NeuRAD: Neural Rendering for Autonomous Driving

Adam Tonderski[†,1,4] Carl Lindström[†,1,2] Georg Hess[†,1,2]
William Ljungbergh[1,3] Lennart Svensson[2] Christoffer Petersson[1,2]
[1]Zenseact [2]Chalmers University of Technology [3]Linköping University [4]Lund University
{firstname.lastname}@{zenseact.com, chalmers.se}

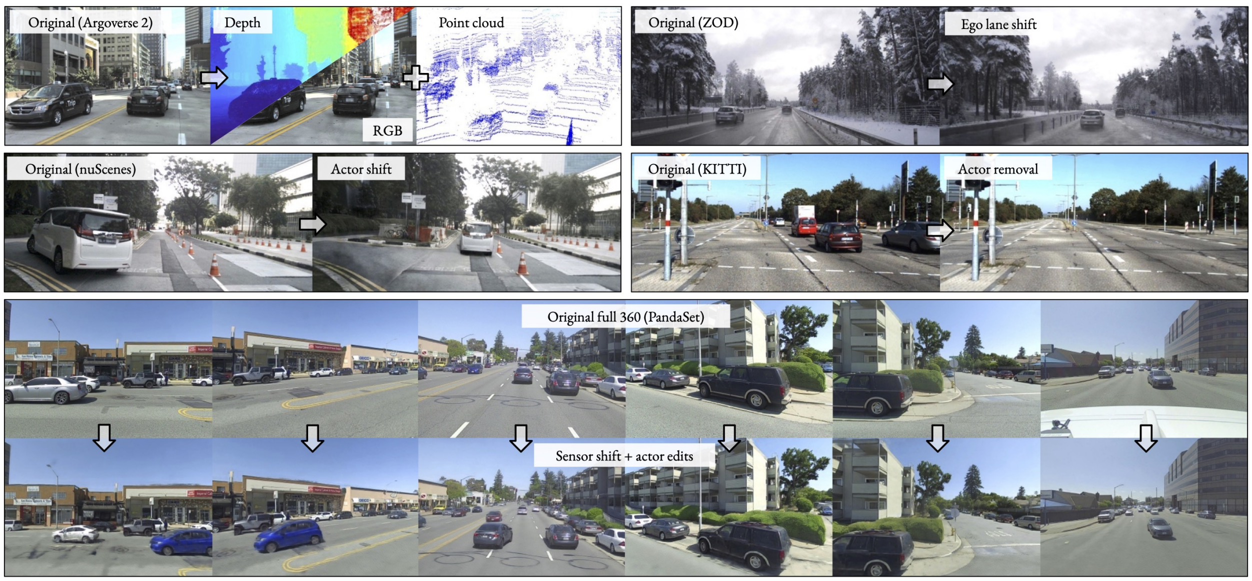


Figure 1. NeuRAD is a neural rendering method tailored to dynamic automotive scenes. With it, we can alter the pose of the ego vehicle and other road users as well as freely add and/or remove actors. These capabilities make NeuRAD suitable to serve as the foundation in components such as sensor-realistic closed-loop simulators or powerful data augmentation engines.

Abstract

Neural radiance fields (NeRFs) have gained popularity in the autonomous driving (AD) community. Recent methods show NeRFs' potential for closed-loop simulation, enabling testing of AD systems, and as an advanced training data augmentation technique. However, existing methods often require long training times, dense semantic supervision, or lack generalizability. This, in turn, hinders the application of NeRFs for AD at scale. In this paper, we propose NeuRAD, a robust novel view synthesis method tailored to dynamic AD data. Our method features simple network design, extensive sensor modeling for both camera and lidar – including rolling shutter, beam divergence and ray dropping – and is applicable to multiple datasets out of the box. We verify its performance on five popular AD datasets, achieving state-of-the-art performance across the board. To encourage further development, we openly release the NeuRAD source code.

1. Introduction

In Neural Radiance Fields (NeRFs) a model is trained to learn a 3D representation from which sensor realistic data can be rendered from new viewpoints [23]. Such techniques have been shown to be useful for a multitude of applications, such as view synthesis [3], generative modeling [11], or pose and shape estimation [37].

Autonomous Driving (AD) is a field where NeRFs may become very useful. By creating editable digital clones of traffic scenes, safety-critical scenarios can be explored in a scalable manner and without risking physical damage. For example, practitioners can investigate the behavior of the system for harsh braking on a highway or aggressive merging in city traffic. Furthermore, a NeRF-powered closed-loop simulator can be used for the targeted generation of corner-case training data.

Multiple works have applied NeRFs to automotive data [17, 26, 29, 31, 39, 41, 42]. Neural Scene Graphs [26]

[†]These authors contributed equally to this work.

extend the original NeRF model [23] to dynamic automotive sequences by dividing the scene into static background and a set of rigid dynamic actors with known location and extent, learning separate NeRFs for each. This enables editing the trajectories of both the ego-vehicle and all actors in the scene. The approach can be further improved by including semantic segmentation [17] or by using anti-aliased positional embeddings [41]. The latter enables NeRFs to reason about scale [3] which is essential for large-scale scenes. However, common for all these approaches is that they require many hours of training, limiting their applicability for scalable closed-loop simulation or data augmentation.

More recent works [39, 42] rely on Instant NGP's (iNGP) [24] learnable hash grids for embedding positional information, drastically reducing training and inference time. Further, these methods generate realistic renderings in their respective settings, namely front-facing camera with 360° lidar. However, their performance in 360° multicamera settings, which is common in many AD datsets [5, 38], is either unexplored [39] or is reported by the authors to be suboptimal [42]. Furthermore, both methods deploy simple lidar models and cannot model ray drop, a phenomenon important for closing the real-to-sim gap [19]. Lastly, using the iNGP positional embedding without anti-aliasing techniques limits performance, especially for larger scenes [4].

In this paper, we present NeuRAD, an editable novel view synthesis (NVS) method, designed to handle large-scale automotive scenes and to work well with multiple datasets off the shelf. We find that modeling sensor characteristics, such as rolling shutter, lidar ray dropping, and beam divergence, is essential for sensor-realistic renderings. Further, our model features a simple network architecture, where static and dynamic elements are discerned only by their positional embeddings, making it a natural extension of recent methods to AD data. We verify NeuRAD's generalizability and achieve state-of-the-art performance across five automotive datasets, with no dataset-specific tuning.

Our contributions are as follows. **(1)** Our method is the first to combine lidar sensor modeling with the ability to handle 360° camera rigs in a unified way, extending the applicability of NeRF-based methods for dynamic AD data. **(2)** We propose using a single network to model dynamic scenes, where dynamics and statics are separated only by their positional embeddings. **(3)** We propose simple, yet effective methods for modeling multiple key sensor characteristics such as rolling shutter, beam divergence, and ray dropping, and highlight their effect on performance. **(4)** Extensive evaluation using five popular AD datasets shows that our method is state-of-the-art across the board.

2. Related work

NeRFs: Neural radiance fields [23] is a novel view synthesis method in which a neural network learns an implicit 3D representation from which new images can be rendered. Multiple works [6, 8, 15, 24] address the long training time of the original formulation. Notably, Instant-NGP (iNGP) [24] uses a multiresolution, learnable hash grid to encode positional information rather than NeRFs frequency-based encoding scheme. A different line of work [2–4, 13] focuses on reducing aliasing effects by embedding pixel frustums instead of extent-free points, where Zip-NeRF [4] combines the anti-aliasing properties of mip-NeRF 360 [3] with the fast hash grid embedding of iNGP [24] by using multisampling and downweighting. Although these works were designed for static scenes and cannot be applied to dynamic sequences, we draw inspiration from Zip-NeRF's anti-aliasing techniques to better model large scenes.

NeRFs for automotive data: Accurately simulating data for AD systems is a promising avenue for efficient testing and verification of self-driving vehicles. While game-engine-based methods [7, 28] have made a lot of progress, they struggle with scalable asset creation, real-to-sim gap, and diversity. NeRFs' sensor-realistic renderings offer an attractive alternative, and consequently, multiple works have studied how to apply neural rendering techniques to automotive data. NSG [26], Panoptic Neural Fields (PNF) [17] and Panoptic NeRF [9] all model the background and every actor as multi-layer perceptrons (MLPs), but struggle with large-scale scenes due to the MLPs limited expressiveness. S-NeRF [41] extends mip-NeRF 360 to automotive data similar to NSG by modeling each actor with a separate MLP, but requires day-long training, making it impractical for downstream applications. Block-NeRF [29] and SUDS [31] both focus on city-scale reconstruction. While handling impressive scale, Block-NeRF filters out dynamic objects and only models static backgrounds, and SUDS uses a single network for dynamic actors, removing the possibility of altering actor behavior.

NeRFs for closed-loop simulation: Among existing work, two methods [39, 42] are the most similar to ours. MARS [39] proposes a modular design where practitioners can mix and match existing NeRF-based methods for rendering dynamic actors and the static background. Similar to our work, the implementation is based on Nerfstudio [30] to promote open-source collaboration. Unlike our work, MARS does not natively support lidar point clouds but relies on dense depth maps from either depth completion or mono-depth networks, limiting the ease of application to any dataset. Further, while MARS' semantic segmentation supervision is optional, performance deteriorates when this supervision is not available, especially on real-world data.

UniSim [42] is a neural sensor simulator, showcasing realistic renderings for PandaSet's [40] front camera and 360° lidar. The method applies separate hash grid features [24] for modeling the sky, the static background, and each dynamic actor, and uses NSG-style [26] transformations for handling dynamics. For efficiency, the static background is only sampled near lidar points. Further, UniSim renders features from the neural field, rather than RGB, and uses a convolutional neural network (CNN) for upsampling the features and producing the final image. This allows them to reduce the number of sampled rays per image significantly. While efficient, multiple approximations lead to poor performance outside their evaluation protocol. In addition, the lidar occupancy has a limited vertical field of view and fails to capture tall, nearby structures which often becomes evident when using cameras with alternative mounting positions or wider lenses, *e.g.*, nuScenes [5], Argoverse2 [38] or Zenseact Open Dataset (ZOD) [1]. In contrast, our method unifies static and sky modeling and relies on proposal sampling [4] for modeling occupancy anywhere. Further, UniSim's upsampling CNN introduces severe aliasing and model inconsistencies, as camera rays must describe entire RGB patches whereas lidar rays are thin laser beams. In this work, we introduce a novel anti-aliasing strategy that improves performance, with minimal impact on runtime.

3. Method

Our goal is to learn a representation from which we can generate realistic sensor data where we can change either the pose of the ego vehicle platform, the actors, or both. We assume access to data collected by a moving platform, consisting of posed camera images and lidar point clouds, as well as estimates of the size and pose of any moving actors. To be practically useful, our method needs to perform well in terms of reconstruction error on any major automotive dataset, while keeping training and inference times to a minimum. To this end, we propose NeuRAD, an editable, open source, and performant neural rendering approach; see Fig. 2 for an overview.

In the following, we first describe the underlying scene representation and sensor modeling. Next, we cover the internals of our neural field and the decomposition of sequences into static background and dynamic actors. We then present the unique challenges and opportunities of applying neural rendering to AD data and how we address them. Last, we discuss learning strategies.

3.1. Scene representation and sensor modeling

Neural scene rendering: Building on the recent advancements in novel view synthesis [4, 42], we model the world with a neural feature field (NFF), a generalization of NeRFs [23] and similar methods [21]. Given a position $\mathbf{x}$, and a view direction $\mathbf{d}$, an NFF outputs an implicit geometry s and a feature vector $\mathbf{f}$ [42]. The NFF, akin to a NeRF, is utilized for volumetric rendering. However, it accumulates implicit geometry and features rather than density and color [23].

To extract features for a ray $\mathbf{r}(\tau) = \mathbf{o} + \tau\mathbf{d}$, originating from the sensor center $\mathbf{o}$ and extending in direction $\mathbf{d}$, we sample $N_\mathbf{r}$ points along the ray in 3D space. The feature descriptors of these samples are aggregated using traditional alpha compositing:

$$\mathbf{f}(\mathbf{r}) = \sum_{i=1}^{N_\mathbf{r}} w_i \mathbf{f}_i, \quad w_i = \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j). \tag{1}$$

Here, α_i represents the opacity at the point $\mathbf{x}_i = \mathbf{o} + \tau_i\mathbf{d}$, and w_i the opacity times the accumulated transmittance along the ray up to $\mathbf{x}_i$. Inspired by its success in recovering high-quality geometry [18, 25], we represent the implicit geometry using a signed distance function (SDF) and approximate the opacity as $\alpha_i = 1/(1+e^{\beta s_i})$, where s_i is the SDF value at $\mathbf{x}_i$ and β is a learnable parameter. While more accurate SDF formulations [32, 34] can provide better performance, they require gradient calculations for each 3D point, negatively impacting the runtime.

Camera modeling: To render an image, we volume render a set of camera rays, generating a feature map $\mathcal{F} \in \mathbb{R}^{H_f \times W_f \times N_f}$. As in [42], we then rely on a CNN to render the final image $\mathcal{I} \in \mathbb{R}^{H_\mathcal{I} \times W_\mathcal{I} \times 3}$. In practice, the feature map has a lower resolution $H_f \times W_f$ than the image $H_I \times W_I$, and we use the CNN for upsampling. This allows us to drastically reduce the number of queried rays.

Lidar modeling: Lidar sensors allow self-driving vehicles to measure the depth and the reflectivity (intensity) of a discrete set of points. They do so by emitting laser beam pulses and measuring the time of flight to determine distance and returning power for reflectivity. To capture these properties, we model the transmitted pulses from a posed lidar sensor as a set of rays and use volume rendering similar to (1). For a lidar point, we shoot a ray $\mathbf{r}(\tau) = \mathbf{o} + \tau\mathbf{d}$, where $\mathbf{o}$ is the origin of the lidar and $\mathbf{d}$ is the normalized direction of the beam. We then find the expected depth D_l of a ray as $\mathbb{E}[D_l(\mathbf{r})] = \sum_{i=1}^{N_\mathbf{r}} w_i \tau_i$. For predicting intensity, we volume render the ray feature following (1) and pass the feature through a small MLP.

In contrast to previous works incorporating lidar measurements [27, 42], we also include rays for laser beams which did not return any points. This phenomenon, known as ray dropping, occurs if the return power has too low amplitude, and is important to model for reducing the sim-to-real gap [19]. Typically, such rays travel far without hitting a surface, or hit surfaces from which the beam bounces off into empty space, *e.g.*, mirrors, glass, or wet road surfaces. Modeling these effects is important for sensor-realistic simulations, but as noted in [14], are hard to capture fully physics-based because they depend on (often undisclosed) details of the low-level sensor detection logic. Therefore, we opt to learn ray dropping from data. Similar to the intensity, we use the rendered ray feature from (1) and pass it through a small MLP to predict the ray drop probability

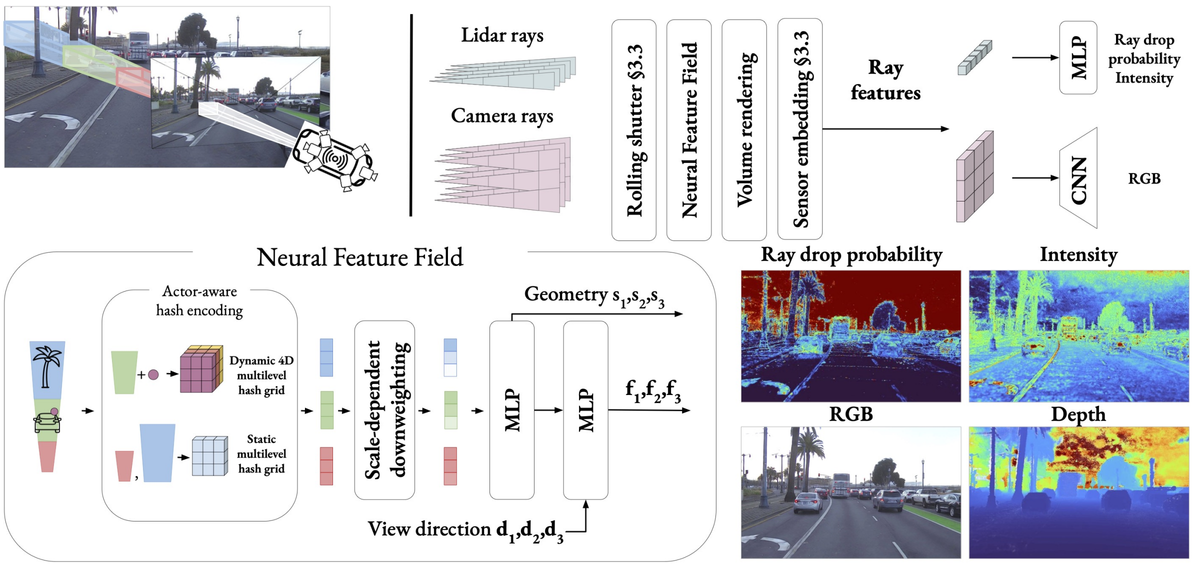


Figure 2. Overview of our approach. We learn a joint neural feature field for the statics and dynamics of an automotive scene, where the two are discerned only by our actor-aware hash encoding. Points that fall inside actor bounding boxes are transformed to actor-local coordinates and, together with actor index, used to query the 4D hash grid. We decode the volume rendered ray-level features to RGB values using an upsampling CNN, and to ray drop probability and intensity using MLPs.

$p_d(\mathbf{r})$. Note that unlike [14], we do not model second returns from lidar beams, as this information is not present in the five datasets considered here.

3.2. Extending Neural Feature Fields

In this section, we delve into the specifics of our volumetric scene representation. We begin by extending the Neural Feature Field (NFF) definition to be a learned function $(s, \mathbf{f}) = \text{NFF}(\mathbf{x}, t, \mathbf{d})$, where $\mathbf{x} \in \mathbb{R}^3$ are the spatial coordinates, $t \in \mathbb{R}$ represents time, and $\mathbf{d} \in \mathbb{R}^3$ indicates the view direction. Importantly, this definition introduces time as an input, which is essential for modeling the dynamic aspects of the scene.

Architecture: Our NFF architecture adheres to well-established best practices in the NeRF literature [4, 24]. Given a position $\mathbf{x}$ and time t we query our *actor-aware hash encoding*. This encoding then feeds into a small Multilayer Perceptron (MLP), which computes the signed distance s and an intermediate feature $\mathbf{g} \in \mathbb{R}^{N_g}$. The view direction $\mathbf{d}$ is encoded using spherical harmonics [24], allowing the model to capture reflections and other view-dependent effects. Finally, the direction encoding and $\mathbf{g}$ are jointly processed through a second MLP, augmented with a skip connection from $\mathbf{g}$, producing the feature $\mathbf{f}$.

Scene composition: Similar to previous works [17, 26, 41, 42], we decompose the world into two parts, the static background and a set of rigid dynamic actors, each defined by a 3D bounding box and a set of SO(3) poses. This serves a dual purpose: it simplifies the learning process, and it allows a degree of editability, where actors can be moved after training to generate novel scenarios. Unlike previous methods which utilize separate NFFs for different scene elements, we employ a single, unified NFF, where all networks are shared, and the differentiation between static and dynamic components is transparently handled by our actor-aware hash encoding. The encoding strategy is straightforward: depending on whether a given sample $(\mathbf{x}, t)$ lies inside an actor bounding box, we encode it using one of two functions.

Unbounded static scene: We represent the static scene with a multiresolution hash grid [24], as this has been proven to be a highly expressive and efficient representation. However, to map our unbounded scenes onto a grid, we employ the contraction approach proposed in MipNerf-360 [3]. This allows us to accurately represent both nearby road elements and far-away clouds, with a single hash grid. In contrast, prior automotive approaches utilize a dedicated NFF to capture the sky and other far-away regions [42].

Rigid dynamic actors: When a sample $(\mathbf{x}, t)$ falls within the bounding box of an actor, its spatial coordinates $\mathbf{x}$ and view directions $\mathbf{d}$ are transformed to the actor's coordinate frame at the given time t. This allows us to ignore the time aspect after that, and sample features from a time-independent multiresolution hash grid, just like for the static scene. Naively, we would need to separately sample multiple different hash grids, one for each actor. However, we instead utilize a single 4D hash grid, where the fourth dimension corresponds to the actor index. This novel approach allows us to sample all actor features in parallel, achieving significant speedups while matching the performance of using separate hash grids.

3.3. Automotive data modeling

Multiscale scenes: One of the biggest challenges in applying neural rendering to automotive data is handling the multiple levels of detail present in this data. As vehicles cover large distances, many surfaces are visible both from afar and close up. Applying iNGP's [24] or NeRF's position embedding naively in these multiscale settings results in aliasing artifacts as they lack a sense at which scale a certain point is observed [2]. To address this, many approaches model rays as conical frustums, the extent of which is determined longitudinally by the size of the bin and radially by the pixel area in conjunction with distance to the sensor [2, 3, 13]. Zip-NeRF [4], which is currently the only anti-aliasing approach for iNGP's hash grids, combines two techniques for modeling frustums: multisampling and downweighting. In multisampling, the positional embeddings of multiple locations in the frustum are averaged, capturing both longitudinal and radial extent. For downweighting, each sample is modeled as an isotropic Gaussian, and grid features are weighted proportional to the fraction between their cell size and the Gaussian variance, effectively suppressing finer resolutions. While the combined techniques significantly increase performance, the multisampling also drastically increases run-time.

Here, we aim to incorporate scale information with minimal run-time impact. Inspired by Zip-NeRF, we propose an intuitive downweighting scheme where we downweight hash grid features based on their size relative to the frustum. Rather than using Gaussians, we model each ray $\mathbf{r}(\tau) = \mathbf{o} + \tau\mathbf{d}$ as a pyramid with cross-sectional area $A(\tau) = \dot{r}_h\dot{r}_v\tau^2$, where $\dot{r}_h, \dot{r}_v$ are horizontal and vertical beam divergence based on the image patch size or the beam divergence of the lidar beam. Then, for a frustum defined by the interval $[\tau_i, \tau_{i+1})$, where A_i and A_{i+1} are the cross-sectional areas at the end-points τ_i and τ_{i+1}, we calculate its volume as

$$V_i = \frac{\tau_{i+1} - \tau_i}{3}\left(A_i + \sqrt{A_i A_{i+1}} + A_{i+1}\right), \quad (2)$$

and retrieve its positional embedding $\mathbf{e}_i$ at the 3D point $\mathbf{x}_i = \mathbf{o} + \frac{\tau_i + \tau_{i+1}}{2}\mathbf{d}$. Finally, for a hash grid at level l with resolution n_l we weight the position embedding $\mathbf{e}_{i,l}$ with $\omega_{i,l} = \mathtt{min}(1, (\frac{1}{n_l V_i^{1/3}}))$, *i.e.*, the fraction between the cell size and the frustum size.

Efficient Sampling: Another difficulty with rendering large-scale scenes is the need for an efficient sampling strategy. In a single image, we might want to render detailed text on a nearby traffic sign while also capturing parallax effects between skyscrapers several kilometers away. Uniformly sampling the ray to achieve both of these goals would require thousands of samples per ray which is computationally infeasible. Previous works have relied heavily on lidar data for pruning samples [42], and as a result struggle to render outside the lidar's field-of-view.

Instead, we draw samples along rays according to a power function [4], such that the space between samples increases with the distance from the ray origin. Even so, we find it impossible to fulfill all relevant conditions without prohibitively increasing the number of samples. Therefore, we also employ two rounds of proposal sampling [23], where a lightweight version of our NFF is queried to generate a weight distribution along the ray. Then, a new set of samples are drawn according to these weights. After two rounds of this procedure, we are left with a refined set of samples that focus on the relevant locations along the ray and that we can use to query our full-size NFF. To supervise the proposal networks, we adopt an anti-aliased online distillation method [4] and further use the lidar for supervision, see $\mathcal{L}^{\mathrm{d}}$ and $\mathcal{L}^{\mathrm{w}}$ introduced in Sec. 3.4.

Modeling rolling shutter: In standard NeRF-based formulations, each image is assumed to be captured from a single origin $\mathbf{o}$. However, many camera sensors have rolling shutters, *i.e.*, pixel rows are captured sequentially. Thus, the camera sensor can move between the capturing of the first row and that of the last row, breaking the single origin assumption. Although not an issue for synthetic data [22] or data captured with slow-moving handheld cameras, the rolling shutter becomes evident with captures from fast-moving vehicles, especially for side-cameras. The same effect is also present in lidars, where each scan is typically collected over $0.1\,\mathrm{s}$, which corresponds to several meters when traveling at highway speeds. Even for ego-motion compensated point clouds, these differences can lead to detrimental line-of-sight errors where 3D points translate to rays that cut through other geometries. To mitigate these effects, we model the rolling shutters by assigning individual times to each ray and adjusting their origin according to the estimated motion. As the rolling shutter affects all dynamic elements of the scene, we linearly interpolate actor poses to each individual ray time. See Appendix E for details.

Differing camera settings: Another problem when modeling autonomous driving sequences is that images come from different cameras with potentially different capture parameters, such as exposure. Here we draw inspiration from research on "NeRFs in the wild" [20], where an appearance embedding is learned for each image, and passed to the second MLP together with $\mathbf{g}$. However, as we know which image comes from which sensor, we instead learn a single embeddings per sensor, minimizing the potential for overfitting, and allowing us to use these *sensor embeddings* when generating novel views. As we render features rather than color, we apply these embeddings after the volume rendering, significantly reducing computational overhead.

Noisy actor poses: Our model relies on estimates of poses for dynamic actors, either in the form of annotations or as tracking output. To account for imperfections, we include

(a) original (b) no modeling (c) modeling lidar only (d) modeling lidar + camera

Figure 3. Impact of modeling rolling shutter in a high-speed scenario (with inset PSNR). (a) original side-camera image. Omitting the rolling shutter entirely (b) results in extremely blurry renderings and unrealistic geometry, especially for the pole. Modeling the lidar rolling shutter (c) improves the quality, but it is only when both sensors are modeled correctly (d) that we get realistic renderings.

the actor poses as learnable parameters in the model, and optimize them jointly. The poses are parameterized as a translation $\mathbf{t} \in \mathbb{R}^3$ and a rotation for which we use a 6D-representation [44].

3.4. Losses

We optimize all model components jointly and use both camera and lidar observations as supervision $\mathcal{L} = \mathcal{L}^{\text{image}} + \mathcal{L}^{\text{lidar}}$. In the following, we discuss the different optimization objectives in more detail.

Image losses: The image loss is computed patch-wise and summed over N_p patches and consists of a reconstruction term $\mathcal{L}^{\text{rgb}}$ and a perceptual term $\mathcal{L}^{\text{vgg}}$:

$$\mathcal{L}^{\text{image}} = \frac{1}{N_p} \sum_{i=1}^{N_p} \lambda^{\text{rgb}} \mathcal{L}_i^{\text{rgb}} + \lambda^{\text{vgg}} \mathcal{L}_i^{\text{vgg}}. \tag{3}$$

The reconstruction loss is the squared error between predicted and true pixel values. The perceptual loss is the distance between VGG features for real and predicted patches [33]. λ^{rgb} and λ^{vgg} are weighting hyperparameters.

Lidar losses: We incorporate the strong geometric prior given by the lidar by adding a depth loss for lidar rays and employing weight decay to penalize density in empty space. Further, to be able to simulate a more realistic lidar we also include objectives for the predicted intensity and the predicted ray drop probability:

$$\mathcal{L}^{\text{lidar}} = \frac{1}{N} \sum_{i=1}^{N} (\lambda^{\text{d}} \mathcal{L}_i^{\text{d}} + \lambda^{\text{int}} \mathcal{L}_i^{\text{int}} + \lambda^{p_d} \mathcal{L}_i^{p_d} + \lambda^{\text{w}} \mathcal{L}_i^{\text{w}}), \tag{4}$$

where λ^{d}, λ^{int}, λ^{p_d}, and λ^{w} are hyperparameters. The depth loss $\mathcal{L}_i^{\text{d}}$ and the intensity loss $\mathcal{L}_i^{\text{int}}$ are the squared error between the prediction and the observation. For dropped rays, we penalize estimates only below the specified sensor range, and do not supervise intensity. For the ray drop probability loss, $\mathcal{L}_i^{p_d}$, we use a binary cross entropy loss. The weight decay is applied for all samples outside of a distance ϵ of the lidar observation:

$$\mathcal{L}_i^{\text{w}} = \sum_{\tau_{i,j} > \epsilon} \left\| w_{ij} \right\|_2, \tag{5}$$

where $\tau_{i,j}$ is the distance from sample $\mathbf{x}_{ij}$ to the lidar observation for ray i. For dropped rays, weight decay is applied up until the specified sensor range. Noteably, we omit the commonly used eikonal loss, as it provided minimal benefits at a high computational cost.

3.5. Implementation details

NeuRAD is implemented in the collaborative, open-source project Nerfstudio [30]. We hope that our developed supporting structures such as data loaders and native lidar support will encourage further research into this area. We train our main method (NeuRAD) for 20,000 iterations using the Adam [16] optimizer. Using a single Nvidia A100, training takes about 1 hour. To showcase the scalability of our approach, we also design a larger model with longer training (NeuRAD-2x). See Appendix A for further details.

4. Experiments

To verify the robustness of our model, we evaluate its performance on several popular AD datasets: nuScenes [5], PandaSet [40], Argoverse 2 [38], KITTI [10], and ZOD [1]. To prove the robustness of our method we use the same model and hyperparameters on all datasets. We investigate novel view synthesis performance both for hold-out validation images and for sensor poses without any ground truth. Furthermore, we ablate important model components. More results, including a study on the real2sim gap as well as failure cases can be found in Appendix F and Appendix G.

4.1. Datasets and baselines

Below, we introduce the datasets used for evaluation. The selected datasets cover various sensors, and the included sequences contain different seasons, lighting conditions, and driving conditions. Existing works typically use one or two datasets for evaluation and build models around assumptions about available supervision, limiting their applicability to new settings. Therefore, for each dataset, we compare our model to SoTA methods that have previously adopted said dataset, and follow their respective evaluation proto-

Table 1. Image novel view synthesis performance comparison to state-of-the-art methods across five datasets. *our reimplementation. †baselines from [39, 41, 42]. §partial results due to training instability. **Bold**/underline for best/second-best.

		PSNR ↑	SSIM ↑	LPIPS ↓
Panda FC	Instant-NGP† [24, 42]	24.03	0.708	0.451
	UniSim [42]	25.63	0.745	0.288
	UniSim*	25.44	0.732	0.228
	NeuRAD (ours)	26.58	0.778	0.190
	NeuRAD-2x (ours)	**26.84**	**0.801**	**0.148**
Panda 360	UniSim*	23.50	0.692	0.330
	NeuRAD (ours)	25.97	0.758	0.242
	NeuRAD-2x (ours)	**26.47**	**0.779**	**0.196**
nuScenes	Mip360† [3, 41]	24.37	0.795	0.240
	S-NeRF [41]	26.21	**0.831**	0.228
	NeuRAD (ours)	26.99	0.815	0.225
	NeuRAD-2x (ours)	**27.13**	0.820	**0.205**
KITTI MOT	SUDS† [31, 39]	23.12	0.821	0.135
	MARS [39]	24.00	0.801	0.164
	NeuRAD (ours)	27.00	0.795	0.082
	NeuRAD-2x (ours)	**27.91**	**0.822**	**0.066**
Argo2	UniSim*	23.22§	0.661§	0.412§
	NeuRAD (ours)	26.22	0.717	0.315
	NeuRAD-2x (ours)	**27.73**	**0.756**	**0.233**
ZOD	UniSim*	27.97	0.777	0.239
	NeuRAD (ours)	29.49	0.809	0.226
	NeuRAD-2x (ours)	**30.59**	**0.857**	**0.210**

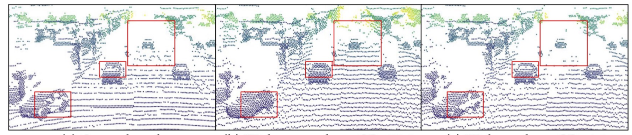
cols. Similar to our method, UniSim [42] imposes few supervision assumptions, and we, therefore, reimplement the method (denoted Unisim*) and use it as a baseline for datasets where no prior work exists. See Appendix C for reimplementation details and Appendix B for further evaluation details.

PandaSet: We compare our method to UniSim [42] and an iNGP version with lidar depth supervision provided by UniSim. We use every other frame for training and the remaining ones for testing, and evaluate on the same 10 scenes as UniSim. We study two settings: one with lidar and front-facing camera (Panda FC) for direct comparison with the results reported in [42], and one with lidar and all six cameras capturing the full 360° field-of-view around the vehicle (Panda 360). We also evaluate UniSim on the full 360° setting using our reimplementation.

nuScenes: We compare our method to S-NeRF [41] and Mip-NeRF 360 [3]. We follow S-NeRF's protocol, *i.e.*, select 40 consecutive samples halfway into the sequences and use every fourth for evaluation while every other among the remaining ones is used for training. We test on the same four sequences as S-NeRF, using the same sensor setup.

KITTI: For KITTI [10], we compare our method to MARS [39]. We use MARS 50% evaluation protocol, *i.e.*, evaluating on every second image from the right camera and using the left and right camera and lidar from remaining time instances for training.

Argo 2 & ZOD: To verify the robustness of our method, we study two additional datasets, Argoverse 2 [38] and



(a) Ground truth (b) Without ray drop, CD: 22.6 (c) With ray drop, CD: 4.1

Figure 4. Visualization of ray drop effects for lidar simulation. Highlighted parts show areas where ray dropping effects are important to consider in order to simulate realistic point clouds. CD denotes Chamfer distance normalized by num. GT points.

Table 2. Lidar novel view synthesis performance comparison to state-of-the-art methods. Depth is median L2 error [m]. Intensity is RMSE. Drop acc. denotes ray drop accuracy. Chamfer denotes chamfer distance, normalized with num. ground truth points [m].

		Depth ↓	Intensity ↓	Drop acc. ↑	Chamfer ↓
Panda FC	UniSim	0.10	0.065	-	-
	UniSim*	0.07	0.085	91.0	11.2
	NeuRAD (ours)	**0.01**	**0.062**	**96.2**	**1.6**
Panda 360	UniSim*	0.07	0.087	91.9	10.3
	NeuRAD (ours)	**0.01**	**0.061**	**96.1**	**1.9**

ZOD [1]. Due to the lack of prior work supporting dynamic actors on these datasets, we compare NeuRAD to our UniSim implementation. For each dataset, we train on every other frame, test on the remaining frames, and evaluate on ten sequences. As ZOD does not have any sequence annotations, we use a 3D-object detector and an off-the-shelf tracker to generate pseudo-annotations for the sequences.

4.2. Novel view synthesis

Camera: We report the standard NVS metrics PSNR, SSIM [35] and LPIPS [43], for all datasets and baselines in Tab. 1. NeuRAD achieves SoTA performance across all datasets. On PandaSet, we improve upon previous work across all metrics, for both FC and 360. On nuScenes, NeuRAD matches the performance of S-NeRF while training much faster (1 hour compared to 17 hours). NeuRAD also outperforms previous SoTA on KITTI with a large margin in terms of PSNR and LPIPS. Finally, NeuRAD also achieves strong performance on Argoverse 2 and ZOD.

Lidar: We measure the realism of our lidar simulation in terms of L2 median depth error, RMSE intensity error and ray drop accuracy. We complement the depth error with the Chamfer distance as it enables us to evaluate performance on dropped rays as well. We compare only to UniSim, evaluated on PandaSet, as no other baseline simulates point clouds. UniSim has no notion of ray dropping, hence we assume rays to be dropped past the reported lidar range. We see in Tab. 2 that NeuRAD decreases the depth error by an order of magnitude compared to UniSim in the front-camera setting. Our method generalizes well to the 360° setting, where similar results are reported. Furthermore, we show that NeuRAD is capable of simulating realistic point clouds, thanks to its high ray drop accuracy and low Chamfer distance. Fig. 4 further shows the importance of mod-

Table 3. FID scores when shifting pose of ego vehicle or actors.

		Ego shift				Actor shift	
		No shift	Lane 2m	Lane 3m	Vert. 1m	Rot.	Trans.
Panda FC	UniSim	-	74.7	97.5	-	-	-
	UniSim*	41.7	79.6	102.0	89.3	65.5	59.6
	NeuRAD	**25.0**	**72.3**	**93.9**	**76.3**	**64.3**	**49.1**
Panda 360	UniSim*	88.3	115.5	128.0	126.7	95.9	93.0
	NeuRAD	45.5	84.0	98.8	91.3	58.8	55.4
	NeuRAD w/ opt	**43.0**	**81.0**	**95.3**	**88.8**	**56.7**	**53.0**

eling ray drop effects for lidar simulation. As noted in the figure, lidar beams that hit the road far away tend to disperse and not return. Similar effects occur for transparent surfaces, such as the car window illustrated in the figure, where the lidar beams shoot right through. Modeling these effects can increase the realism of simulated point clouds.

4.3. Novel scenario generation

In order for our method to be useful in practice, it must not only perform well when interpolating between views, but also when exploring new views, as examplified in Fig. 1. To that end, we investigate NeuRAD's capability to generate images from poses that are significantly different from those encountered during training. We adapt UniSim's protocol on PandaSet, *i.e.*, translating the ego vehicle sensors laterally two or three meters to simulate a lane shift, and extend the protocol to include one meter vertical shift, simulating other mounting positions. We further investigate "actor shift", and rotate (± 0.5 radians) or translate (± 2 meters laterally) dynamic actors in the scene to simulate different actor behaviors. As no ground truth images exist, we report FID [12], with "no shift" for reference. The results in Tab. 3 show that NeuRAD is able to generalize to new viewpoints and learns meaningful actor representations. We also include results where we optimize the camera poses following [36], as this further increases sharpness.

4.4. Ablations

We validate the effectiveness of some key components in Tab. 4. To avoid biases toward any specific dataset, we report averaged metrics from sequences from all five datasets considered in this work. We select 4 diverse sequences from each dataset, see details in Appendix B. Our full model corresponds to the model used in all prior experiments and strikes a good balance between run-time and performance. We see that the CNN decoder (a) significantly increases both quality and speed, by requiring significantly fewer rays and allowing for interaction between rays. Accurate sensor modeling is also very important, as each of our contributions in that area provide complementary performance boost: considering rolling shutter (b) or lidar rays that did not return (e), modeling each ray as a frustum (c) and per-sensor appearance embeddings (d). We also demonstrate that replacing individual actor hash grids with a single 4D hash grid (f) has no detrimental impact on quality, while significantly increasing training speed. Finally, we replace our SDF with a NeRF-like density formulation (g). The performance is overall almost identical and shows that our model can be configured to either of these field representations depending on the need. If we desire to extract surfaces from our model, we can use an SDF, but if our scenes are dominated by fog, transparent surfaces, or other effects where an SDF breaks down, we can fall back to a density formulation. Interestingly, our ablations only show a modest impact of considering rolling shutter. However, upon closer inspection of the qualitative results, see Fig. 3, it is apparent that both the renderings and underlying geometry break down without considering this effect.

Table 4. Ablations when *removing* core parts of our model. We report NVS performance for images and lidars, scene generation, and training megapixels per second (MP/s). Results are averaged over 20 sequences, evenly split across all five datasets.

		PSNR ↑	LPIPS ↓	SSIM ↑	Depth ↓	Scen. gen. ↓	MP/s ↑
	Full model	27.26	0.213	0.786	0.030	75.5	1.9
a)	CNN decoder	25.29	0.329	0.720	0.107	127.9	0.2
b)	Rolling shutter	26.77	0.246	0.763	0.060	80.6	1.9
c)	Downweighting	26.12	0.283	0.741	0.146	100.6	2.0
d)	Appearance emb.	25.50	0.270	0.744	0.080	102.6	1.9
e)	Missing points	25.36	0.361	0.685	0.050	106.3	1.8
f)	4D actor grid	27.22	0.217	0.779	0.030	76.5	1.5
g)	SDF	27.37	0.211	0.790	0.029	75.5	1.9

5. Conclusions

In this paper, we have proposed NeuRAD, a neural simulator tailored specifically for dynamic autonomous driving (AD) data. The model jointly handles lidar and camera data in 360° and decomposes the world into its static and dynamic elements, allowing the creation of sensor-realistic editable clones of real world driving scenarios. NeuRAD incorporates novel modeling of various sensor phenomena including beam divergence, ray dropping, and rolling shutters, all increasing the quality during novel view synthesis. We demonstrate NeuRAD's efficacy and robustness by obtaining state-of-the-art performance on five publicly AD datasets, using a single set of hyperparameters. Lastly, we publicly release our source-code to foster more research into NeRFs for AD.

Limitations: NeuRAD assumes actors to be rigid and does not support any deformations. Further, many modeling assumptions are invalid for harsh weather like heavy rain or snow. We hope to address these limitations in future work.

Acknowledegments: We thank Maryam Fatemi for valuable feedback. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Computational resources were provided by NAISS at NSC Berzelius, partially funded by the Swedish Research Council, grant agreement no. 2022-06725.

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. 3, 6, 7

[2] 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 *Int. Conf. Comput. Vis.*, pages 5855–5864, 2021. 2, 5

[3] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5470–5479, 2022. 1, 2, 4, 5, 7

[4] 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. 2, 3, 4, 5

[5] 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 multimodal dataset for autonomous driving. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 11621–11631, 2020. 2, 3, 6

[6] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *Eur. Conf. Comput. Vis.*, pages 333–350. Springer, 2022. 2

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017. 2

[8] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 2

[9] 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 *2022 International Conference on 3D Vision (3DV)*, pages 1–11. IEEE, 2022. 2

[10] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 6, 7

[11] Ayaan Haque, Matthew Tancik, Alexei Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *Int. Conf. Comput. Vis.*, 2023. 1

[12] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Adv. Neural Inform. Process. Syst.*, 30, 2017. 8

[13] Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields. In *Int. Conf. Comput. Vis.*, pages 19774–19783, 2023. 2, 5

[14] Shengyu Huang, Zan Gojcic, Zian Wang, Francis Williams, Yoni Kasten, Sanja Fidler, Konrad Schindler, and Or Litany. Neural lidar fields for novel view synthesis. In *Int. Conf. Comput. Vis.*, 2023. 3, 4

[15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):1–14, 2023. 2

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 6

[17] Abhijit Kundu, Kyle Genova, Xiaoqi Yin, Alireza Fathi, Caroline Pantofaru, Leonidas J Guibas, Andrea Tagliasacchi, Frank Dellaert, and Thomas Funkhouser. Panoptic neural fields: A semantic object-aware neural scene representation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12871–12881, 2022. 1, 2, 4

[18] 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. 3

[19] Sivabalan Manivasagam, Ioan Andrei Bârsan, Jingkang Wang, Ze Yang, and Raquel Urtasun. Towards zero domain gap: A comprehensive study of realistic lidar simulation for autonomy testing. In *Int. Conf. Comput. Vis.*, pages 8272–8282, 2023. 2, 3

[20] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7210–7219, 2021. 5

[21] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4460–4470, 2019. 3

[22] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Trans. Graph.*, 38(4):1–14, 2019. 5

[23] 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 *Eur. Conf. Comput. Vis.*, pages 405–421, Cham, 2020. Springer International Publishing. 1, 2, 3, 5

[24] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):1–15, 2022. 2, 4, 5, 7

[25] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Int. Conf. Comput. Vis.*, pages 5589–5599, 2021. 3

[26] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2856–2865, 2021. 1, 2, 3, 4

[27] Konstantinos Rematas, Andrew Liu, Pratul P Srinivasan, Jonathan T Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12932–12942, 2022. 3

[28] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th International Conference*, pages 621–635. Springer, 2018. 2

[29] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8248–8258, 2022. 1, 2

[30] 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, 6

[31] Haithem Turki, Jason Y Zhang, Francesco Ferroni, and Deva Ramanan. Suds: Scalable urban dynamic scenes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12375–12385, 2023. 1, 2, 7

[32] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *Adv. Neural Inform. Process. Syst.*, pages 27171–27183, 2021. 3

[33] 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 *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018. 6

[34] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *Int. Conf. Comput. Vis.*, pages 3295–3306, 2023. 3

[35] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4): 600–612, 2004. 7

[36] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf–: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021. 8

[37] Bowen Wen, Jonathan Tremblay, Valts Blukis, Stephen Tyree, Thomas Müller, Alex Evans, Dieter Fox, Jan Kautz, and Stan Birchfield. Bundlesdf: Neural 6-dof tracking and 3d reconstruction of unknown objects. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 606–617, 2023. 1

[38] 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. 2, 3, 6, 7

[39] Zirui Wu, Tianyu Liu, Liyi Luo, Zhide Zhong, Jianteng Chen, Hongmin Xiao, Chao Hou, Haozhe Lou, Yuantao Chen, Runyi Yang, Yuxin Huang, Xiaoyu Ye, Zike Yan, Yongliang Shi, Yiyi Liao, and Hao Zhao. Mars: An instance-aware, modular and realistic simulator for autonomous driving. *CICAI*, 2023. 1, 2, 7

[40] 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, 6

[41] Ziyang Xie, Junge Zhang, Wenye Li, Feihu Zhang, and Li Zhang. S-neRF: Neural radiance fields for street views. In *The Eleventh International Conference on Learning Representations*, 2023. 1, 2, 4, 7

[42] 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, 3, 4, 5, 7

[43] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 586–595, 2018. 7

[44] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5745–5753, 2019. 6