uses neural networks to blend the source images.

Implementation details: In our implementation, we utilized a UV feature size of 8192×8192 with 12 channels for the neural texture component. Additionally, we employed 6 skyboxes placed at a distance ranging from $150m$ to $2.4km$ from the inner cuboid center. The MLP consists of 3 layers, with each layer comprising of 32 hidden units. Both codebooks have a size of 1024. We trained the model using the Adam optimizer with a learning rate of 0.01 for $20K$ iterations. We use a single machine equipped with an A5000 GPU for all reported runtimes and memory usage benchmarking, including the baselines. Further implementation details are provided in the supp. materials.

4.2. Fast Rendering on Large-scale Scenes

Driving scenes on PandaSet: For self-driving, we need to simulate the camera images at significantly different viewpoints (*interpolation* and *extrapolation*). We evaluate the *interpolation* setting following [27]: sub-sampling the sensor data by two, training on every other frame and testing on the remaining frames. We report PSNR, SSIM [58],

and LPIPS [71]. Moreover, we evaluate a more challenging *extrapolation* setting (*Lane Shift*) following [64] by simulating a new trajectory shifted laterally to the left or right by 2 or 3 meters. We report FID [38] since there is no ground truth for comparison. We use the neural meshes reconstructed by UniSim [64] for the experiments.

As shown in Fig. 4, our approach strikes the best trade-off between rendering quality and rendering speed. Table 1 shows all the metrics, specifically, our method only sacrifices 0.8 PSNR compared to the best approach while maintaining > 100 FPS ($80\times$ faster than UniSim, $40\times$ faster than Instant-NGP). In contrast, implicit-based neural fields obtain slightly better (UniSim) or worse (InstantNGP, ENerF) PSNR with much lower inference speed. Compared to geometry-based approaches such as multi-view warping and FVS, NeuRas achieves better realism and faster rendering speed. Furthermore, the qualitative results presented in Fig. 5 demonstrate that our method exhibits comparable or superior visual realism when compared to the baselines.

Drone scenes on BlendedMVS: We leverage the provided reconstructed meshes in BlendedMVS as our geometry scaffold and evaluate interpolation setting (*i.e.*, random 50–50 train-validation split). As shown in Table 2, NeuRas strikes a good balance between rendering quality and speed. Fig. 6 shows qualitative comparisons with state-of-the-art NVS methods. Except for view-warping, which has visible artifacts, it is difficult to discern differences between our method and slower neural rendering methods.

Methods	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
No MLP	24.48	0.664	0.374
MLP-shader w/o viewdir	25.01	0.728	0.316
MLP shader	25.34	0.738	0.308

Table 3. **Ablation on MLP shader.** Metrics are reported on the $\log-53$ in PandaSet.

Methods	Interpolation		Lane Shift		Storage \downarrow (MB)
	PSNR \uparrow	LPIPS \downarrow	FID \downarrow @2m	FID \downarrow @3m	
w/o VQ	25.46	0.317	81.5	98.3	4644
w/ VQ	25.34	0.308	78.2	95.3	394

Table 4. **Ablation on vector quantization.** Vector quantization improves extrapolation and reduces storage, with minimal impact on realism. Metrics are reported on $\log-53$ in PandaSet.



Figure 7. **Qualitative comparison between using mesh with 50K and 500K vertices**

Method Ablation: We ablate our design choices for NeuRas on neural shader and vector quantization (VQ). As shown in Table 3, using an MLP-backed fragment shader can improve realism in generating view-dependent results. It can also help compensate for artifacts in the geometry. Please refer to supp. for visual comparison. In Table 4, we show vector quantization significantly reduce the disk storage while maintaining similar realism. Besides, adding VQ helps regularize the neural texture maps thus resulting in better perceptual quality especially in extrapolation results (indicated by smaller FID in the lane-shift setting).

Mesh Ablation: We also demonstrate that NeuRas can perform well with coarser low poly-count meshes in Fig 7. Given the geometry mesh extracted with UniSim [64], we perform triangle decimation [18] to achieve desired vertex counts, and then learn the texture maps with NeuRas. Our approach achieves similar visual quality despite the low resolution mesh, demonstrating the value of the neural textures.

Comparison against real-time NeRFs: When comparing our method with real-time rendering approaches such as SNeRG [21], PlenOctrees [69], and MobileNeRF [13], we encountered difficulties in applying these methods to large scenes and achieving satisfactory results. These challenges included low-resolution representation, out-of-memory errors, limited model capacity, high training cost, and poor geometry. We therefore compare against these methods on

Methods	Chair			Lego			FPS \uparrow
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	
NeRF [34]	33.00	0.967	0.046	32.54	0.961	0.050	0.02
Mip-NeRF [5]	37.14	0.988	0.011	35.74	0.984	0.013	0.02
TensorRF [11]	35.76	0.985	0.022	36.46	0.983	0.018	1.5
NSVF [30]	33.19	0.968	0.043	32.29	0.973	0.029	0.84
SNeRG [21]	33.24	0.975	0.025	33.82	0.973	0.022	176
PlenOctree [69]	33.19	0.970	0.039	32.12	0.965	0.046	270
MobileNeRF [13]	34.02	0.978	0.025	34.18	0.975	0.025	720
Ours	33.15	0.975	0.036	30.66	0.951	0.061	461

Table 5. **Comparison with real-time NeRFs on synthetic dataset.** Despite being designed for large-scale scenes, our method still achieves comparable performance when rendering small objects.

Methods	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	FPS \uparrow
InstantNGP [35]	23.55	0.75	0.23	8.2
Ours + InstantNGP	20.26	0.52	0.31	455
UniSim [64]	23.30	0.76	0.21	4.7
Ours + UniSim	21.19	0.65	0.23	336

Table 6. **Speed up neural radiance fields.** NeuRas is able to speed up popular NeRFs even if the geometry is not high-quality. The metrics are reported on Church in BlendedMVS.

object-level scenes in Table 5. We use VoxSurf [61] to extract the geometries for NeuRas. Please see supp. for more details. While we focus on the real-time rendering for large scenes, our approach has reasonable performance on small-objects from NeRF synthetic dataset [34].

4.3. Speeding-up NeRFs

We highlight that NeuRas can speed up popular NeRF approaches. We consider two representative approaches: Instant-NGP (camera supervision, density geometry) and UniSim (camera + depth supervision, SDF geometry). We use marching cubes [32] to extract the geometry for a selected scene in BlendedMVS and then adopt NeuRas for real-time rendering. As shown in Table 6 and Fig 8, our approach dramatically speeds up rendering performance while maintaining reasonable photorealism. Please see supp. materials for visual comparison and more analysis.

4.4. Limitations

Our method has several limitations, including the use of opaque meshes, which poses challenges in accurately modeling semi-transparent components such as fog and water. Additionally, while our neural shader design improves the robustness of the model w.r.t. geometry quality, rendering performance may still suffer when dealing with meshes with severe artifacts in Fig. 9. Our method cannot fix severe mesh artifacts and incorrect boundaries, which requires a fully differentiable rasterization pipeline. Additionally, our approach has difficulty rendering completely unseen regions that are far from the training views. Scene completion

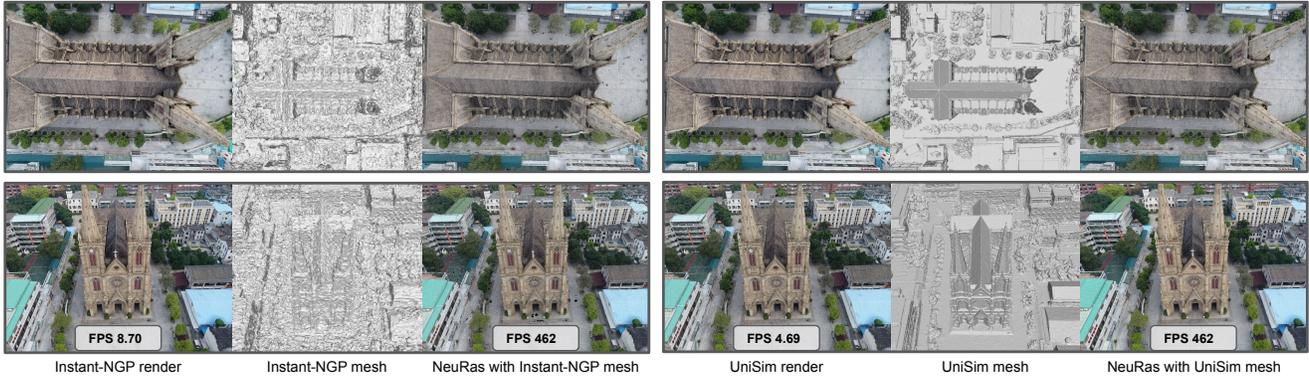


Figure 8. **Speed up neural radiance fields with NeuRas.** We use Marching Cubes [32] to extract the geometry from the trained radiance fields model and then adopt NeuRas for real-time rendering. For each example, we show radiance field rendering in the left, extracted mesh in the middle, and NeuRas rendering in the right. NeuRas can significantly speed up the rendering speed while maintaining a similar rendering photorealism even with poor geometry scaffold (e.g. Instant-NGP [35]).



Figure 9. **Limitations of NeuRas.** From left to right: Difficulties in modelling intricate structures such as power lines in the sky, artifacts at the boundary of foreground mesh and skybox due to missing geometry at the top of building, and lack of details at high zoom levels.

may help address this. Moreover, the current implementation uses only one UV texture level, which can cause aliasing or blurriness when scaling extensively, such as when moving very close to or far away from an object. Using a multi-level texture representation such as mipmaps [59] could mitigate these issues. More analysis is available in the supp. materials.

5. Conclusion

In this paper, we present NeuRas, a novel approach for realistic real-time novel view synthesis of large scenes. Our approach combines the strengths of neural rendering and traditional graphics to achieve the best trade-off between realism and efficiency. NeuRas utilizes a scaffold mesh as input and incorporates a neural texture field to model view-dependent effects, which can then be exported and rendered in real-time with standard rasterization engines. NeuRas can render urban driving scenes at 1920×1080 resolution at over 100 FPS while delivering comparable realism to existing neural rendering approaches. We hope NeuRas can open up possibilities for scalable and immersive exper-

iences for self-driving simulation and VR applications.

Acknowledgements: We thank Ioan-Andrei Barsan for profound discussion and constructive feedback. We thank the Waabi team for their valuable assistance and support.

References

- [1] Altizure: Mapping the world in 3d. <https://www.altizure.com>.
- [2] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 2011.
- [3] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *ECCV*, 2020.
- [4] Benjamin Attal, Selena Ling, Aaron Gokaslan, Christian Richardt, and James Tompkin. Matryodshka: Real-time 6dof video view synthesis using multi-sphere images. In *ECCV*, pages 441–459. Springer, 2020.
- [5] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021.

- [6] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022.
- [7] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.
- [8] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. *Computer graphics and interactive techniques*, 2001.
- [9] Ang Cao, Chris Rockwell, and Justin Johnson. Fwd: Real-time novel view synthesis with forward warping and depth. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15713–15724, 2022.
- [10] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG)*, 32(3):1–12, 2013.
- [11] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *Computer Vision—ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, pages 333–350. Springer, 2022.
- [12] Yun Chen, Frieda Rong, Shivam Duggal, Shenlong Wang, Xinchen Yan, Sivabalan Manivasagam, Shangjie Xue, Ersin Yumer, and Raquel Urtasun. Geosim: Realistic video simulation via geometry-aware composition for self-driving. *CVPR*, 2021.
- [13] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv*, 2022.
- [14] Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H. Kim, and Jan Kautz. Extreme view synthesis. *ICCV*, 2019.
- [15] Yves Egels and Michel Kasser. *Digital photogrammetry*. CRC Press, 2001.
- [16] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *CVPR*, pages 2367–2376, 2019.
- [17] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *ICCV*, 2021.
- [18] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.
- [19] Kaiwen Guo, Peter Lincoln, Philip Davidson, Jay Busch, Xueming Yu, Matt Whalen, Geoff Harvey, Sergio Orts-Escolano, Rohit Pandey, Jason Dourgarian, et al. The relightables: Volumetric performance capture of humans with realistic relighting. *ACM Transactions on Graphics (ToG)*, 38(6):1–19, 2019.
- [20] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.
- [21] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *arXiv*, 2021.
- [22] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. Efficientnerf efficient neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12902–12911, 2022.
- [23] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 2013.
- [24] Petr Kellnhofer, Lars C Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4287–4297, 2021.
- [25] Naruya Kondo, Yuya Ikeda, Andrea Tagliasacchi, Yutaka Matsuo, Yoichi Ochiai, and Shixiang Shane Gu. Vaxnerf: Revisiting the classic for voxel-accelerated neural radiance field. *arXiv preprint arXiv:2111.13112*, 2021.
- [26] Zhuopeng Li, Lu Li, Zeyu Ma, Ping Zhang, Junbo Chen, and Jianke Zhu. Read: Large-scale neural scene rendering for autonomous driving. *arXiv*, 2022.
- [27] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *PAMI*, 2022.
- [28] Haotong Lin, Sida Peng, Zhen Xu, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Efficient neural radiance fields for interactive free-viewpoint video. In *SIGGRAPH Asia Conference Proceedings*, 2022.
- [29] Haotong Lin, Sida Peng, Zhen Xu, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Efficient neural radiance fields for interactive free-viewpoint video. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022.
- [30] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *arXiv*, 2020.
- [31] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *ACM Transactions on Graphics (ToG)*, 40(4):1–13, 2021.
- [32] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.
- [33] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.
- [34] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *ECCV*, 2020.
- [35] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. 2022.
- [36] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H Mueller, Chakravarty R Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. Donerf: Towards real-time rendering of compact neural radiance fields using depth oracle networks. In *Computer Graphics Forum*, 2021.

- [37] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020.
- [38] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. On aliased resizing and surprising subtleties in gan evaluation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11410–11420, 2022.
- [39] John C. Perry. xatlas: A c++11 library for texture atlas generation. <https://github.com/jpcy/xatlas>, 2018. GitHub repository.
- [40] Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lepitsky, and Evgeny Burnaev. Npbg++: Accelerating neural point-based graphics. In *CVPR*, 2022.
- [41] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. *ICCV*, 2021.
- [42] Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P. Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T. Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *arXiv preprint arXiv: Arxiv-2302.12249*, 2023.
- [43] Konstantinos Rematas, An Liu, Pratul P. Srinivasan, J. Barron, A. Tagliasacchi, T. Funkhouser, and V. Ferrari. Urban radiance fields. *Computer Vision And Pattern Recognition*, 2021.
- [44] Gernot Riegler and Vladlen Koltun. Free view synthesis. *ECCV*, 2020.
- [45] Gernot Riegler and Vladlen Koltun. Stable view synthesis. *CVPR*, 2021.
- [46] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. *CVPR*, 2016.
- [47] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [48] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.
- [49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.
- [50] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. *CVPR*, 2022.
- [51] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wilmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems*, 34:251–266, 2021.
- [52] Matthew Tancik, Vincent Casser, Xinchun Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretschmar. Block-nerf: Scalable large scene neural view synthesis. In *CVPR*, 2022.
- [53] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *Ac Transactions on Graphics (TOG)*, 38(4):1–12, 2019.
- [54] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *CVPR*, 2022.
- [55] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *NeurIPS*, 30, 2017.
- [56] JingKang Wang, Sivabalan Manivasagam, Yun Chen, Ze Yang, Ioan Andrei Bârsan, Anqi Joyce Yang, Wei-Chiu Ma, and Raquel Urtasun. CADSim: Robust and scalable in-the-wild 3d reconstruction for controllable sensor simulation. In *6th Annual Conference on Robot Learning*, 2022.
- [57] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *CVPR*, 2018.
- [58] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *TIP*, 2004.
- [59] Lance Williams. Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 1–11, 1983.
- [60] Liwen Wu, Jae Yong Lee, Anand Bhattad, Yu-Xiong Wang, and David Forsyth. Diver: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16200–16209, 2022.
- [61] Tong Wu, Jiaqi Wang, Xingang Pan, Xudong XU, Christian Theobalt, Ziwei Liu, and Dahua Lin. Voxurf: Voxel-based efficient and accurate neural surface reconstruction. In *The Eleventh International Conference on Learning Representations*, 2023.
- [62] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *Computer Vision—ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, pages 106–122. Springer, 2022.
- [63] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *ITSC*, 2021.
- [64] Ze Yang, Yun Chen, JingKang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. *arXiv*, 2023.
- [65] Ze Yang, Sivabalan Manivasagam, Yun Chen, JingKang Wang, Rui Hu, and Raquel Urtasun. Reconstructing objects in-the-wild for realistic sensor simulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.
- [66] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks. *CVPR*, 2020.

- [67] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. Baked sdf: Meshing neural sdfs for real-time view synthesis. *arXiv preprint arXiv: Arxiv-2302.14859*, 2023.
- [68] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *CVPR*, 2022.
- [69] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenotrees for real-time rendering of neural radiance fields. *ICCV*, 2021.
- [70] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *NeurIPS*, 2022.
- [71] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CVPR*, 2018.
- [72] Yuqi Zhang, Guanying Chen, and Shuguang Cui. Efficient large-scale scene representation with a hybrid of high-resolution grid and plane features. *arXiv preprint arXiv: Arxiv-2303.03003*, 2023.
- [73] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *SIGGRAPH*, 2018.