

LidaRF: Delving into Lidar for Neural Radiance Field on Street Scenes

Supplementary Material

Shanlin Sun¹ Bingbing Zhuang³ Ziyu Jiang³ Buyu Liu³
Xiaohui Xie¹ Manmohan Chandraker^{2,3}

¹University of California, Irvine ²University of California, San Diego ³NEC Labs America

A. Additional Ablation Results

Lidar Encoding. Here, we discuss additional experiments to study the advantage of our Lidar encoding based on the 3D sparse convolutional network. Firstly, in view of the increased representation capacity brought by Lidar encoding, we evaluate the impact of naively increasing the hash grid feature size without using any Lidar encoding, specifically doubling it from two to four features per level, denoted as “Double Hash” in Table S2. This modification yields improved results in interpolation settings but exhibits a marginal performance decline in lane shift scenarios, indicative of potential overfitting.

Furthermore, instead of 3D convolutional network, we explore LiDAR encoding utilizing MLPs (as applied in [2]) and PointNet++[7]. For the former, the MLP contains three hidden layers with feature size (64, 96, 128), and a final layer outputting a 64-dim feature vector. For PointNet++, the hidden feature dimensions mirror those used in our sparse UNet-based method, and its point encoder samples 4096, 1024, 256, and 56 points at different levels. To enhance the efficiency of its farthest point sampling and neighbor point grouping, we utilize CUDA-based implementations from PyTorch3D [8]. The results, as outlined in Table S2, indicate that Lidar encoding with MLP and PointNet++ underperforms our encoding with sparse UNet. This indicates the benefits brought about by learning from a more global context with the 3D convolutional network, which has proven a powerful backbone widely applied in state-of-the-art 3D perception frameworks [5, 11].

Lastly, the memory and time overhead incurred by our LiDAR encoding module is modest. Specifically, compared to Nerfacto with robust depth supervision, our LiDAR encoding introduces an additional memory usage of 1846MB and an incremental time cost of 0.1s per training iteration. Notably, during inference, this overhead is further reduced as LiDAR encoding is required only once per scene, rather than per batch.

Robust Depth Supervision. In Fig. S1, we present visual

comparisons of different depth supervision settings under the lane shift scenario. With sparse or noisy depth supervision, Nerfacto w/ \mathcal{L}_{ds}^1 and Nerfacto w/ \mathcal{L}_{ds}^{10} fail to model thin structures such as light poles. In contrast, our proposed scheme is robust to occlusions and able to learn delicate structures with noisy depth maps.

Augmented View Supervision. Here, we provide more analyses on the impact of augmented view supervision. As shown in Fig. S2(a), we observe good performance with around 80-320 synthetic views, beyond which performance drops as it may dominate real training views. We randomly perturb the original views with Gaussian noise, and observe good performance with the standard deviation σ around 1.0-1.5 meters, as shown in Fig. S2(b). We did not observe noticeable benefits from adding new orientations, likely as vehicles are mostly in forward motion.

CDF and Mid-point Approximation. As mentioned in the main paper, Tab. S4 provides a quantitative comparison of two implementations of LiDAR depth loss using single-frame LiDAR points. We observe improved performance from our exact implementation based on CDF, in comparison to the mid-point approximation. Despite the marginal gap, they are consistently observed across all tested sequences in the Pandaset, particularly in the interpolation setting.

B. Quantitative Results by Range

We perform more detailed evaluations by separating pixels into different depth ranges. In Tab. S1, we evaluate under two strategies for range separation – 1) group the pixels with matching Lidar points into different distance ranges; 2) group pixels into foreground, sky, and their boundary region that is prone to artifacts due to discontinuity. The quantitative results indicate superior performance from our approach regardless of the range, compared to UniSim. That said, our proposed components are compatible to UniSim and may be combined in future work.

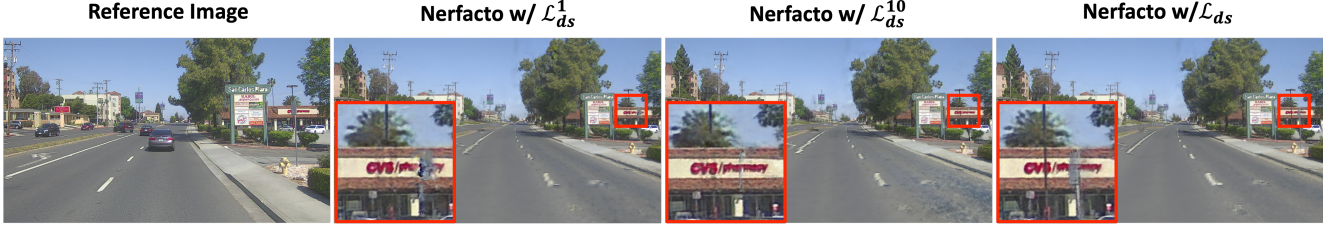


Figure S1. **Qualitative comparison between different LiDAR depth supervisions on shift lane setting.** Our proposed method achieve significantly better rendering quality on the delicate structures.

Methods	Range by Distance			Range by Semantics		
	Near ($\leq 10m$)	Middle (10-60m)	Far ($\geq 60m$)	Foreground	Boundary	Sky
UniSim	25.44 25.92	25.55 26.07	22.41 21.99	25.44 26.32	22.07 22.23	34.81 36.61
Ours	26.68 27.62	27.64 28.36	23.73 23.52	26.71 28.17	23.64 23.93	39.06 41.17

Table S1. PSNR (mean|median) evaluation separated by range.

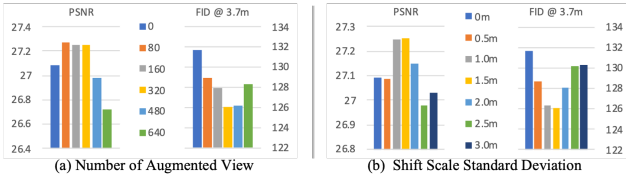


Figure S2. Analysis on the impact of augmented synthetic views.

Methods	Interpolation			Lane Shift	
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	FID \downarrow @ 2m	FID \downarrow @ 3.7m
Original Hash	27.090	0.804	0.247	110.0	131.7
Double Hash	27.153	0.808	0.234	109.3	132.1
MLP	27.119	0.805	0.246	108.0	131.6
PointNet++	27.076	0.804	0.247	108.7	131.2
Ours	27.219	0.810	0.228	105.6	128.7

Table S2. **Quantitative comparisons on Lidar encoding designs** – Our proposed sparse convolution based Lidar encoding is better than simply doubling hash grid feature sizes and encoding Lidar feature with MLP or PointNet++ [7]. “Original Hash” indicates Nerfacto using our robust depth supervision. “Double Hash” doubles hash feature size.

C. Experiments on Argoverse

As mentioned in the main paper, in this section we provide more comprehensive evaluation on the Argoverse [10] dataset in order to compare with the closely related work of Chang et al. [2]. We train our model on the 8 sequences selected by [2], with their split of training and validation set. We use their open-source code along with the released models to reproduce the rendering results of [2].

We report the quantitative comparison in Tab. S3. It is evident that our method consistently outperforms [2] with a significant margin across all 8 sequences. In addition, we show more qualitative examples in Fig. S3. As can be seen, our rendering is not only complete (without blank pixels)

but also with far higher resolution than the results from [2].

D. Additional Implementation Details

D.1. Data Processing

Mask Dynamic Object. We mask dynamic objects from images given annotated 2D bounding boxes. Specifically, a Mask-RCNN model takes as input the bounding boxes of dynamic objects, and outputs the corresponding masks. Pixel points within the dynamic object masks are not sampled during training and not counted when computing PSNR, SSIM and LPIPS. We also remove all Lidar points within the dynamic 3D bounding boxes. The remaining Lidar points are used to generate depth maps and augmented data. In PandaSet, the dynamic objects are in eight categories: ‘Car’, ‘Pickup Truck’, ‘Medium-sized Truck’, ‘Semi-truck’, ‘Other Vehicle - Construction Vehicle’, ‘Other Vehicle - Uncommon’, ‘Other Vehicle - Pedicab’, ‘Emergency Vehicle’ and ‘Bus’.

Lidar Depth Generation. In PandaSet, each Lidar frame is paired with a synchronized RGB frame. Lidar depth maps are generated from the accumulated training Lidar frames. Specifically, for a given image frame, Lidar points from ten closest Lidar frames are transformed into world coordinates and then projected onto the image coordinates. The depth value represents the distance along the ray from the camera’s center to a Lidar point. When multiple Lidar points are projected onto the same pixel in the image, the depth value is derived from the nearest Lidar point.

Scene Normalization. Scene normalization is required to apply the scene contraction strategy [1] for handling unbounded scene. We do so based on the radius of camera trajectory. Specifically, we model the scene of interest with a sphere, whose diameter is the maximum distance between

Methods	Seq. 4d7b			Seq. 2b04			Seq. 4690			Seq. 0a13		
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
Chang et al. [2]	24.319	0.652	0.127	22.257	0.598	0.123	26.049	0.728	0.181	20.318	0.615	0.161
Ours	30.186	0.855	0.066	29.540	0.906	0.049	31.892	0.886	0.077	26.520	0.855	0.110

Methods	Seq. 2aea			Seq. 42c8			Seq. 4d32			Seq. 3e7c		
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
Chang et al. [2]	24.901	0.706	0.098	22.990	0.708	0.161	25.186	0.706	0.143	26.440	0.720	0.143
Ours	31.996	0.885	0.071	29.650	0.881	0.121	30.748	0.853	0.116	33.876	0.912	0.084

Table S3. **Quantitative comparisons on Argoverse with Chang et al. [2].** We report PSNR, SSIM, and LPIPS on eight different sequences.

Methods	Interpolation			Lane Shift	
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	FID \downarrow @ 2m	FID \downarrow @ 3.7m
Mid-Point	26.945	0.795	0.270	112.7	139.0
CDF	27.017	0.800	0.264	111.2	138.2

Table S4. **Quantitative comparisons on depth loss implementation.**

any two camera positions in the training log, plus 50 meters.

D.2. Network Architectures

Our network architecture can be decomposed into four main parts: proposal sampler, Lidar encoding, density network and color network.

Proposal sampler is composed of two consecutive neural density fields, each represented by “fused” MLPs [6]. These MLPs employ hash encoding to process 3D positions, followed by density prediction using an MLP. Both fused MLPs share common parameters: a minimum hash grid resolution of 16, five levels of hash encoding, 2-dim features for each hash encoding level, two MLP layers, and 16 hidden features in the MLP layers. The primary distinction between these two fused MLPs lies in their maximum hash grid resolution, with the first set at 512 and the second at 1024.

Lidar encoding takes as input all Lidar points and outputs high dimensional features for each Lidar point, which are then queried by sampled positions. To get started, the LiDAR points undergo voxelization within a grid of $512 \times 512 \times 512$ cells, with the cell features represented by the mean 3D positions of the LiDAR points they contain. Owing to the inherent sparsity, most cells remain empty, signifying an absence of LiDAR points and hence carrying negligible information. Subsequently, a 3D sparse UNet [3], adhering to the encoder-decoder architecture similar to its 3D dense counterpart, is employed for encoding LiDAR geometry. This architecture integrates skip connections and multi-level feature fusion. Sparse convolution, a key component of this process, adapts the conventional convolution operation by applying filters exclusively to active (non-zero) input elements, thereby substantially enhancing

computational efficiency and reducing memory demands. Within this architecture, the encoder’s feature dimensions are set at 32, 64, 96, and 128, while the decoder’s feature dimensions are 128, 96, 64, and 64. The Lidar features are the output of the final layer of this sparse UNet. Our implementation relies on the torchsparse [9] package.

Density network computes hash encoding, then fuses it with the Lidar encoding, and pass them to an MLP. Specifically, it is characterized by a hash grid with multiple resolutions ranging from a minimum of 16 to a maximum of 4096, and includes 16 levels of hash encoding, with each level comprising 2-dim features. The hash features queried from the hash grid are concatenated with the Lidar encoding and then passed to an MLP. The MLP component consists of two layers, each with 64-dim hidden features. In addition to predicting density values, the network also generates a 15-dimensional density embedding for each sampled position, which is subsequently fed into the color network.

Color network takes as input the density embedding and the ray direction encoded via spherical harmonics. It employs a two-layer MLP with 64-dim hidden features to predict an RGB value for each sampled position.

D.3. Learning Hyper-parameters

Our loss weights are set to $\lambda_1 = 0.0005$, $\lambda_2 = 1$, $\lambda_3 = 0.005$ and $\lambda_4 = 1$. We set all ϵ values in the unnormalized scale ($\epsilon_t^0 = 10m$, $\epsilon_t = 100m$, $\epsilon_o^0 = 1m$, $\epsilon_o = 0.15m$, $\epsilon_n = 0.15m$, $\epsilon_a = 1.5m$). The scheduling rate α_t and α_o is set to be 1.00004 and 0.99995, respectively.

All modules in our network are optimized end-to-end for 100000 iterations, with RAdam [4] algorithm. We utilize a per-iteration decay in the learning rate for each parameter. The learning rate is initially set at 0.01 and gradually reduced to 0.0001, with the learning rate scheduler having a maximum limit of 50,000 iterations. The number of sampled points per iteration is 4096.

References

- [1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded

- anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 2
- [2] MingFang Chang, Akash Sharma, Michael Kaess, and Simon Lucey. Neural radiance field with lidar maps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17914–17923, 2023. 1, 2, 3, 5
- [3] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 3
- [4] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019. 3
- [5] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela L Rus, and Song Han. Bevfusion: Multi-task multi-sensor fusion with unified bird’s-eye view representation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023. 1
- [6] Thomas Müller. tiny-cuda-nn, 2021. 3
- [7] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 1, 2
- [8] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020. 1
- [9] Haotian Tang, Shang Yang, Zhijian Liu, Ke Hong, Zhongming Yu, Xiuyu Li, Guohao Dai, Yu Wang, and Song Han. Torchsparse++: Efficient training and inference framework for sparse convolution on gpus. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023. 3
- [10] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, et al. Argoverse 2: Next generation datasets for self-driving perception and forecasting. *arXiv preprint arXiv:2301.00493*, 2023. 2
- [11] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021. 1



Figure S3. **Qualitative comparison on the Argoverse dataset.** Our results are complete and of significantly higher resolution in comparison to those from Chang et al. [2].