Real-Time Neural Rasterization for Large Scenes
Supplementary Material

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

Abstract

This supplementary material contains comprehensive details regarding our proposed NeuRas method and the baselines, experiment setting, additional results, and limitations of our method. We first describe the implementation details of our approach in Sec. 1. Then we provide the additional implementation details for the baseline methods, along with the modifications made to suit our experimental settings in Sec. 2. In Sec. 3, we provide more details for the experimental settings. In Sec. 4 we show additional qualitative results and ablations. Finally, we analyze the limitations and future works of our method in Sec. 5. Please refer to our project page https://waabi.ai/neuras/ for video results on driving scenes as well as drone scenes.

1. NeuRas Implementation Details

1.1. Scene Representation

Foreground mesh reconstruction: NeuRas takes a geometry scaffold foreground mesh as input. For scene mesh reconstruction on PandaSet [16], we train UniSim [17] on each driving scene using both front-facing camera images and the 360° spinning LiDAR point clouds. Unisim [17] leverages the LiDAR information to perform efficient ray-marching and adopts a signed distance function (SDF) representation to model the scene geometry. The SDF representation is regularized with an Eikonal loss [3], which helps the model learn a smooth zero level set. Finally, we use Marching Cubes [6] to extract the mesh from the learned SDF volume .

Foreground mesh simplification: The reconstructed mesh obtained from a neural representation could contain over tens of millions of triangle faces to model the detailed geometry, as it is not optimized to represent the mesh cleanly and compactly. In order to reduce computational cost and enhance the UV texture mapping quality, we perform quadric mesh decimation [2] to simplify the mesh while preserving essential structure. Subsequently, we perform face culling to eliminate non-visible triangle faces. The face culling is implemented in OpenGL by rasterizing `gl_PrimitiveID` from each of the source camera poses.

Neural texture map creation: To generate a neural texture map for the foreground mesh, we leverage a UV map generation tool [11] to unfold the mesh and derive the UV mapping for each mesh vertex. Based on the generated UV mapping, we initiate a learnable UV feature map $\mathbf{T} \in \mathbb{R}^{V \times U \times D}$ to represent the scene appearance covered by the mesh. By using a neural texture map instead of an RGB texture map, we can effectively model view-dependent effects during the rendering process.

Neural skyboxes: To learn far-away regions for the large scene, we represent distant regions with neural skyboxes. We utilize 6 neural skyboxes, with the same dimensions as the foreground bounding box (*e.g.* $300 \times 80 \times 100$ m for PandaSet)

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

For each neural skybox, we initialize 6 learnable neural texture maps for each face of the box to represent the opacity and appearance of the skybox, and each neural texture map $\mathbf{S} \in \mathbb{R}^{V \times U \times D}$ corresponds to one face of the skybox.

Training time: We train the model on an single A5000 GPU. It takes 3 hours to train the model. The training memory is approximately 22 gigabytes.

1.2. Interactive Rendering

NeuRas can achieve interactive, real-time rendering thanks to its mesh, skyboxes, and UV mapping representations being directly compatible with the OpenGL framework. We implement the two MLP shaders as fragment shaders in OpenGL. During each rendering, we first rasterize the mesh and skyboxes to screen space as a set of fragments. The fragment shader then maps each fragment's features to the opacity value and view-dependant RGB color. Finally we sort the scene mesh and skyboxes by depth and perform alpha composition to synthesize the final RGB rendering.

2. Baseline Implementation Details

2.1. Free View Synthesis (FVS)

One key component of FVS is that it requires a proxy geometry to perform warping. Instead of using COLMAP to perform structure-from-motion as in the original paper, we take advantage of the fact that the data collection platform records accurate LiDAR depth information for the scene. For the static scene, we aggregate LiDAR points across all frames with dynamic points removed and then create triangle surfels for them. We first estimate per-point normals from 200 nearest neighbors, and then we downsample the points into $4cm^3$ voxels [22]. After that, triangle surfels are created with a 5cm radius. This includes static vehicles. To enhance the photorealism of the rendered images, we also add the a adversarial loss which helps produce higher quality results.

To render a target image, a few source images need to be selected. During training, we randomly select from nearby frames (10 frames before and 10 frames after). To test the interpolation during inference, we select the 2 most nearby frames. To test the lane shift, we select every 4 frames starting from 12 frames before to 4 frames after current frame. We heuristically tried multiple settings and found this setting usually achieves better balance between performance and efficiency. Fewer source images will result in images with large unseen region, and the network may fail to in-paint it. More source images may produce blurry results because of geometry misalignment and runs out-of-memory on the GPU. The FVS is trained on 2 A5000 GPUs with learning rate 0.0001, batch size 8 for 50000 iterations. Images are cropped to 256×256 patches during training.

2.2. Multi-View Warping

The multi-view warping technique utilizes the same geometric proxy as FVS. The process involves rendering the depth of the target view using the geometric proxy, followed by selecting a set of source images with camera poses that are in close proximity to the target view, and rendering their respective depths. Any distant regions in the scene, such as the sky, are filled with a large value, typically 65535. Subsequently, the source images are warped to the target view using both the depth and extrinsics to find the correspondence. The resulting set of warped images is then blended together to generate a final output image. Each warped image is ranked based on its distance to the target pose, and for each pixel in the output image, only the source pixel with the shortest distance to the target view is retained.

2.3. ENeRF

ENeRF builds a cascade cost volume to predict the coarse geometry of the scene thus enabling efficient depth-guided sampling. We crop the image to 1920×1056 during training and inference as CNN requires the image dimension can be divided by 32. We trained the model from scratch on PandaSet `Log 53` for 50K iterations using official repository [1]. It takes around one day with $2\times$ A5000. Empirically, we find this model performs better compared to using the released DTU model and finetuning. We hypothesize this is because the DTU camera distributions are significantly different from driving scenes. During training, we select two or three source images with closest viewpoints with a probability of 0.9 and 0.1. We choose two source images for fast inference and validate on all 9 logs.

[1]https://github.com/zju3dv/ENeRF

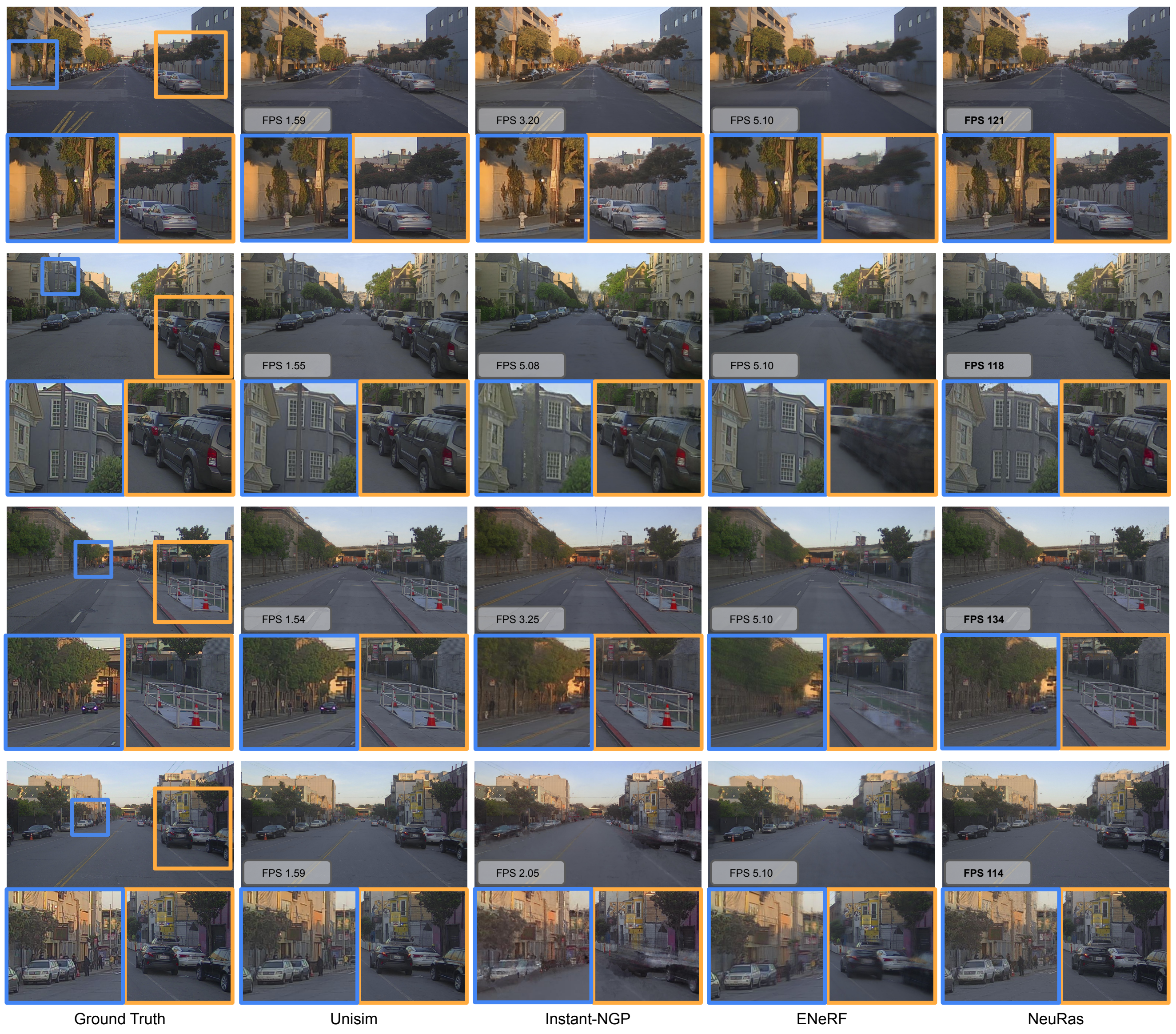

Figure 1. **Qualitative comparison on driving scenes.** We show rendering results in testing frames.

2.4. Instant-NGP

Instant-NGP [9] achieves state-of-the-art NeRF rendering by introducing an efficient hash encoding, accelerated ray sampling and fully fused MLPs. In our experiments, we scale the PandaSet scenes to fully occupy the unit cube and set `aabb_scale` as 32 to handle the background regions (e.g., far-away buildings and sky) outside the unit cube. We tuned `aabb_scale` to be 1, 2, 4, 8, 16, 32, 64 and find 32 leads to the best performance. The model is trained for 20K iterations and converges on the training views. However, we find there are obvious artifacts including floating particles and large missing regions on the ground. This is due to the radiance-geometry ambiguity [21] and limited viewpoints that are sparse and far apart in the driving scenarios compared to dense indoor scenes. Therefore, to improve the geometry for better extrapolation (e.g., lane shift), we enhance the original method with LiDAR depth supervision. Specifically, we aggregate the recorded LiDAR data and create a surfel triangle representation. We also enable the near-surface sampling and early ray termination mechanism in CUDA that accelerate learning and inference. We follow the same surfel asset generation pipeline as stated in the implementation of FVS. We follow official instructions to extract good-quality mesh from instant-NGP models. It is known that the following factors could affect reconstruction quality. Modeling large scenes using hash-encoding may lead to

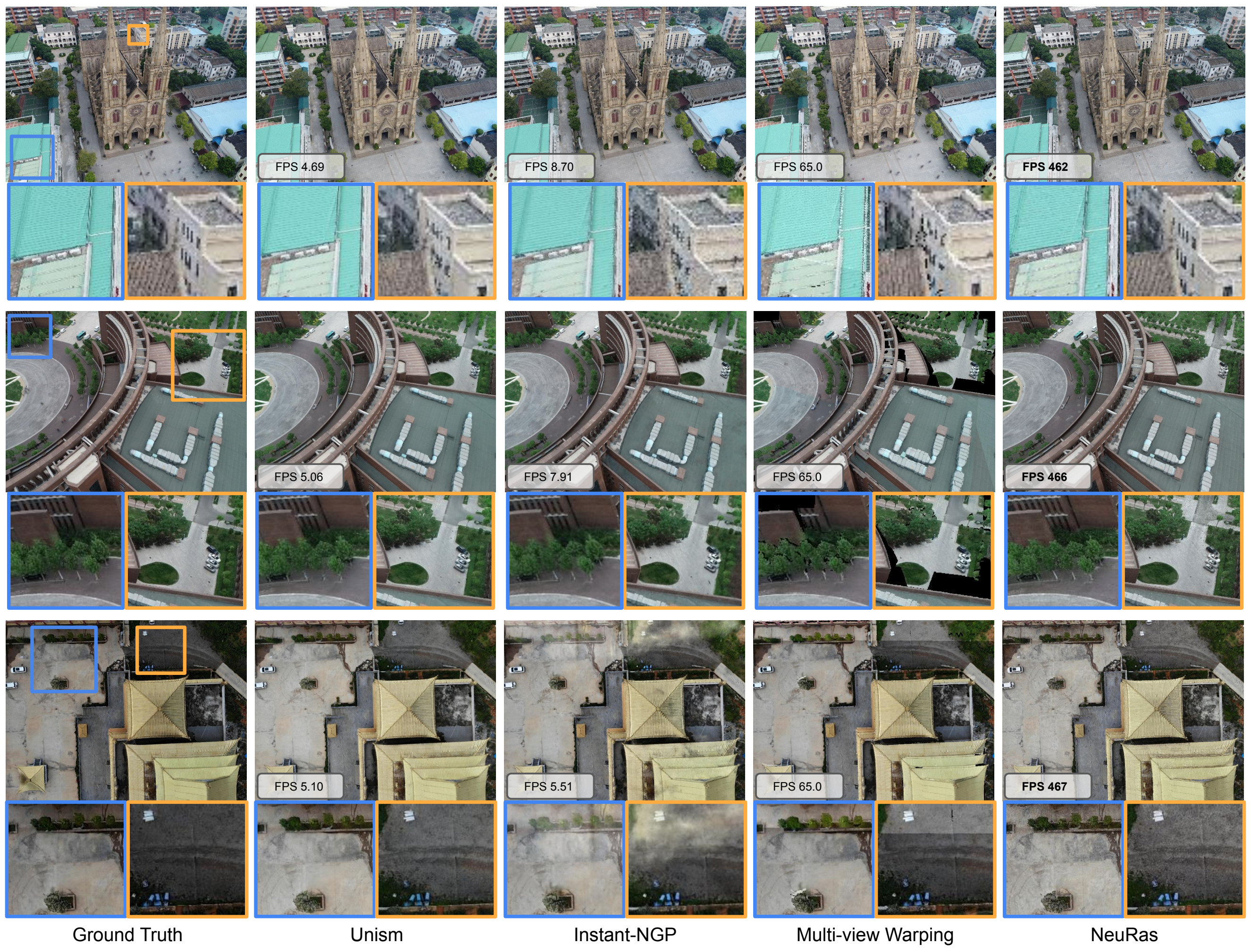

Figure 2. **Qualitative comparison on drone scenes.** We show rendering results in testing frames.

more collisions, and surface regularization is needed for smoother surface reconstruction. Improving these aspects requires additional efforts and remains an open research problem. We then use the surfel mesh representation to render a depth image at each camera training viewpoint. For BlendedMVS, we normalizes the scenes to occupy the unit cube and set `aabb_scale` to be 8. The model is trained for 100K iterations. Note that no depth supervision is used for BlendedMVS. We use the default model on two datasets except for using a large hashmap `log2_hashmap_size = 22`.

2.5. UniSim

UniSim [17] is a neural sensor simulator that takes a set of camera images and LiDAR point cloud from a data collection platform, and then converts them into a digital twin that can be manipulated. It achieves the state-of-the-art results for synthesizing new views and scenarios on self-driving datasets [16]. UniSim represent the 3D scene as the neural feature fields, which are parameterized by multi-resolution feature grids. Given the camera viewpoint and intrinsics, UniSim first perform volume rendering to create an image feature map, then it leverages a 2D CNN network to take the feature map as input and output the RGB image. The CNN network allows for volume rendering the feature map at a lower resolution and upsampling it to full resolution, this can significantly reduces the amount of computation required for ray queries and radiance computation. During the volume rendering process, UniSim also utilizes geometry priors from LiDAR observations and sparsifies the features grids to reduce the computation cost. UniSim is trained with image photometric consistency loss and perceptual loss. In addition to the image loss, it also leverages the LiDAR point cloud to supervise the scene geometry and employs two surface regularizers to ensure the model learns a smooth surface for geometry modelling. Dynamic actor

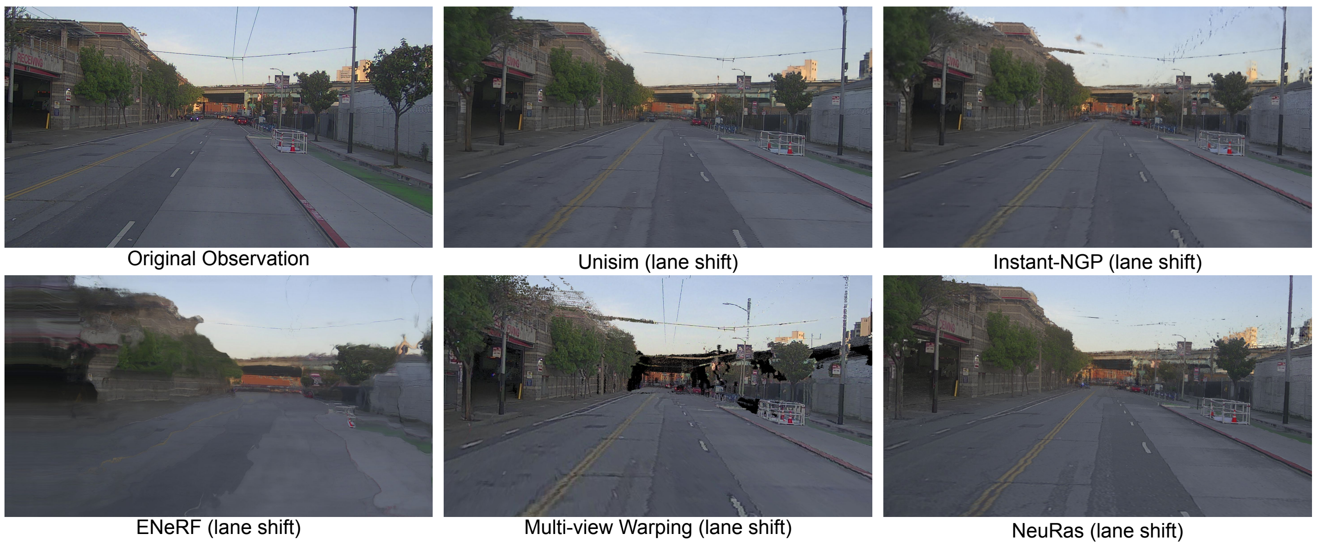

Figure 3. **Extrapolation comparison on driving scenes.** We show lane shift results comparison on driving scenes.

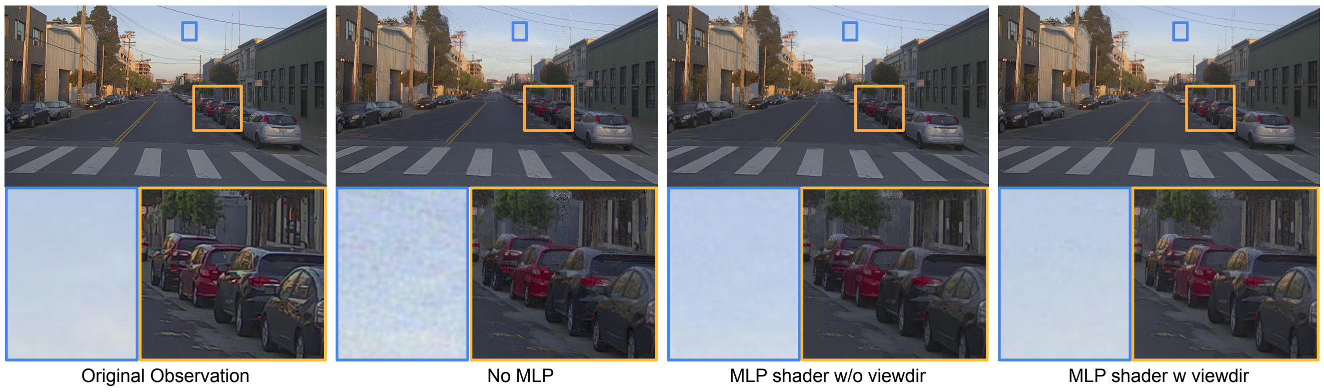

Figure 4. **Effects of MLP shader.** The MLP shader aids in improving the realism of view-dependent results (see the vehicle windows), while also mitigating artifacts in the sky.

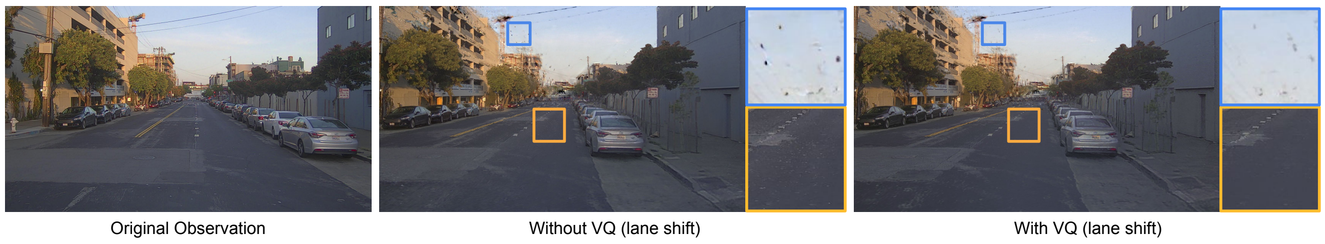

Figure 5. **Effects of vector quantization (VQ).** The vector quantization improves extrapolation results and also reduces offline storage.

models and adversarial training are not included in our setting. For each scene, we trained UniSim for each scene using a single A5000 GPU for 20k iterations, approximately 1.7 hours. The Adam optimizer was used to train UniSim, with a learning rate of 0.001 for the CNN and 0.01 for the neural feature fields. To ensure optimal training, we used exponential learning rate decay to gradually decrease the learning rate by 0.1 throughout the entire training process.

Methods	PSNR↑	SSIM↑	LPIPS↓
W/o MLP shader	22.53	0.780	0.184
W/ MLP shader	**24.19**	**0.790**	**0.176**

Table 1. **Ablation on using MLP shader on BlendedMVS.** The view-dependant MLP shader helps to improve the photorealism. Ablation is conducted on the BlendedMVS dataset.

3. Experiment Details

3.1. Driving Scene Dataset

We conduct experiments on the PandaSet [16] dataset to evaluate self-driving scenes. We use 50 snippets of urban driving scenes captured in San Francisco. Each snippet includes 80 frames (1080×1920 resolution) that last for 8 seconds and cover an area of approximately 300×80 square meters. To quantitatively evaluate our approach against baseline methods that are computationally expensive to train, we selected 9 scenes that have few dynamic actors: `021, 027, 028, 029, 038, 039, 040, 053, 055, 056`.

3.2. Drone Scene Dataset

To evaluate drone scenes, we conduct experiments on the BlendedMVS [19] dataset, which includes a variety of small-scale objects and large-scale outdoor scenes. The five large scenes selected for evaluation are `58eaf1513353456af3a1682a, 5bbb6eb2ea1cfa39f1af7e0c, 5b08286b2775267d5b0634ba, 5b69cc0cb44b61786eb959bf` and `5b6eff8b67b396324c5b2672`. Regarding the UV, we have employed a resolution of $4096{\times}4096$ for four of the logs, whereas the fifth log (5bbb) features a significantly larger scene with a camera positioned closer to the mesh, prompting us to adopt an 8192x8192 resolution for this particular log. We do not use skyboxes for the drone scenes, since they usually only encompass a small fraction of the footage captured. Additional, due to the opaque mesh we use, we are not good at model fog, smoke, window and water .

3.3. NeRF Synthetic Dataset

We encounter challenges when attempting to apply real-time rendering methods like SNeRG [4], PlenOctrees [20], and MobileNeRF [1] to large-scale outdoor scenes and obtain satisfactory results. Therefore we compared our method to these approaches using object-level scenes from the NeRF synthetic dataset [8]. We choose two examples to evaluate the model performance: `Lego` and `Chair` examples. We run Voxurf [15] follow the official repository [2] to extract the geometry scaffold for running our method. The setting is the same, except that we change the MLP to be 5 layers with 16 hidden units to better handle the significant view dependant effects on the dataset. Besides, we set perceptual loss weight to be 0.0001, as they seems insignificant in the synthetic dataset.

4. Additional Results and Analysis

4.1. Ablation Study

MLP shader ablation: Fig. 4 shows qualitative comparison on different shaders. The comparisons are conducted under 3 settings: 1) No MLP, where the scene appearance is represented using an RGB texture map. 2) MLP-shader without view direction, where the MLP does not take the view direction as input and therefore cannot model view-dependent appearance. 3) MLP-shader with view direction, which is the default setting in the main paper. It can be seen from the comparisons that using an MLP-shader can enhance realism in generating view-dependent results, as well as compensate for artifacts in the scene. Besides the quantitative ablation on single log in Table 3 in the main paper, we further conduct ablation on all logs on BlendedMVS, as shows in Table 1, using MLP shader consistency improve the realism.

Vector quantization (VQ) ablation: Fig. 5 shows a visual training a model with and without vector quantization. It can be seen from the figure that vector quantization helps to regularize the neural texture maps, leading to improvement in rendering quality in extrapolation viewpoints. For quantitative analysis, please refer to Table 4 in the main paper.

[2]https://github.com/wutong16/Voxurf

4.2. Additional NVS Results on Large-scale Scenes

We show additional novel view synthesis results in Fig. 1 for PandaSet and Fig. 2 for BlendedMVS large-scale scenes, respectively. We also show qualitative results for *extrapolation setting (Lane Shift)* in Fig. 3. Our NeuRas approach captures fine-grained details and achieve comparable or superior visual realism when compared to the baseline approaches, while being significant faster ($80\times$ faster than UniSim, and $40\times$ faster than Instant-NGP). Our approach strikes the best trade-off between rendering quality and rendering speed.

5. Failure cases and Future Works

While NeuRas is capable of producing photo-realistic results, there is still room for potential improvements in some scenarios. NeuRas is not able to effectively represent delicate and narrow structures, such as powerlines against the sky. Moreover, on occasion, conspicuous artifacts have been detected at the interface of the background skyboxes and the volume. Also, at high zoom levels, we experience a loss of detail. However, future work involving the use of mipmap could potentially resolve this issue. We hope that future works on dynamic actor modelling [10, 17], material modelling [12, 18, 14] and extensions to other sensors such as LiDAR [7, 13, 17, 5] could further enhance the capabilities of NeuRas system for scalable data-driven sensor simulation.

References

[1] 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. 6

[2] 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. 1

[3] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *ICML*, 2020. 1

[4] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *arXiv*, 2021. 6

[5] Shengyu Huang, Zan Gojcic, Zian Wang, Francis Williams, Yoni Kasten, Sanja Fidler, Konrad Schindler, and Or Litany. Neural lidar fields for novel view synthesis. *arXiv preprint*, 2023. 7

[6] 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. 1

[7] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world. In *CVPR*, 2020. 7

[8] 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. 6

[9] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. 2022. 3

[10] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. *CVPR*, 2021. 7

[11] John C. Perry. xatlas: A c++11 library for texture atlas generation. https://github.com/jpcy/xatlas, 2018. GitHub repository. 1

[12] 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. 7

[13] Jingkang Wang, Ava Pun, James Tu, Sivabalan Manivasagam, Abbas Sadat, Sergio Casas, Mengye Ren, and Raquel Urtasun. Advsim: Generating safety-critical scenarios for self-driving vehicles. In *CVPR*, 2021. 7

[14] Zian Wang, Tianchang Shen, Jun Gao, Shengyu Huang, Jacob Munkberg, Jon Hasselgren, Zan Gojcic, Wenzheng Chen, and Sanja Fidler. Neural fields meet explicit geometric representations for inverse rendering of urban scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2023. 7

[15] 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. 6

[16] 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. 1, 4, 6

[17] 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. 1, 4, 7

[18] 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. 7

[19] 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. 6

[20] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. *ICCV*, 2021. 6

[21] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv*, 2020. 3

[22] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018. 2