Supplementary Material for NeO 360: Neural Fields for Sparse View Synthesis of Outdoor Scenes

Muhammad Zubair Irshad[1,2] Sergey Zakharov[2] Katherine Liu[2] Vitor Guizilini[2] Thomas Kollar[2]
Adrien Gaidon[2] Zsolt Kira*[1] Rares Ambrus*[2]

* denotes shared last authorship

[1]Georgia Institute of Technology [2]Toyota Research Institute

{mirshad7, zkira}@gatech.edu, {firstname.lastname}@tri.global

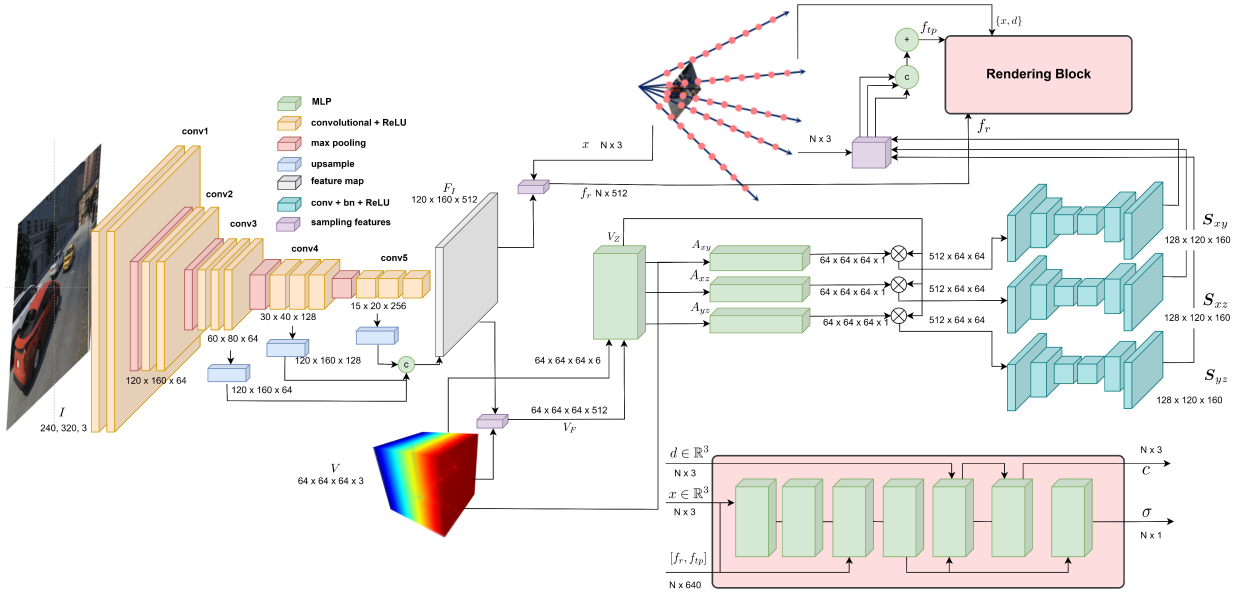

Figure 10: **Detailed Architecture** of NeO 360 showing the construction of image conditional triplanes, along with local residual features and rendering MLPs to output density and color for a 3D point x and viewing direction d.

A. Network Architecture Details:

In this section, we provide more details about our architectural design, specifically our image-conditional tri-planar representation, as discussed in Fig. 2 and our rendering MLPs as discussed in Fig. 1 in the main paper. Our detailed architecture is presented in Fig. 10 and described in the following sections. We first describe the details of our encoder in Sec. A.1, next we describe the details of our tri-planar features and residual features in Sec. A.2. Finally we provide details of our rendering MLPs in Sec. A.3.

A.1. Encoder:

The Encoder network E comprises a pre-trained Resnet32 [4] backbone. We extract features from the initial convolutional layer and subsequent layer 1 to layer 3 and upsample all features to the same spatial resolution (i.e. $H/2 \times W/2$) where W, H are image dimensions respectively, before concatenating along the feature dimension to output feature map F_I with dimension $512 \times H/2 \times W/2$ as shown in Fig. 10.

A.2. Image-conditional tri-planar features and residual features:

In this section, we delve into the image-conditional tri-planar features and residual features. These elements, formed through volumetric projection, depth encoding, feature aggregation, and 2D convolutions, are essential for our approach's effectiveness in enhancing scene understanding.

Volumetric local features: We back project local feature map F_I along every ray to the world grid (V) to get 3D feature volume feature (V_F) with dimensions $K \times K \times K \times 512$ where K is the resolution of feature grid and we use $K = 64$. Note that there is a tradeoff between the size of the feature grid i.e. expressivity and computational cost. We found $K = 64$ to give a reasonable performance while avoiding any OOM (out-of-memory issues) due to a larger grid size in our network training.

Feature depth-encoding: As detailed in Sec. 4.1, we learn the depth of each feature in the feature grid using an additional 2-layer MLP with hidden dimension 512 to output depth-encoded features V_Z of dimensions $K \times K \times K \times 512$. Our feature aggregation module comprises three 2-layer MLPs with hidden dimension 512 and outputs learned weights w_i over individual volumetric feature dimensions to outputs weights w_{xy}, w_{xz} and w_{yz} each with dimensions $K \times K \times K \times 1$. After performing $softmax$ and summing over the z, y and x dimensions respectively, we obtain 2D feature maps for each of the three planes with dimension $K \times K \times 512$.

2D Convolutions: We further use a series of 2D convolutions with upsampling layers to transform the planar features to dimension $H/2 \times W/2 \times 128$. The convolutional layers comprise three convolutional layers with input channels 512, 256, and 128 and output channels 256, 128, and 128 respectively, with a kernel size of 3, a stride of 2, and a padding of 1, followed by an upsampling layer with a scale factor of 2 and another convolutional layer with an input channel and output channel of 128. Finally, an upsampling layer with an output dimension of $H/2 \times W/2$ is employed before outputting the features with a final convolution layer with input and output channels 128. All convolutional layers are followed by the BatchNorm and ReLU layers. The output of each convolutional block becomes our tri-planar features S, each with dimension $128 \times 120 \times 160$. We sample into each plane in S by projecting x into each plane i.e. by getting the absolute xy, xz and yz coordinates of x before concatenating and summing over the channel dimension to retrieve feature f_{tp} with dimension $N \times 128$ where N denotes the number of sampled points and 128 is the feature dimension. The residual local feature f_r after sampling into F_I has dimensions $N \times 512$.

A.3. Rendering MLPs:

The rendering MLPs for both foreground and background rendering comprises 7 fully-connected layers with hidden dimension of 128 and ReLU activation. We apply positional encoding [6] to the input positions x and viewing direction d. We concatenate positions x with triplanar features f_{tp} and residual features f_r as an input to the first layer of the MLP. We also supply the conditioning feature as a skip connection to the third layer in the MLP and mean pool the features along the viewing dimension in the forth MLP layer, if there is more than one image in the input. We found this pooling strategy to work better than pooling before the rendering stage i.e. earlier on in the tri-planar construction stage (A.2). In total, we use the first 4 layers to output features of dimension $N \times 128$, before utilizing a final density MLP to output 1 channel value for every sampled point N. We further use two additional dedicated MLP layers with a hidden dimension of 128 to output a 3-channel color value for every sampled point N, conditioned on the positionally-encoded viewing direction and the output of the fourth MLP layer.

B. Implementation Details:

Sampling rays: We scale all samples in the dataset so that cameras lie inside a unit hemisphere and use near and far values of 0.02 and 3.0 respectively. We use 64 coarse and 64 fine samples to sample each ray.

Training procedure: To optimize NeO 360, we first sample 3 source images from one of the 75 scenes in the training dataset. For our initial training phase, we sample 20 random destination views different from the source images used for encoding the NeO 360's network. We sample 1000 rays from all 20 destination views. We use these randomly sampled rays to decode the color and density for each of the 1000 rays. This training strategy helps the network simultaneously decode from a variety of camera distributions and helps with network convergence. We do this by sampling two different sets of points i.e. one for each near and far background MLP, as employed in [7]. These points samples differ based on the intersection between the origin of rays and the unit sphere.

Loss function and optimizer: For the first training phase, we employ a mean squared error loss on predicted color and target pixels at the sampled point locations in the ground-truth images, as discussed in Sec. 4.3. We also add a regularization penalty (Eq. 9) to encourage the weights to be sparse for both near and far background MLPs, as proposed in [1]. For our second training phase, we select a single destination view and sample 40×40 patches of target RGB for training the network using an additional perceptual similarity loss, as described in Sec. 4.3 with a λ value set to 0.3.L_{LPIPS} loss encourages perceptual similarity between patches of rendered color, c^p and ground color $\tilde{c}_t$, where we only enforce it after 30 training epochs to improve background modeling. We optimize the network for 100 epochs in total and employ early stopping based on the validation dataset which is a subset of the training dataset with different viewpoints than the training camera distribution. We use an Adam optimizer with an initial learning rate of $5.0e^{-4}$ and a learning-rate ramp-up strategy to increase the learning rate from $5.0e^{-5}$ to the value $5.0e^{-4}$ and then decrease it exponentially to a final learning rate $5.0e^{-6}$.

Figure 11: **NeRDS360 training samples:** We show diverse training samples from our proposed multi-view dataset, showing 3 examples for each map (shown on the y-axis) and displaying 5 randomly sampled images (shown on the x-axis) for each scene, from 100 rendered images with cameras placed in a hemisphere at a fixed radius from the center of the scene.

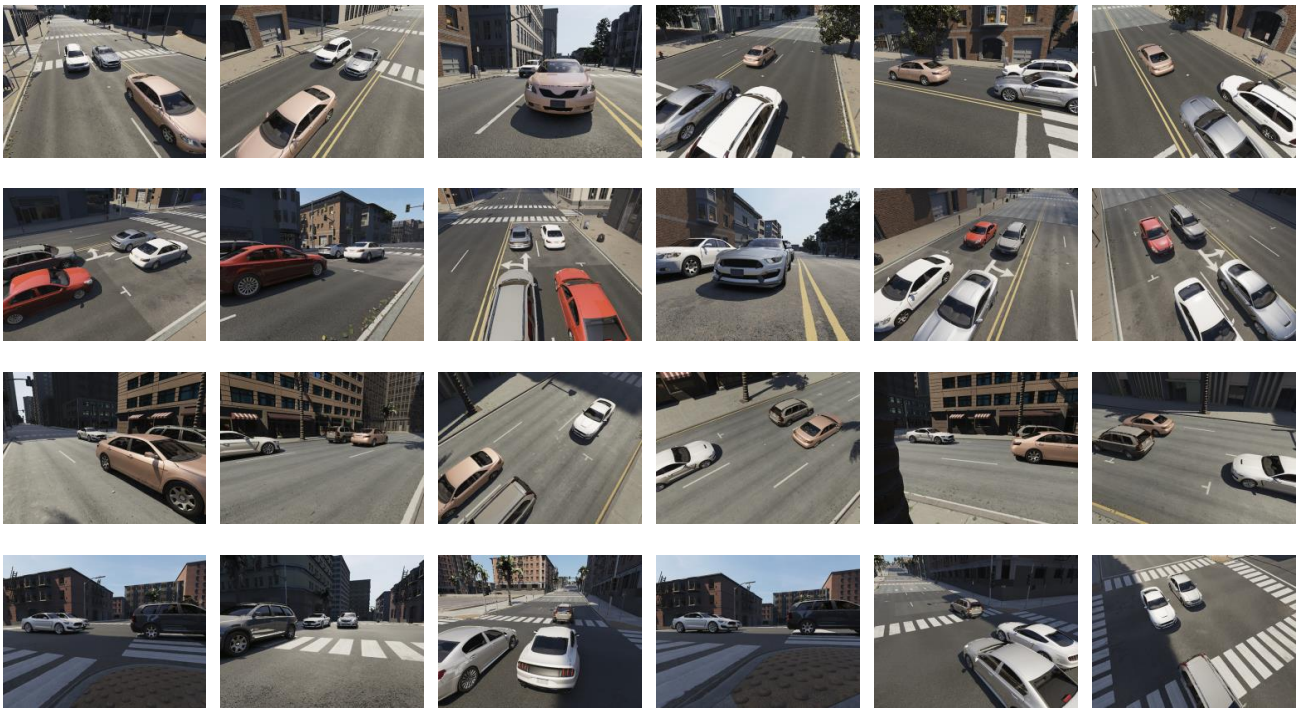

Figure 12: **NeRDS360 test samples:** We show unseen test samples with completely different backgrounds and objects not seen during training. Test samples include completely different camera viewpoints that are not observed during training which are still sampled in a hemisphere around the foreground objects of interest. Here, we show 4 different scenes from our evaluation dataset, different from the training dataset (shown on the y-axis) and show 5 randomly sampled images (shown on the x-axis) for each scene, from 100 rendered images with cameras placed in a hemisphere at a fixed radius from the center of the scene.

Compute: We train the model end-to-end on 8 A-100 Nvidia GPUs for approximately 1 day for network convergence.

Parameters: Since NeO 360 has the ability to overfit to a large number of scenes, unlike NeRF [6], we use a larger model size of 17M parameters. Both ours and NeRF [6]'s rendering MLP size is the same (i.e. 1.2M parameters), although our larger model size is attributed to employing ResNet feature block for local features ($\sim$10M parameters) and additional convolutional blocks for tri-planar feature.

Optional fine-tuning: Although our network gives reasonable zero-shot performance, we also employ an additional finetuning stage using the same few views (e.g. 1, 3 and 5 source views) to further improve the performance of our network. Note that we employ the same finetuning strategy for the comparing baselines (cf. Sec. 5 in the main paper) and show that the additional finetuning stage improves the performance of both our proposed method and competing baseline, while our approach, NeO 360, still achieves superior overall performance. For our finetuning experiments, we freeze the rest of the network and only the optimize tri-planar network i.e. freezing the encoder E. We employ a lower learning rate of $5^{10^{-6}}$ to finetune the network from 1,3 or 5 source views.

$$\mathcal{L}_{\text{reg}}(s,w) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} w_i w_j \left|\frac{s_i+s_{i+1}}{2} - \frac{s_j+s_{j+1}}{2}\right| + \frac{1}{3}\sum_{i=0}^{N-1} w_i^2 (s_{i+1}-s_i) \tag{9}$$

C. Experimental Setting Details:

In this section, we detail our experimental setting to evaluate the effectiveness of our proposed method against the state-of-the-art baselines on the NeRDS360 dataset. We mainly evaluate for **a.** Prior-based sampling and **b.** Novel-scene rendering. Note that unlike [2] which performs both unconditional and conditional prior-based sampling, our task only considers image-conditional prior-based sampling for **a**, since we don't optimize a latent code for each scene and our method doesn't rely on inference-time GAN-inversion like [2] to find a latent code for a new scene. Rather, our method works in a zero-shot manner reason-

ably well without any inference time finetuning or inversion, since it takes as input one or few images or a novel scene and is trained as such. We now describe more details about each experimental setting. **a. Prior-based sampling** tests for our network's ability to overfit the training distribution of a large number of scenes. In essence, we keep the evaluation scenes fixed to one of the scenes seen during training and use 1,3, and 5 source camera views as input while decoding from novel camera viewpoints not seen during training. While vanilla NeRF [6] can do this with many different networks, each optimized from scratch from 100s of views for a new scene, our proposed approach, thanks to its generalizability can overfit to a large number of scenes with just a single network without optimizing a different latent code or vector per-scene, hence demonstrating our network's ability to memorize the training distribution for a large number of scenes seen during training. **b. Novel-scene rendering** considers evaluating our approach on a completely new set of scenes and objects never seen during training. We test for our model's ability to generalize well in this scenario which is a core aspect of our approach. This is a more challenging evaluation setup than prior-based sampling since the network has not seen any scenes or objects, neither it has seen these viewpoints during training. Rather, it only relies on the priors learned during training and the few views available during testing (1, 3, or 5 views in our evaluation setup) to infer the complete 360° surroundings of novel scenes.

D. NeRDS360 Dataset:

In this section, we discuss our proposed NeRDS360 dataset. We show qualitative examples of our proposed dataset in Fig. 11 and Fig. 12. Fig. 11 displays training samples of 3 different scenes from each of the 3 different maps in our dataset. Our dataset is very diverse both in terms of the scenes represented and the foreground car shapes and textures. NeRDS360's scenes also depict high variety in terms of occlusion of foreground objects (i.e. not all foreground cars are observed from all views and there are various occluders such as trees and lightning poles present in the scene), varied number of objects represented (i.e. we sample from 1 to 4 foreground cars for each scene with various textures, lightning, and shadows) as well as varied lighting and shadows in a scene (i.e. lightning and shadows in each scene is not constant). Hence, our dataset and the corresponding task are extremely challenging. We also show different testing samples in Fig. 12. As shown in the figure, we render completely novel viewpoints not seen during training as well as different textures and shapes of cars that are also not rendered during training. We evaluate for all 100 evaluation cameras sampled inside the hemisphere, as shown in Fig. 5 in the main paper while giving as input 1, 3, or 5 source views to the network.

Figure 13: **Real-world results:** on KITTI-360 [5], Panoptic NeRF [3] test split

E. Additional Qualitative Results:

Results on Kitti-360: To demonstrate its applicability to real-world data, it's important to capture a comparable dataset to NeRDS360 in a real-world context. While our approach is tailored for a 360° environment, we've successfully adapted NeO 360 for the KITTI-360 [5] dataset. This adaptation involves removing the distinction between near and far and employing a single MLP for rendering. Our method employs a source view window of the last 3 frames to render the subsequent frame. Examining overfitting outcomes (Fig. 13), we observe that our representation achieves significantly improved SSIM and comparable PSNR in contrast to NeRF, when dense views are available for real-world unbounded scenes.

References

[1] 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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 2

[2] Miguel Angel Bautista, Pengsheng Guo, Samira Abnar, Walter Talbott, Alexander Toshev, Zhuoyuan Chen, Laurent Dinh, Shuangfei Zhai, Hanlin Goh, Daniel Ulbricht, et al. Gaudi: A neural architect for immersive 3d scene generation. *arXiv preprint arXiv:2207.13751*, 2022. 4

[3] Xiao Fu, Shangzhan Zhang, Tianrun Chen, Yichong Lu, Lanyun Zhu, Xiaowei Zhou, Andreas Geiger, and Yiyi Liao. Panoptic nerf: 3d-to-2d label transfer for panoptic urban scene segmentation. In *International Conference on 3D Vision (3DV)*, 2022. 5

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[5] Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *Pattern Analysis and Machine Intelligence (PAMI)*, 2022. 5

[6] 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. In *ECCV*, 2020. 2, 4, 5

[7] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 2