NeuRAD: Neural Rendering for Autonomous Driving

Supplementary Material

In the supplementary material, we provide implementation details for our method and baselines, evaluation details, and additional results. In Appendix A, we describe our network architecture more closely and provide hyperparameter values. In Appendix B, we provide details on the experimental setting. Then, in Appendix C, we provide details on our baseline implementation. We closely describe the process of inferring lidar rays that did not return in Appendix D. Next, we cover additional details of our proposed rolling shutter modeling in Appendix E. Last, in Appendix G, we showcase additional results and highlight some limitations of our method.

A. Implementation details

Here we describe our model and training in more detail.

Learning: We train all parts of our model jointly for 20,000 iterations, using the Adam optimizer. In each iteration, we randomly sample 16,384 lidar rays, and 40,960 camera rays, the latter corresponding to 40 (32×32) patches. For most parameters, we use a learning rate of 0.01, with a short warmup of 500 steps. For the actor trajectory optimization and the CNN decoder, we adopt a longer warmup of 2500 steps, and a lower learning rate of 0.001. If enabled, camera optimization uses a learning rate of 0.0001, also with a warmup of 2500. We use learning rate schedules that decay the rate by an order of magnitude over the course of the training.

Networks: As we primarily compare our method with UniSim [13], we follow their network design to a large degree. Our first (geo) MLP has one hidden layer, our second (feature) MLP has two hidden layers, and the lidar decoder also has two hidden layers. For details on the CNN decoder, we refer to Appendix C.2. All networks use a hidden dimension of 32, which is also the dimensionality of the intermediate NFF features.

Hashgrids: We use the efficient hashgrid implementation from tiny-cuda-nn [7], with two separate hashgrids for the static scene and the dynamic actors. We use a much larger hash table for the static world, as actors only occupy a small portion of the scene, see Tab. 5.

Proposal Sampling: First, we draw uniform samples according to the power function $\mathcal{P}(0.1x, -1.0)$ [2], where we have adjusted the parameters to better match our automotive scenes. Next, we perform two rounds of proposal sampling, represented by two separate density fields. Both fields use our actor-aware hash encoding, but with smaller hash tables and a feature dimension of one in the hash tables. Instead of an MLP, we decode density with a single linear layer. The proposal fields are supervised with the anti-aliased online distillation approach proposed for ZipNeRF [2]. Additionally, we supervise lidar rays directly with $\mathcal{L}^{\text{d}}$ and $\mathcal{L}^{\text{w}}$.

NeuRAD-2x: We upscale NeuRAD in a straightforward manner – by doubling the size of all hash tables, thereby approximately doubling the model's parameter count. As this model is primarily intended for long sequences and large scenes, we also double the resolution of each level of the static hashgrid. To accommodate the expanded model complexity, we extend the training to 50,000 iterations and adjust the warm-up periods correspondingly. All other hyperparameters remain the same. We find that while further scaling offers benefits in some cases, it leads to diminishing returns in others.

B. Evaluation details

Here, we describe in detail the evaluation protocol of each SoTA method that we have compared NeuRAD to.

Pandaset (UniSim): UniSim uses a simple evaluation protocol, where the entire sequence is used, with every other frame selected for evaluation and the remaining half of the frames for training. The authors report numbers for the front camera and the 360° lidar on the following sequences: `001, 011, 016, 028, 053, 063, 084, 106, 123, 158`. We call this protocol Panda FC, and additionally report Panda 360 results, with all 6 cameras (and the 360° lidar). For the backward-facing camera, we crop away 250 pixels from the bottom of the image, as this mainly shows the trunk of the ego vehicle.

nuScenes (S-NeRF): S-Nerf uses four sequences for evaluation: `0164, 0209, 0359, 0916`. The first 20 samples from each sequence are discarded, and the next 40 consecutive samples are considered for training and evaluation. The remaining samples are also discarded. Out of the selected samples, every fourth is used for evaluation and the rest are used for training. We train and evaluate on all 6 cameras.

KITTI (MARS): MARS reports NVS quality on a single sequence, `0006`, on frames 5-260. We adopt their 50%-protocol, where half of the frames are used for training, and 25% for evaluation. Following their implementation, we adopt a repeating pattern where two consecutive frames are used for training, one is discarded, and the fourth is used for evaluation.

Argoverse 2 & ZOD: Here, we use a simple evaluation protocol that is analogous to that used for PandaSet. We select 10 diverse sequences for each dataset, and use each sequence in its entirety, alternating frames for training

Table 5. Hyperparameters for NeuRAD.

	Hyperparameter	Value
Neural feature field	RGB upsampling factor	3
	proposal samples	128, 64
	SDF β	20.0 (learnable)
	power function λ	-1.0
	power function scale	0.1
	appearance embedding dim	16
	hidden dim (all networks)	32
	NFF feature dim	32
Hashgrids	hashgrid features per level	4
	actor hashgrid levels	4
	actor hashgrid size	2^{15}
	static hashgrid levels	8
	static hashgrid size	2^{22}
	proposal features per level	1
	proposal static hashgrid size	2^{20}
	proposal actor hashgrid size	2^{15}
Loss weights	λ^{rgb}	5.0
	λ^{vgg}	5e-2
	λ^{int}	1e-1
	λ^{d}	1e-2
	λ^{w}	1e-2
	λ^{p_d}	1e-2
	proposal λ^{d}	1e-3
	proposal λ^{w}	1e-3
	interlevel loss multiplier	1e-3
Learning rates	actor trajectory lr	1e-3
	cnn lr	1e-3
	camera optimization lr	1e-4
	remaining parameters lr	1e-2

and evaluation. For Argoverse, we use all surround cameras and both lidars on the following sequences: `05fa5048-f355-3274-b565-c0ddc547b315`, `0b86f508-5df9-4a46-bc59-5b9536dbde9f`, `185d3943-dd15-397a-8b2e-69cd86628fb7`, `25e5c600-36fe-3245-9cc0-40ef91620c22`, `27be7d34-ecb4-377b-8477-ccfd7cf4d0bc`, `280269f9-6111-311d-b351-ce9f63f88c81`, `2f2321d2-7912-3567-a789-25e46a145bda`, `3bffdcff-c3a7-38b6-a0f2-64196d130958`, `44adf4c4-6064-362f-94d3-323ed42cfda9`, `5589de60-1727-3e3f-9423-33437fc5da4b`. For ZOD, we use the front-facing camera and all three lidars on the following sequences: `000784`, `000005`, `000030`, `000221`, `000231`, `000387`, `001186`, `000657`, `000581`, `000619`, `000546`, `000244`, `000811`. As ZOD does not provide sequence annotations, we use a lidar-based object detector and create tracklets using ImmortalTracker [9].

Ablation Dataset: We perform all ablations on 20 sequences, four from each dataset considered above. We use sequences `001`, `011`, `063`, `106` for PandaSet, `0164`, `0209`, `0359`, `0916` for nuScenes, `0006`, `0010`, `0000`, `0002` for KITTI, `000030`, `000221`, `000657`, `000005` for ZOD, and `280269f9-6111-311d-b351-ce9f63f88c81`, `185d3943-dd15-397a-8b2e-69cd86628fb7`, `05fa5048-f355-3274-b565-c0ddc547b315`, `0b86f508-5df9-4a46-bc59-5b9536dbde9f` for Argoverse 2. Here, we no longer adopt the dataset-specific evaluation protocols corresponding to each SoTA method. Instead, we evaluate on the full sequences, on all available sensors, alternating frames for training and evaluation. The exception is nuScenes, where we find the provided poses to be too poor to train on the full sequences. If we optimize poses during training, we get qualitatively good results, and strong FID scores, but poor reconstruction scores due to misalignment between the learned poses and the evaluation poses, see Appendix G for a more detailed exposition. Therefore, we re-use S-NeRF's shortened evaluation protocol, where this problem is mostly avoided, and leave the problem of proper evaluation on nuScenes for future work.

C. UniSim implementation details

UniSim [13] is a neural closed-loop sensor simulator. It features realistic renderings and imposes few assumptions about available supervision, *i.e.*, it only requires camera images, lidar point clouds, sensor poses, and 3D bounding boxes with tracklets for dynamic actors. These characteristics make UniSim a suitable baseline, as it is easy to apply to new autonomous driving datasets. However, the code is closed-source and there are no unofficial implementations either. Therefore, we opt to reimplement UniSim, and as our model, we do so in Nerfstudio [8]. As the UniSim main article does not specify many model specifics, we rely on the supplementary material available through IEEE Xplore[1]. Nonetheless, some details remain undisclosed, and we have tuned these hyperparameters to match the reported performance on the 10 selected PandaSet [12] sequences. We describe the design choices and known differences below.

C.1. Data processing

Occupancy grid dilation: UniSim uses uniform sampling to generate queries for its neural feature field. Inside dynamic actors' bounding boxes, the step size is 5 cm and inside the static field, the step size is 20 cm. To remove

[1] `https://ieeexplore.ieee.org/document/10204923/media`

samples far from any surfaces and avoid unnecessary processing, UniSim deploys an occupancy grid. The grid, of cell size $0.5\,\mathrm{m}$, is initialized using accumulated lidar point clouds where the points inside the dynamic actors have been removed. A grid cell is marked occupied if at least one lidar point falls inside of it. Further, the occupancy grid is dilated to account for point cloud sparseness. We set the dilation factor to two. We find the performance to be insensitive to the selection of dilation factor, where larger values mainly increase the number of processed samples.

Sky sampling: UniSim uses 16 samples for the sky field for each ray. We sample these linearly in disparity (one over distance to the sensor origin) between the end of the static field and $3\,\mathrm{km}$ away.

Sample merging: Each ray can generate a different number of sample points. To combine the results from the static, dynamic, and sky fields, we sort samples along the ray based on their distance and rely on nerfacc [5] for efficient rendering.

C.2. Model components

CNN: The CNN used for upsampling consists of four residual blocks with 32 channels. Further, a convolutional layer is applied at the beginning of the CNN to convert input features to 32 channels, and a second convolutional layer is applied to predict the RGB values. For both layers, we use kernel size one with no padding. We set the residual blocks to consist of convolution, batch norm, ReLU, convolution, batch norm, and skip connection to the input. The convolutional layers in the residual block use a kernel size of seven, with a padding of three. Between the second and third residual blocks, we apply a transposed convolution to upsample the feature map. We set the kernel size and stride to the upsample factor. The upsample factor in turn is set to three. Although we follow the specifications of UniSim, we find our implementation to have fewer parameters than what they report (0.7M compared to 1.7M). Likely reasons are interpretations of residual block design (only kernel size and padding a specified), kernel size for the first and last convolution layer, and the design of the upsampling layer. Nonetheless, we found that increasing the CNN parameter count only increased run-time without performance gains.

GAN: UniSim deploys an adversarial training scheme, where a CNN discriminator is trained to distinguish between rendered image patches at observed and unobserved viewpoints, where unobserved viewpoints refer to jittering the camera origins. The neural feature field and upsampling CNN are then trained to improve the photorealism at unobserved viewpoints. UniSim results show adversarial training to hurt novel view synthesis metrics (PSNR, LPIPS, SSIM), but boost FID performance for the lane-shift setting.

Unfortunately, the discriminator design is only briefly described in terms of a parameter count, resulting in a large potential design space. As training is done on patches, we opted for a PatchGAN [4] discriminator design inspired by pix2pixhd [10]. However, we found it difficult to get consistent performance increases and hence removed the adversarial training from our reimplementation. This is likely the reason for our reimplementation to perform slightly worse than the original results in terms of FID for lane shift. However, using adversarial training does not seem to be necessary in general for achieving low FID scores. In Tab. 3, we see NeuRAD, which does not use any GAN training, outperforming the original UniSim method, which does rely on adversarial supervision.

SDF to occupancy mapping: UniSim approximates the mapping from signed distance s to occupancy α as

$$\alpha = \frac{1}{1+e^{\beta s}}, \tag{1}$$

where β is a hyperparameter. As β is unspecified, we follow [14], which uses a similar formulation for neural rendering in an automotive setting. Specifically, we initialize β to 20.0 and make it a learnable parameter to avoid sensitivity to its specific value.

C.3. Supervision

Loss hyperparameters: We set $\lambda_{\text{rgb}} = 1.0$ and $\lambda_{\text{vgg}} = 0.05$. All other learning weights are given in UniSim's supplementary material and hence are used as is.

Regularization loss: For lidar rays, UniSim uses two regularizing losses. The first decreases the weights for samples far from any surface and the second encourages the signed distance function to satisfy the eikonal equation close to any surface

$$\mathcal{L}_{\text{reg}} = \frac{1}{N}\sum_{i=1}^{N}\Bigg(\sum_{\gamma_{i,j}>\epsilon} ||w_{ij}||_2 + \sum_{\gamma_{i,j}<\epsilon} (||\nabla s(\mathbf{x}_{ij})|| - 1)^2\Bigg). \tag{2}$$

Here, i is the ray index, j is the index for a sample $\mathbf{x}_{ij}$ along the ray, $\gamma_{i,j}$ denotes the distance between the sample and the corresponding lidar observation, *i.e.*, $\gamma_{i,j} = |\tau_{ij} - D_i^{\text{gt}}|$. We set $\epsilon = 0.1$.

Furthermore, we rely on tiny-cuda-nn [7] for fast implementations of the hash grid and MLPs. However, the framework does not support second-order derivatives for MLPs, and hence cannot be used when backpropagating through the SDF gradient $\nabla s(\mathbf{x}_{ij})$. Hence, instead of analytical gradients, we use numerical ones. Let

$$\begin{bmatrix}\mathbf{k}_1\\ \mathbf{k}_2\\ \mathbf{k}_3\\ \mathbf{k}_4\end{bmatrix} = \begin{bmatrix}1 & -1 & -1\\ -1 & -1 & 1\\ -1 & 1 & -1\\ 1 & 1 & 1\end{bmatrix}. \tag{3}$$

To find $\nabla s(\mathbf{x}_{ij})$, we query the neural feature field at four locations $\mathbf{x}_{ij} + \delta\mathbf{k}_l, l = 1,2,3,4$ where $\delta = \frac{0.01}{\sqrt{3}}$, resulting in four signed distance values s_1, s_2, s_3, s_4. Finally, we calculate

$$\nabla s(\mathbf{x}_{ij}) = \frac{1}{4\delta}\sum_l s_l \mathbf{k}_l. \tag{4}$$

The use of numerical gradients instead of analytical ones has been shown to be beneficial for learning signed distance functions for neural rendering [6].

Perceptual loss: Just like NeuRAD, UniSim uses a perceptual loss where VGG features of a ground truth image patch are compared to a rendered patch. While multiple formulations of such a loss exist, we opted for the one used in pix2pixHD [10] for both methods.

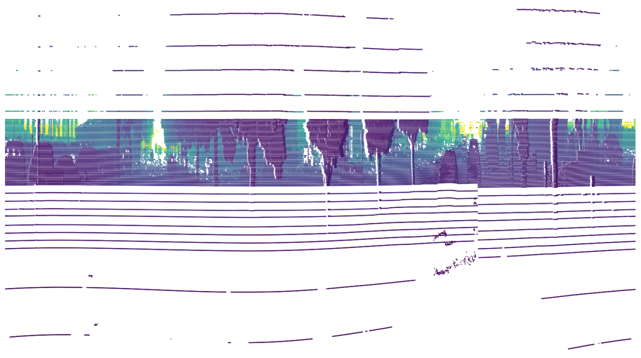
D. Inferring ray drop

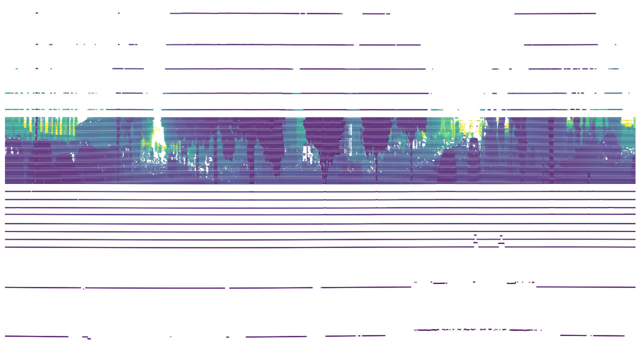
The inclusion of dropped lidar rays during supervision increases the fidelity of sensor renderings in all aspects, as shown in Tab. 4. The process of inferring which lidar beams are missing in a point cloud differs somewhat between datasets, as they contain different types of information. However, in general, the process consists of three steps: removal of ego-motion compensation, diode index assignment, and point infilling. In Fig. 5, we show a lidar scan from PandaSet [12] (sequence `106`) at different stages.

Removal of ego-motion compensation: To figure out which points are missing in a single sweep, we want to express their location in terms of azimuth (horizontal angle), elevation (vertical angle), and range at the time the beam was shot from the sensor. However, for all datasets, the provided points have been ego-motion compensated, *i.e.*, their Cartesian coordinates are expressed in a common coordinate frame. Simply converting them to spherical coordinates is therefore not possible until the ego-motion compensation is removed.

For each 3D lidar point (x, y, z) captured at time t we first project the point into world coordinates using its assigned sensor pose. For PandaSet [12], this first step is omitted, as points are provided in world coordinates. We then find the pose of the lidar sensor at time t by linearly interpolating existing sensor poses. For rotation, we use a quaternion representation and spherical linear interpolation (slerp). Given the sensor pose at t, we project the 3D point back into the sensor frame. We note that this process is susceptible to noise, since lidar poses are typically provided at a low frequency 10 Hz-20 Hz. We find elevation ϕ, azimuth θ and range r as

$$r = \sqrt{x^2 + y^2 + z^2}, \tag{5}$$
$$\phi = \arcsin(z/r), \tag{6}$$
$$\theta = \arctan(y/x). \tag{7}$$


(a) Before removing ego-motion compensation.


(b) After removing ego-motion compensation.

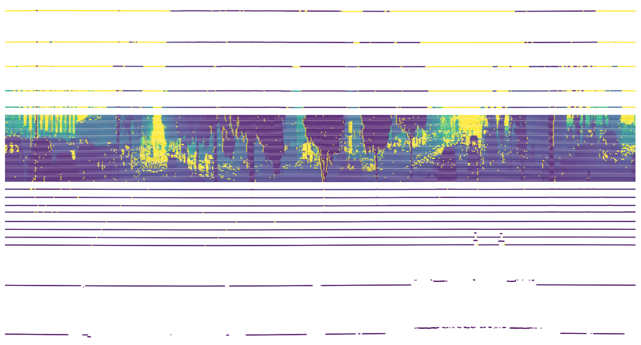

(c) After removing ego-motion compensation and adding missing points.

Figure 5. Lidar scans in spherical coordinates at different stages during inference of missing lidar rays. The color indicates range, where missing points have been set to a large distance for visualization purposes. Note that we do not add missing points for the two bottom rows, as they typically hit the ego vehicle.

Diode index assigment: All datasets considered in this work use spinning lidars, where a set of diodes are rotated 360° around the sensor and each diode is mounted at a fixed elevation angle. Typically, all diodes (or channels) transmit the same number of beams each revolution, where the number depends on the sensors' horizontal resolution. To use this information for inferring missing rays, we need to assign each return to its diode index. For most datasets considered here [1, 3, 11], this information is present in the raw data. However, for the other [12], we must predict diode assignment based on the points' elevation. As there is no ground truth available for this information, we use qualitative inspections to verify the correctness of the procedures outlined below.

PandaSet uses a spinning lidar with a non-linear elevation distribution for the diodes, *i.e.*, diodes are not spaced equally along the elevation axis. Instead, a few channels, the ones with the largest and smallest elevations, have a longer distance from their closest diode neighbor. Points corresponding to these channels are easily found by using sensor specifications. The remaining diodes use equal spacing, but inaccuracies in the removal of ego-motion compensation result in many wrongful diode assignments if sensor specifications are trusted blindly. Thus, we devise a clustering algorithm for inferring diode indices for points originating from diodes within the equal elevation distribution range.

The following is done separately for each lidar scan. First, we define the expected upper and lower bounds for elevation for each diode. These decision boundaries are spaced equally between the lowest and highest observed elevations based on the number of diodes. Then, we use histogram binning to cluster points based on their elevation. We use 2,000 bins, and the resulting bin widths are smaller than the spacing between diodes. Next, we identify consecutive empty bins. For any *expected* decision boundary that falls into an empty bin, we mark it as a *true* decision boundary. The same is true if the expected decision boundary is within $0.03°$ of an empty bin. Following this, if the number of true decision boundaries is smaller than the number of expected decision boundaries, we insert new boundaries between existing ones. Specifically, for the two boundaries with the largest distance between them, we insert as many boundaries as the vertical resolution dictates, but at least one, and at most as many decision boundaries that are missing. This insertion of boundaries is repeated until the required number of boundaries is reached.

Point infilling: After removing ego-motion compensation, transforming the points to spherical coordinates (elevation, azimuth, range), and finding their diode index, we can infer which laser rays did not return any points. Separately, for each diode, we define azimuth bins, spanning $0°$ to $360°$ with a bin width equal to the horizontal resolution of the lidar. If a returning point falls into a bin, we mark it as returned. For the remaining bins, we calculate their azimuth and elevation by linear interpolation.

E. Modeling rolling shutter

As shown in Fig. 3 and Tab. 4, modeling the rolling shutter improves generated renderings, especially at high velocities. Fig. 6 further shows the effects of rolling shutter on an ego-motion compensated lidar point cloud. To capture these effects, we assign each ray an individual timestamp. For lidar, these timestamps are typically available in the raw data, else we approximate them based on the rays' azimuth and the sensors' RPM. For cameras, individual timestamps are not available in the data. Instead, we manually approximate the shutter time and offset each image row accordingly. Given the individual timestamps, we linearly interpolate sensor poses to these times, effectively shifting the origin of the rays. Moreover, we model that dynamic actors may move during the capture time. Given the timestamps, we linearly interpolate their poses to the said time before transforming ray samples to the actors' coordinate systems.

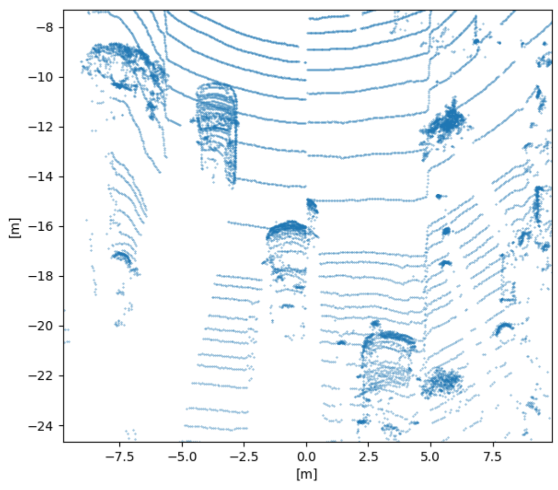


Figure 6. Bird's-eye-view of ego-motion compensated point cloud. Cuts in the circular patterns on the ground indicate the distance traveled by the ego-vehicle during one lidar revolution. Further, the cut through the car shows the importance of interpolating actor poses to the time when each lidar ray was shot.

F. Simulation gap

In the following, we show results for the simulation gap. To study the real2sim gap, we train the 3D object detector BEVFormer on real images from PandaSet-360 and evaluate its object detection performance on synthesized images from the ten sequences used for NVS (not part of the training set for BEVFormer). For BEVFormer, we use the official implementation[2] and the small version of the model. In Tab. 6 we see that the detector achieves similar validation performance for the real images and the synthesized images from NeuRAD. UniSim, struggling in the 360 setting, exhibits a larger gap.

Further, for a more general and dataset-agnostic evaluation, we use a zero-shot depth estimator, DepthAnything

[2]https://github.com/fundamentalvision/BEVFormer

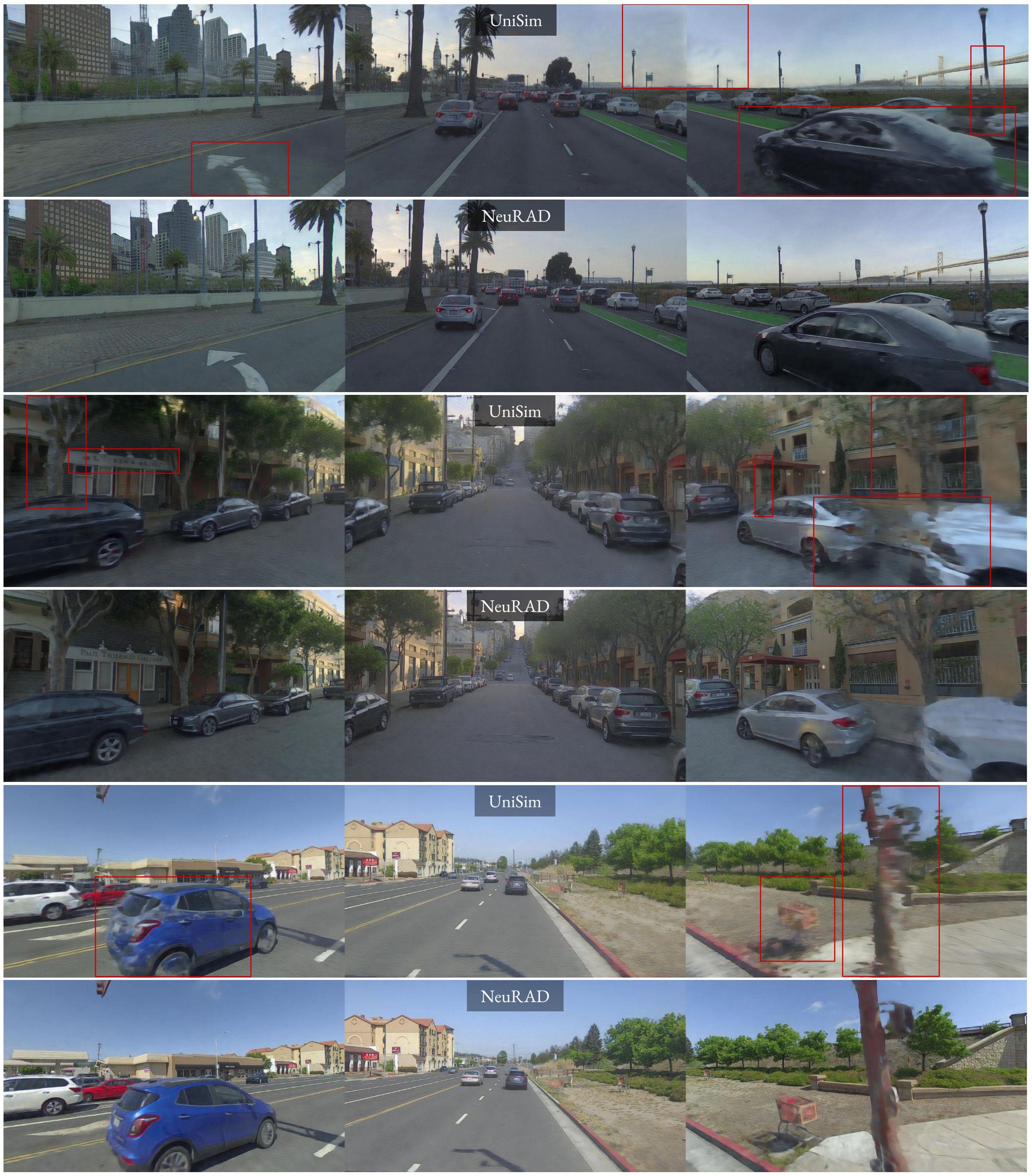


Figure 7. Qualitative comparison between NeuRAD and UniSim across three Pandaset sequences (016, 028, 158). Displayed are the front left, front center, and front right camera perspectives. NeuRAD overall captures more details than UniSim, although the difference is not dramatic for the front camera. However, as highlighted by red boxes, NeuRAD clearly outperforms UniSim for side cameras.

Table 6. Real2Sim gap: BEVFormer (mAP) on different images.

Data source	NeuRAD	UniSim*	Real
mAP	**32.0**	30.1	32.4

Table 7. Real2Sim gap: DepthAnything relative depth ($\delta_1 \uparrow$).

	PandaFC	Panda360	AV2	ZOD	NuScenes	KITTI
UniSim*	0.927	0.872	0.860	0.901	-	-
Neurad	**0.968**	**0.944**	**0.928**	**0.958**	**0.894**	**0.947**

(DA). We measure the agreement between depth estimations on synthesized and real images using the standard δ_1 metric. Tab. 7 shows consistent depth estimations, indicating a low domain gap across several datasets. For reference, DA reports $\delta_1 = 0.947$ when comparing against ground truth depth. We find these studies to give valuable insights and will include them in the manuscript.

G. Additional results

In the following, we provide additional results and insights, as well as some failure cases of our method.

Comparison with UniSim: We begin with a direct qualitative comparison between NeuRAD and UniSim [13] as depicted in Fig. 7. For the front camera, the distinction in quality is subtle but observable; NeuRAD demonstrates superior image clarity, exhibiting notably less noise and artifact presence. In contrast, the disparity in quality is more pronounced with the side cameras. Here, NeuRAD's output markedly surpasses UniSim, which is particularly evident in the highlighted areas where UniSim exhibits significant motion blur and visual distortion that NeuRAD effectively mitigates.

Proposal sampling: To efficiently allocate samples along each ray, we use two rounds of proposal sampling. For comparison, UniSim instead samples along the rays uniformly and relies on a lidar-based occupancy grid to prune samples far from the detected surfaces. Although the occupancy grid is fast to evaluate, it has two shortcomings. First, the method struggles with surfaces far from any lidar points. In the case of UniSim, the RGB values must instead be captured by the sky field, effectively placing the geometry far away regardless of its true position. The upper row of Fig. 8 shows an example of this, where a utility pole becomes very blurry without proposal sampling. Second, uniform sampling is not well suited for recovering thin structures or fine details of close-up surfaces. Doing so would require drawing samples very densely, which, instead, scales poorly with computational requirements. We examine both failure cases in Fig. 8, with thin power lines in the upper row and close-ups of vehicles in the lower row.

Sensor embedding: As described in Sec. 3.3, and shown in Tab. 4, the effect of different camera settings for different sensors in the same scene has a significant impact on reconstruction results. Fig. 9 shows qualitative results of this effect. Ignoring this effect causes shifts in color and lighting, often at the edge of images where the overlap between sensors is bigger, and is clearly visible in the second column of Fig. 9. Including sensor embeddings allows the model to account for differences in the sensors (e.g., different exposure), resulting in more accurate reconstructions.

Camera optimization: Neural rendering is reliant on access to accurate sensor poses. For instance, a small translation or rotation of a camera in world coordinates might translate to a small shift in the image plane as well, but this can drastically change each pixels' value.

In this work, we rely on sensor poses provided in the datasets, which typically are the result of IMU and GPS sensor fusion, SLAM, or a combination of both. As a result, sensor poses are often accurate to centimeter precision. While nuScenes [3] follows this example, the dataset does not provide height, roll, or pitch information, as this information has been discarded. We found this to be a limiting factor for the performance of NeuRAD, especially for sequences where the ego vehicle does not traverse a simple, flat surface. To address this, we instead enable optimization of the sensor poses, similar to how we optimize the poses of dynamic actors, see Sec. 3.3.

Applying sensor pose optimization qualitatively results in sharp renderings and quantitatively yields strong FID scores, see Tab. 3. However, we found novel view synthesis performance – in terms of PSNR, LPIPS and SSIM – to drop sharply. We find that the reason is that the sensor pose optimization creates an inconsistency between the world frame of the training data and the validation poses. Due to noisy validation poses, we render the world from a slightly incorrect position, resulting in large errors for the NVS per-pixel metrics. We illustrate this in Fig. 10, where the image from the training without sensor pose optimization is more blurry, but receives higher PSNR scores than the one with pose optimization.

We explored multiple methods for circumventing these issues, including separate training runs for finding accurate training and validation poses, or optimizing only the poses of validation images post-training. However, to avoid giving NeuRAD an unfair advantage over prior work, we simply disabled sensor pose optimization for our method. Nonetheless, we hope to study the issue of NVS evaluation when accurate poses are not available for neither training or validation in future work.

G.1. Limitations

In this work, we have proposed multiple modeling strategies for capturing important phenomena present in automotive data. Nonetheless, NeuRAD builds upon a set of assumptions, which when violated, result in suboptimal performance. Here, we cover some of these failure cases.

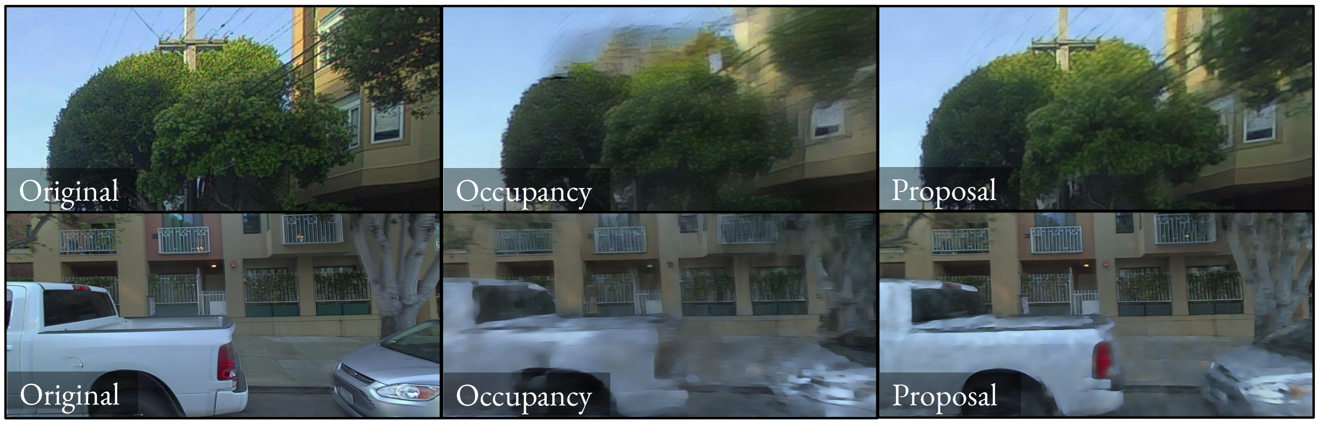


Figure 8. Two failure-cases that demonstrate the importance of proposal sampling over occupancy-based sampling: regions without lidar occupancy that are improperly modeled by sky field (upper), and nearby object that require extremely dense sampling (lower).

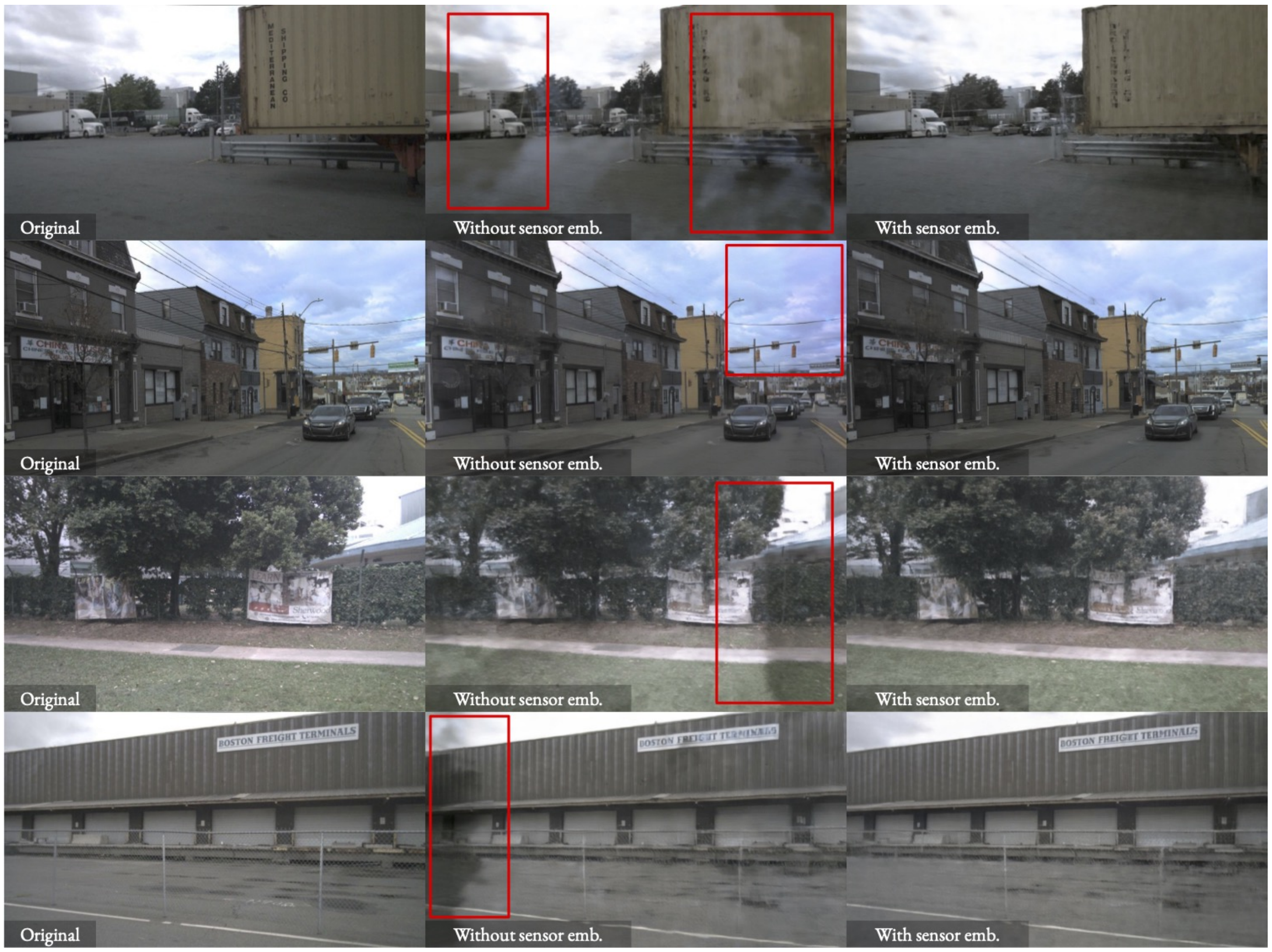


Figure 9. Effect of sensor embedding. The second column shows rendered images from the model trained without sensor embeddings, where a clear degradation is visible due to the shift in appearance (e.g., different exposure) between different sensors. As can be seen in the third column, this effect is remedied by including sensor embeddings.

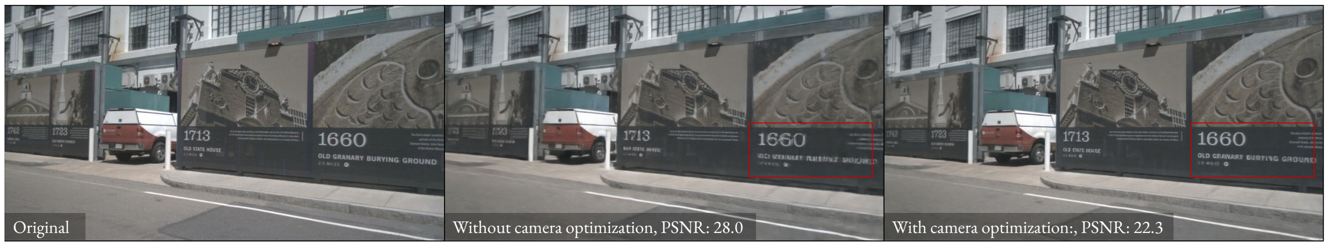

Original
Without camera optimization, PSNR: 28.0
With camera optimization:, PSNR: 22.3
1713
1660
OLD GRANARY BURYING GROUND

Figure 10. Effect of camera optimization on nuScenes. Despite clearly sharper image quality, we get drastically reduced PSNR scores when using camera optimization. This is due to the misalignment between the learned poses and the evaluation poses. This can be seen in the far left of the image, where the image with camera optimization displays less of a window.

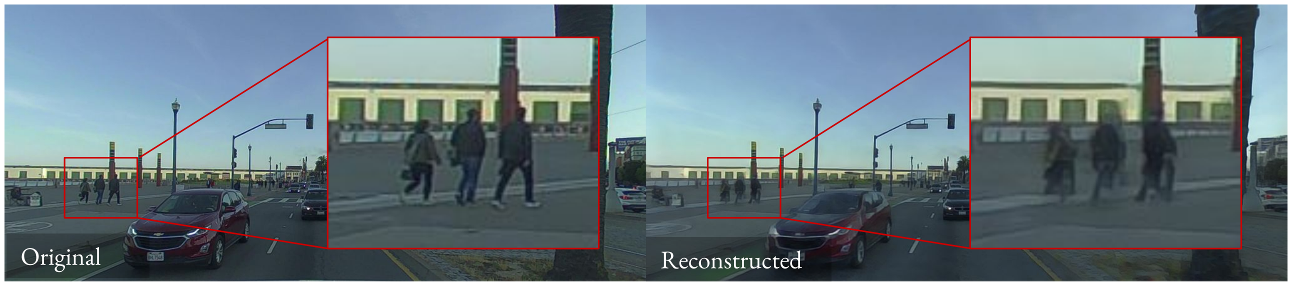


Figure 11. Failed reconstruction of deformable actors. The assumption that all actors are rigid is invalid for pedestrians and the like, leading to blurry reconstruction as seen here.

Deformable dynamic actors: When modeling dynamic actors, we make one very strong assumption — that the dynamics of an actor can be described by a single rigid transform. This is a reasonable approximation for many types of actors, such as cars, trucks, and to a lesser degree even cyclists. However, pedestrians break this assumption entirely, leading to very blurry reconstructions, as can be seen in Fig. 11.

Night scenes: Modelling night scenes with NeRF-like methods can be quite tricky for several reasons. First, night images contain a lot more measurement noise, which hinders the optimization as it is not really related to the underlying geometry. Second, long exposure times, coupled with the motion of both the sensor and other actors, lead to blurriness and can even make thin objects appear transparent. Third, strong lights produce blooming and lens-flare, which have to be explained by placing large blobs of density where there should not be any. Finally, dynamic actors, including the ego-vehicle, frequently produce their own illumination, such as from headlights. While static illumination can usually be explained as an effect dependent on the viewing direction, this kind of time-varying illumination is not modelled at all.

Time-dependent object appearance: In order to build a fully-useable closed-loop simulation we need to model brake lights, turning indicators, traffic lights, etc. While the problem is similar to that of deformable actors, it differs in some ways. First, we do not require the geometry to vary over time, potentially simplifying the problem. Second, we can probably treat these appearances as a set of discrete states. Third, the current set of perception annotations/detections might not cover all necessary regions where this effect is present. For instance, most datasets do not explicitly annotate traffic lights. Finally, we require full control and editability for this effect, to the degree that we can enable brake lights for a car that never braked. For general deformable actors, we might be satisfied with reconstructing the observed deformation, without being able to significantly modify it.

References

[1] Mina Alibeigi, William Ljungbergh, Adam Tonderski, Georg Hess, Adam Lilja, Carl Lindström, Daria Motorniuk, Junsheng Fu, Jenny Widahl, and Christoffer Petersson. Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving. In *Int. Conf. Comput. Vis.*, pages 20178–20188, 2023. 4

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. In *Int. Conf. Comput. Vis.*, pages 19697–19705, 2023. 1

[3] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multi-

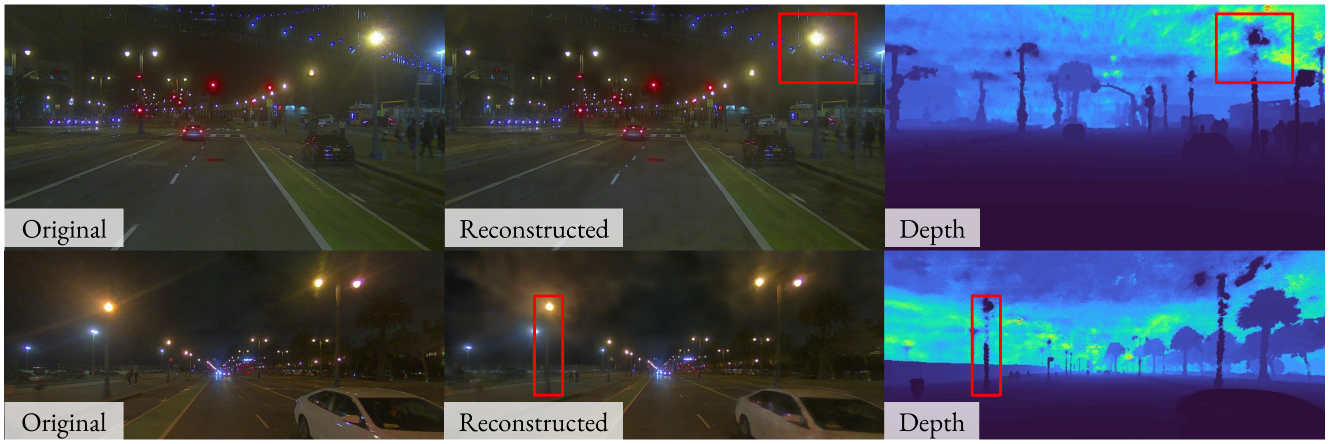


Figure 12. Novel view synthesis at night is challenging. For instance, strong lights can produce flares in the camera lens. These are hard to model with the standard NeRF rendering equations, as it requires the network to place density around the lights. Further, longer exposure times at night lead to dark, thin objects appearing semi-opaque, obscuring the learned scene geometry. Last, moving vehicles, including the ego-vehicle, illuminate the scene, resulting in a change of color over time for certain static parts of the scene. For instance, the road contains artifacts due to illumination from the ego-vehicle headlights.

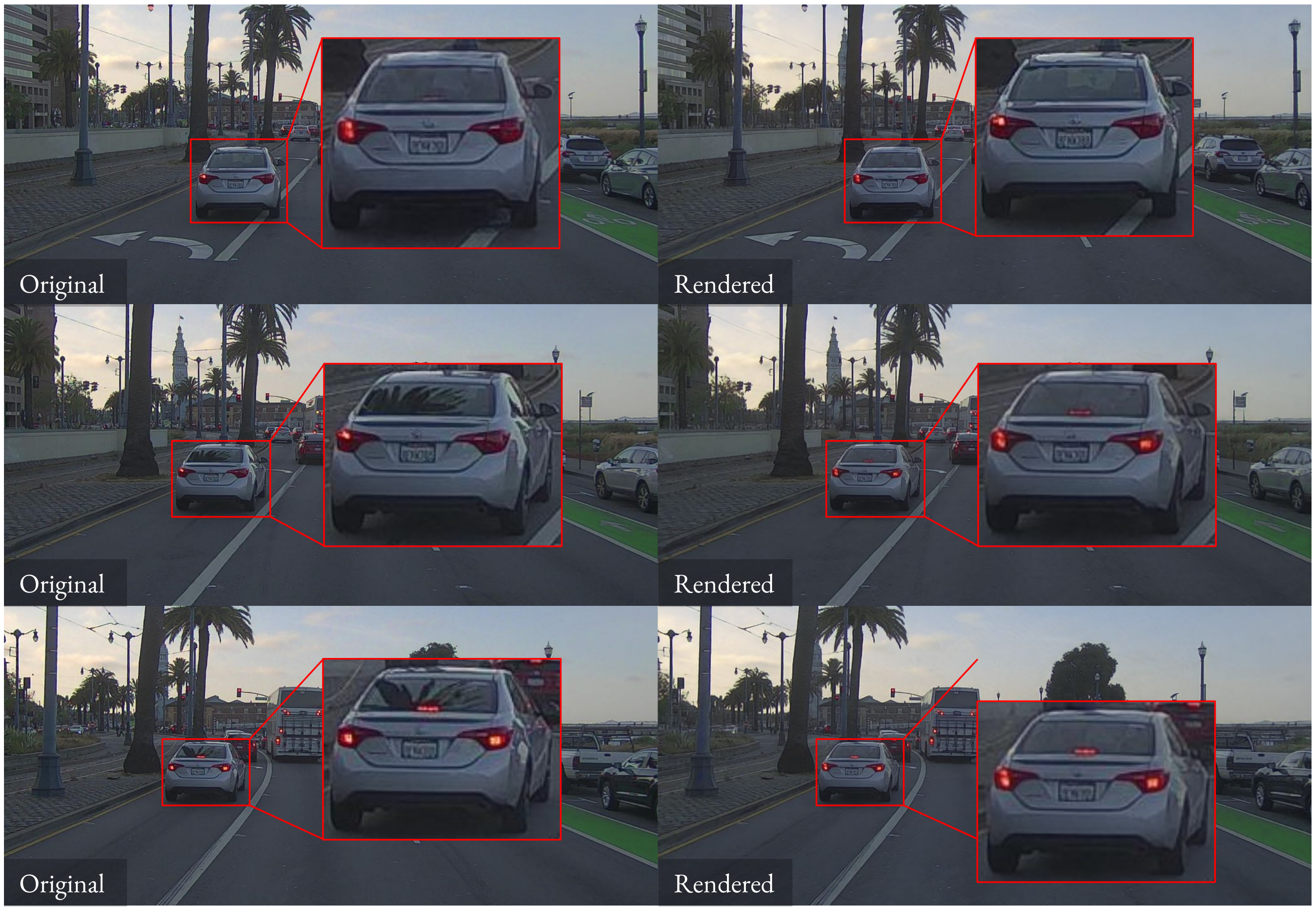


Figure 13. NeuRAD assumes all radiance to be static over time, even for dynamic actors. Thus, our method cannot express changes in light conditions, such as brake lights highlighted here. Interestingly, the model compensates by making the brake lights a function of the viewing angle instead, as the two are correlated in this particular scene.

modal dataset for autonomous driving. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 11621–11631, 2020. 4, 7

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 3

[5] Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. Nerfacc: Efficient sampling accelerates nerfs. *arXiv preprint arXiv:2305.04966*, 2023. 3

[6] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8456–8465, 2023. 4

[7] Thomas Müller. tiny-cuda-nn, 2021. 1, 3

[8] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, et al. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–12, 2023. 2

[9] Qitai Wang, Yuntao Chen, Ziqi Pang, Naiyan Wang, and Zhaoxiang Zhang. Immortal tracker: Tracklet never dies. *arXiv preprint arXiv:2111.13672*, 2021. 2

[10] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018. 3, 4

[11] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*, 2021. 4

[12] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, Yunlong Wang, and Diange Yang. Pandaset: Advanced sensor suite dataset for autonomous driving. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3095–3101, 2021. 2, 4, 5

[13] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1389–1399, 2023. 1, 2, 7

[14] Ze Yang, Sivabalan Manivasagam, Yun Chen, Jingkang Wang, Rui Hu, and Raquel Urtasun. Reconstructing objects in-the-wild for realistic sensor simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11661–11668, 2023. 3