

LiDAR-RT: Gaussian-based Ray Tracing for Dynamic LiDAR Re-simulation

Chenxu Zhou^{1*} Lvchang Fu^{2*} Sida Peng¹ Yunzhi Yan¹ Zhanhua Zhang³
Yong Chen³ Jiazhi Xia² Xiaowei Zhou¹

¹Zhejiang University

²Central South University

³Geely Automobile Research Institute

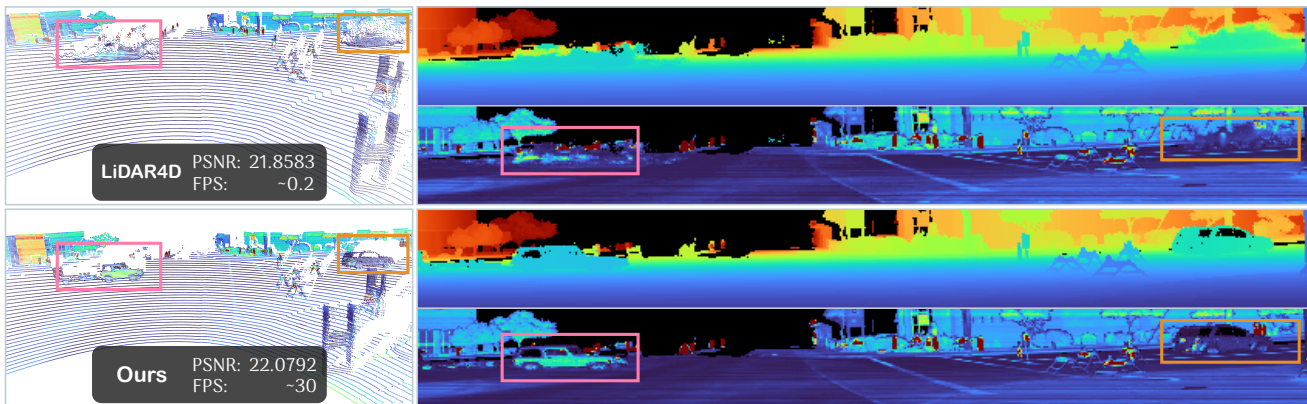


Figure 1. **Realistic and real-time rendering of LiDAR view in dynamic driving scenes.** Our LiDAR-RT produces high-fidelity LiDAR view at 30 FPS (64×2650) within 2 hours of training. SOTA method [58] struggles to model the dynamic objects in complex scenes and suffers from high training and rendering costs (15 hours for training and 0.2 FPS for rendering a range image).

Abstract

This paper targets the challenge of real-time LiDAR re-simulation in dynamic driving scenarios. Recent approaches utilize neural radiance fields combined with the physical modeling of LiDAR sensors to achieve high-fidelity re-simulation results. Unfortunately, these methods face limitations due to high computational demands in large-scale scenes and cannot perform real-time LiDAR rendering. To overcome these constraints, we propose LiDAR-RT, a novel framework that supports real-time, physically accurate LiDAR re-simulation for driving scenes. Our primary contribution is the development of an efficient and effective rendering pipeline, which integrates Gaussian primitives and hardware-accelerated ray tracing technology. Specifically, we model the physical properties of LiDAR sensors using Gaussian primitives with learnable parameters and incorporate scene graphs to handle scene dynamics. Building upon this scene representation, our framework first constructs a bounding volume hierarchy (BVH), then casts rays for each pixel

*The first two authors contributed equally.

and generates novel LiDAR views through a differentiable rendering algorithm. Importantly, our framework supports realistic rendering with flexible scene editing operations and various sensor configurations. Extensive experiments across multiple public benchmarks demonstrate that our method outperforms state-of-the-art methods in terms of rendering quality and efficiency. Our project page is at <https://zju3dv.github.io/lidar-rt>.

1. Introduction

Modeling dynamic urban scenes is of great importance for a variety of applications, such as digital twin simulation, autonomous driving, and virtual reality. Recent research has shown impressive results of novel view synthesis in driving scenarios [7, 8, 50, 59] with camera sensors, significantly enhancing the fidelity and diversity of data for downstream tasks and applications. However, the majority of current approaches overlook the re-simulation of LiDAR sensors, which are key components for many 3D perception algorithms, thereby restricting the scope of simulation applications. In contrast to cameras, the inherent physical properties of LiDAR sensors and sparsity of point clouds make it challenging to reconstruct and simulate, especially

in dynamic scenes with complicated motions.

Previous methods [19, 25] initially reconstruct 3D scenes from real-world data with explicit representations like dense point clouds or triangular meshes, which are utilized to render novel LiDAR views via ray-casting. Although these two-stage methods manage to produce acceptable results, they are hampered by geometric inaccuracies and struggle to model the physical properties of LiDAR sensors. Additionally, they are only capable of modeling static scenes and fail to handle dynamic objects.

As neural scene representations advance, there have been some works [15, 42, 49, 56] that attempt to synthesize novel views of LiDAR with neural radiance fields [27]. These NeRF-based methods [15, 42, 49, 56] combine the rendering power of neural fields with the physical modeling process of LiDAR sensors, achieving physically realistic LiDAR rendering. By introducing 4D hybrid features or tracked bounding boxes, some recent works [46, 58] extend these methods to handle dynamic scenes. However, these approaches are constrained by high training costs and slow rendering speeds due to the large amount of network parameters, and struggle to model complex dynamic scenes with long-distance vehicle motion and occlusions.

In this paper, we propose a novel framework, named LiDAR-RT, for novel LiDAR view synthesis of dynamic driving scenes. Our primary contribution is the development of an efficient and effective rendering pipeline for Gaussian-based [14, 16] LiDAR representation. Firstly, We enhance Gaussian primitives with additional learnable parameters to accurately model the physical characteristics of LiDAR sensors. Furthermore, to tackle the challenges of dynamic scenes, we integrate the scene graphs [29] into LiDAR representation, which provides flexible modeling capabilities under various environmental conditions.

Building upon proposed scene representation, we design a differentiable Gaussian-based ray tracer to simulate the physical formation of LiDAR sensors. Specifically, our method constructs the corresponding proxy geometries for Gaussian primitives and inserts them into a bounding volume hierarchy (BVH). To compute the LiDAR radiance, we cast rays for each pixel from the sensor against the BVH to determine the intersections and store the information in a sorted buffer [3]. Subsequently, we calculate the response of the intersected Gaussians and integrate the radiance along the ray via volumetric rendering techniques. This process continues until all Gaussian primitives have been traversed or the accumulated transmittance reaches a pre-defined threshold. Then the rendered properties are fused to generate novel LiDAR views. Compared to the vanilla tile-based Gaussian rasterizer [16], our method employs a physically accurate ray tracing process, which enhances both the realism and accuracy for LiDAR re-simulation. Thanks to the components we proposed above, our method

not only reconstructs high fidelity LiDAR point clouds and achieves real-time rendering of novel LiDAR views, but also supporting flexible manipulations of LiDAR sensors.

We evaluate the proposed method on Waymo Open (Waymo) [40] and KITTI-360 [20] datasets, which cover a wide range of complex dynamic scenes. Our method achieves state-of-the-art performance in terms of rendering quality and efficiency across all datasets. Moreover, our comprehensive experiments have confirmed that the components we proposed are highly adaptable for LiDAR re-simulation and maintain great performance under various editing manipulations.

2. Related work

LiDAR Simulation. Traditional self-driving simulators such as CARLA [9] or Airsim [39] do possess the ability to simulate LiDAR sensors, but suffer from the sim-to-real gap [21, 26] and costly manual effort in creating virtual assets. LiDARsim [25] uses a multi-stage and data-driven approach to reconstruct the surfel-based [34] scene from real-world data and model the ray-drop property via a neural network for improving realism. PCGen [19] utilizes the first peak averaging (FPA) ray-casting and surrogate model ray-drop to further close the domain gap. Nevertheless, these explicit methods [19, 25, 52] are sensitive to the geometry quality and only applicable to static scenes. By leveraging generative models, some works [6, 62, 63] provide an alternative but lack the physical control to model the real-world LiDAR sensors.

Several recent methods [15, 42, 49, 53, 56] are proposed for high quality novel view LiDAR synthesis based on neural radiance fields (NeRF) [27], which combines the differentiable rendering capability of NeRF and the physical model of LiDAR sensors. Notably, NFL [15] demonstrates the detailed physical modeling of LiDAR sensors firstly, and provides a comprehensive reference for following works. NeRF-LiDAR [56] and UniSim [53] take multimodal inputs to enhance the performance of LiDAR re-simulation. Furthermore, with the introduction of 4D hybrid features and additional tracking labels, some methods [41, 43, 46, 58] extend to dynamic driving scenes. However, all these NeRF-based approaches are constrained primarily by the high computational demands during training and rendering, and struggle to accurately model or re-simulate scenes with complex motions.

Dynamic Scene Reconstruction. Expanding on the accomplishments of NeRF, some methods [2, 31, 35] build 4D neural representations to reconstruct object-level scenes via various extensions, such as continuous deformation fields [30] or spatial-temporal encoding [11]. Some recent methods extend to dynamic urban scenes with extra supervision like optical flow [44, 48] and transformer

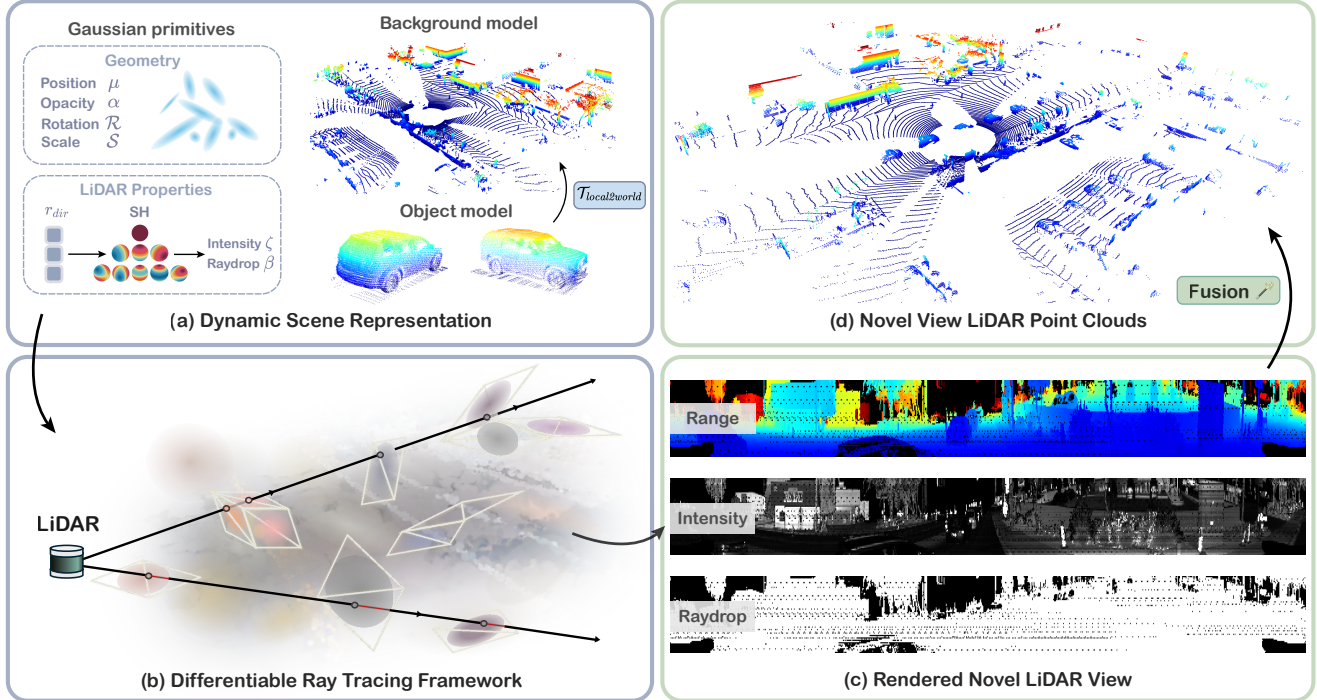


Figure 2. **Overview of LiDAR-RT.** (a) We decompose the dynamic scene into a background model and multiple object models, with each represented by a set of Gaussian primitives. In addition to geometric attributes, we introduce learnable parameters (SHs) on Gaussians to emulate the intrinsic properties (ζ, β) of LiDAR sensors. (b) Based on this representation, we design a differentiable ray tracing framework. We first construct the proxy geometry for Gaussian primitives and then cast rays from the sensor to perform intersection tests. (c) By evaluating the response from these intersections, we accumulate point-wise properties along each ray, and finally render the novel LiDAR view as range images. (d) The range images are fused and re-projected into LiDAR point clouds for downstream tasks.

features [51]. Apart from these works, another group of methods [29, 43, 47, 53] model the dynamic scene as the composition of dynamic objects and static background. UniSim [53] and NeuRAD [43] also simulate LiDAR observations at new views, which is similar to our work. Nevertheless, these methods suffer from high computational costs on large scale scenes and cannot perform real-time rendering. Aiming for rapid reconstruction and rendering, some methods prefer to represent the dynamic scene with 3D Gaussians [7, 50, 59, 60], which perform impressive results for novel view synthesis of camera sensors. OmniRe [8] further constructs multiple local canonical spaces to model diverse dynamic objects in a driving log, including non-rigid pedestrians and cyclists. Despite these advancements, there is a significant gap when they simulate other types of sensors like LiDAR, which is crucial for many perception and planning tasks.

Gaussian Splatting. 3D Gaussian Splatting [16, 23] made a breakthrough by using a differentiable splatting algorithm to render a scene represented by Gaussian primitives, many works conduct extensions to 3DGS, such as anti-aliasing [55] and view-consistent rendering [36]. Based on the 3DGS, 2D Gaussian Splatting [14] collapses the 3D volume into a set of 2D oriented planar Gaussian

disks and design a perspective-accurate 2D splatting process to model and reconstruct geometrically accurate radiance fields from multi-view images. However, these rasterization-based rendering is difficult to model general ray effects (shadows, reflections etc.) and image formation processes that exhibit non-linear optical models, which limits its application for many tasks. To address this issue, some approaches [5, 12, 24, 28] propose to combine the ray tracing technologies with Gaussian particles. 3DGRT [28] utilizes the high-performance GPU ray tracing hardware to render the particle scenes in real-time.

3. Method

In this section, we begin by delineating the problem formulation for the novel LiDAR view synthesis and the preliminary of Gaussian Splatting [16]. Following that, we provide a detailed exposition of our proposed LiDAR-RT.

Problem Formulation. In this problem, our input is a collection of LiDAR scans \mathcal{L} captured by a moving sensor within a driving scenario, along with the corresponding calibrated sensor poses P and timestamps T . Additionally, a group of bounding boxes \mathcal{B} are provided to track the moving vehicles in the scene.

Our goal is to reconstruct this dynamic scene as an explicit representation and render realistic LiDAR views from any given moment t_{new} and novel viewpoint P_{new} . Furthermore, rendering must maintain high fidelity for various scene manipulations and sensor configurations.

Gaussian Splatting. Kerbl et al. [16] propose to represent 3D scenes with 3D Gaussian primitives and render images using differentiable volume splatting. The i -th Gaussian is characterized by a full 3D covariance matrix Σ_i defined in world space [61] centered at point (mean) μ_i :

$$\mathcal{G}_i(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_i)^\top \Sigma_i^{-1}(\mathbf{x} - \mu_i)\right), \quad (1)$$

where the covariance matrix $\Sigma_i = \mathbf{R}_i \mathbf{S}_i \mathbf{S}_i^\top \mathbf{R}_i^\top$ is factorized into a scaling matrix \mathbf{S}_i and a rotation matrix \mathbf{R}_i . Each Gaussian is paired with an opacity value σ_i and the spherical harmonics coefficients (SHs) [37] to model the view-dependent appearance \mathbf{c}_i .

When rendering novel views, 3D Gaussians are rasterized to image planes and form 2D Gaussians \mathcal{G}' with the 2D covariance $\Sigma'_i = \mathbf{J} \mathbf{W} \Sigma_i \mathbf{W}^\top \mathbf{J}^\top$, where \mathbf{W} is the viewing transformation and \mathbf{J} is the Jacobian of the local affine transformation [61]. The color of a pixel is calculated via alpha blending with depth ordered 2D Gaussians:

$$\mathcal{C}(\mathbf{x}) = \sum_{i=1}^K T_i \alpha_i \mathbf{c}_i, \quad (2)$$

$$\text{where } \alpha_i = \sigma_i \mathcal{G}'_i(\mathbf{x}) \quad \text{and} \quad T_i = \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (3)$$

3.1. Overview of LiDAR-RT

Towards the target of rendering realistic novel LiDAR views in real-time, we propose a novel framework named LiDAR-RT. As illustrated in Fig. 2, we represent the dynamic driving scene as a set of point clouds consisting of a static background and multiple foreground vehicles, each associated with Gaussian primitives (Section 3.2). To ensure physical realism and high fidelity in rendering, we design an efficient and effective ray tracer based on Gaussian primitives to model the LiDAR imaging process (Section 3.3). Moreover, we implement the backward pass of ray tracing to support differentiable LiDAR rendering (Section 3.4) and scene optimization (Section 3.5).

3.2. Dynamic Scene Representation

We decompose the dynamic scene into a static background and several moving vehicles, with each component represented by a distinct set of Gaussian primitives to softly learn the continuous scene geometry and radiance. Similar to 3DGS [16], both the Gaussian primitives of background model and dynamic actor models share a set of common

base parameters, including the mean position μ , covariance matrix Σ , opacity σ , and SH coefficients, the covariance matrix Σ can further be factorized into a scaling matrix \mathbf{S} and a rotation matrix \mathbf{R} as in Eq. 1.

Additionally, we introduce learnable parameters on Gaussian primitives to emulate the intrinsic properties of LiDAR sensors, denoted by $\mathcal{P} = (\zeta, \beta)$, where $\zeta \in \mathbb{R}^1$ represents the reflection intensity and $\beta \in [0, 1]$ indicates the ray-drop probability. Ray-drop is a common phenomenon of real-world LiDAR sensors that occurs when the return signal is too weak [26], and the emitted ray is considered as dropped. Unlike previous methods that directly model the ray-drop with a single variable [15, 42, 46, 58], we learn the ray-drop probability β with two logits ($\beta_{drop}, \beta_{hit}$), activated by the softmax function [4]:

$$\beta = \frac{e^{\beta_{drop}}}{e^{\beta_{drop}} + e^{\beta_{hit}}}. \quad (4)$$

Since the intensity and ray-drop are strongly affected by view directions, our method models $(\zeta, \beta_{drop}, \beta_{hit})$ with a set of SH coefficients as shown in Fig. 2(a).

To handle the foreground moving vehicles, we utilize tracked bounding boxes to trace their motion trajectories. Different from the background model, the mean position μ_o and rotation matrix \mathbf{R}_o of the Gaussian primitives for dynamic objects are defined within the object local coordinate system. To transform them into the world coordinate system, same as [43, 50], we define the tracked poses for each rigid object as a series of rotation matrices $\{\mathbf{R}_t\}_{t=1}^{N_t}$ ($\mathbf{R}_t \in \mathbb{R}^{3 \times 3}$) and translation vectors $\{\mathbf{T}_t\}_{t=1}^{N_t}$ ($\mathbf{T}_t \in \mathbb{R}^{3 \times 1}$), where N_t represents the number of frames. The transformation is formulated as:

$$\begin{aligned} \mu_w &= \mathbf{R}_t \mu_o + \mathbf{T}_t, \\ \mathbf{R}_w &= \mathbf{R}_t \mathbf{R}_o, \end{aligned} \quad (5)$$

where μ_w and \mathbf{R}_w denote the mean position and rotation matrix of the Gaussian primitives in the world coordinate system, respectively. With this representation, we can reconstruct and render the dynamic scene with easy composition of separate models, which further enables flexibly scene editing applications.

For the initialization of point clouds for background model, we estimate the normals of all Gaussian primitives with KNN algorithms and initialize the Gaussian orientations towards the normals. Then we fuse the multi-frame LiDAR point clouds and downsample them via voxel downsampling with a voxel size of 0.15. For object model less than $8K$ points, we randomly sample points inside the 3D bounding box and concatenate with the original points to reach $8K$ points.

3.3. Gaussian-based Ray Tracing

To render novel LiDAR views, a straightforward approach is to rasterize the Gaussian primitives onto the LiDAR

imaging plane and accumulated LiDAR properties through Eq. 2, similar to the process of general camera sensors. However, the vanilla rasterization approach of 3DGS [16] does not support the rendering of cylindrical range image produced by LiDAR sensors. Furthermore, the rasterization imaging process is not aligned with the active nature of LiDAR sensors, which rely on the emission and reception of laser beams. Therefore, a ray-based rendering approach is essential for accurately simulating the LiDAR sensors. Building upon these insights and inspired by the latest point-based ray tracing methodologies [5, 12, 24, 28], we develop a Gaussian-based ray tracer based on the NVIDIA OptiX framework [32] for hardware acceleration.

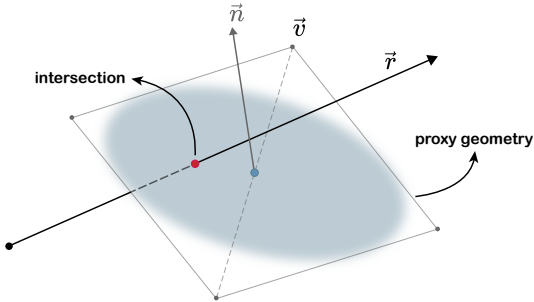


Figure 3. **Ray intersection with proxy geometries.** We construct the proxy geometry for each 2D Gaussian primitive as a pair of co-planar triangles, then the ray tracer performs intersection tests with the vertices \vec{v} and surface normal \vec{n} .

Proxy Geometries. In the realm of traditional graphics rendering [1, 22], proxy geometries serve as the simplified representations to approximate the complex original geometries, which can accelerate the ray tracing process. Considering the efficiency in constructing the BVH and the performance of intersection tests, we opt for a two-dimensional Gaussian primitive in the form of a planar disk, as elaborated in [14]. Thereby, the most efficient and simplest proxy geometry for encapsulating such Gaussian primitive is a pair of co-planar triangles (Fig. 3). Compared to axis-aligned bounding boxes (AABBs) or other polyhedron meshes, this approach reduces the number of meshes and tightly wraps the Gaussian primitive. Additionally, the sample locations are same as the ray intersections, without the approximation with maximum response, as described in [18, 28]. More details on proxy geometries for Gaussian variants can be found in our supplementary material.

Forward Ray Tracing. After constructing the acceleration structures, our method proceed to generate batch rays from the LiDAR sensor, formulated as $\mathcal{R} = \{\mathbf{r}_o, \mathbf{r}_d\}$, where $\mathbf{r}_o \in \mathbb{R}^{N_r \times 3}$ and $\mathbf{r}_d \in \mathbb{R}^{N_r \times 3}$ denote the ray origins and normalized ray directions, with N_r being the total number of batch rays. our ray tracing pipeline is initialized by the *optixLaunch* [32] program. Specifically, given a ray r within a batch of rays \mathcal{R} , the *ray-gen* program is invoked

to cast the ray against the BVH. The tracer then utilizes triangle vertices and surface normal to compute intersections (Fig. 3), using the *any-hit* program to maintain the intersections in an sorted buffer. However, sorting all intersections along each ray directly is inefficient, particularly in large-scale scenes with many Gaussian primitives. To deal with this problem, we divide the ray into several chunks during the round of tracing, each containing a fixed number of intersections, thereby reducing the sorting overhead to a single chunk. When the count of intersections reaches the chunk size, we take out the indices \mathcal{I} of the intersected primitives and depth values \mathcal{D} at the intersections from the sorted buffer. Subsequently, we evaluate the Gaussian response and calculate the point-wise LiDAR properties (ζ_i, β_i) at each sample point. The pixel value is rendered through Eq. 2 by replacing c_i with ζ_i and β_i . Following this, ray marching advances to the next chunk, starting from the last evaluated point. This iterative process continues until all Gaussian primitives intersecting the ray have been traversed, or the accumulated transmittance along the ray falls below a predefined minimum threshold T_{min} .

3.4. Differentiable LiDAR Rendering

LiDAR Modeling. With the purpose of rendering LiDAR views, we present the LiDAR view as the form of range image. Given a LiDAR scan \mathcal{L}_i contains \mathcal{M}_i points each parameterized by (x, y, z, ζ) , we first calculate the distance d to the sensor center with $d = \sqrt{x^2 + y^2 + z^2}$. Then the azimuth angle θ and elevation angle ϕ is computed as follows:

$$\begin{aligned} \theta &= \arctan(y, x), \\ \phi &= \arcsin(z, d). \end{aligned} \quad (6)$$

Considering a LiDAR sensor with \mathbf{H} laser beams in the vertical plane and \mathbf{W} horizontal emissions, we can project the points onto the LiDAR imaging plane from the world coordinate system:

$$\begin{pmatrix} h \\ w \end{pmatrix} = \begin{pmatrix} 1 - (\phi + |f_{down}|/f_v) \\ (1 - \theta/\pi)/2 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{H} \\ \mathbf{W} \end{pmatrix}, \quad (7)$$

where (h, w) is the 2D coordinates on range image, and $f_v = |f_{up}| + |f_{down}|$ is the vertical FOV of LiDAR sensor.

Differentiable Rendering. Apart from the forward LiDAR rendering, our ray tracer also supports differentiable rendering to optimize the scene representation. In previous works [14, 16], the back-propagation is conducted in a back-to-front blending order:

$$\frac{\partial L}{\partial \alpha_i} = T_i c_i - \sum_{j=i+1}^N T_j \alpha_j c_j, \quad (8)$$

To compute gradients for the i -th Gaussian, we need the contributions from all subsequent blended Gaussians.

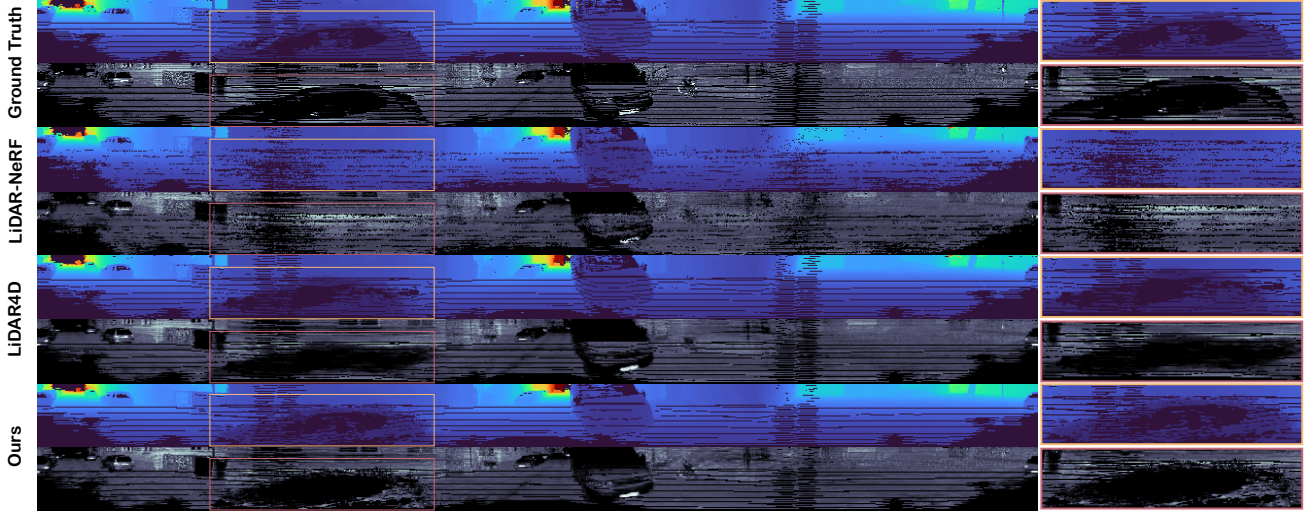


Figure 4. **Qualitative comparison on the KITTI-360 [20] dataset.** Dynamic vehicles are zoomed in for better visualization.

3DGS [16] meets this need by using per-tile sorted buffers. However, this approach is not available for us due to the distinction between the panoramic LiDAR models and camera models. A naive method to record the blending order in a global buffer during the forward pass is impractical, as it leads to unaffordable memory overhead and reduced ray tracing efficiency. To address this issue, we take the strategy proposed by [36] to perform the back-propagation in a front-to-back blending order, same as forward rendering:

$$\frac{\partial L}{\partial \alpha_i} = T_i c_i - (\mathcal{C} - \mathcal{C}_i) / (1 - \alpha_i), \quad (9)$$

where \mathcal{C} is the rendered pixel value and \mathcal{C}_i is the accumulated attribute including the i -th Gaussian in front-to-back order. Specifically, our backward pass starts by launching a new `optixLaunch` program defined in the backwards kernel, we then re-cast the same rays against the BVH to get the sorted intersections along the ray, computing the gradients via Eq. 9 and accumulating them in shared global buffers with atomic operations.

3.5. Optimization

Loss Function. Our method optimize the scene representation with the following loss functions:

$$\mathcal{L} = \lambda_d \mathcal{L}_d + \lambda_i \mathcal{L}_i + \lambda_r \mathcal{L}_r + \lambda_{CD} \mathcal{L}_{CD}, \quad (10)$$

where the λ_* denote the weights of the individual loss terms \mathcal{L}_* . The depth loss \mathcal{L}_d and intensity loss \mathcal{L}_i follow \mathcal{L}_1 loss, and ray-drop loss \mathcal{L}_r follows \mathcal{L}_{bce} loss. We employ the Chamfer Distance (CD) loss \mathcal{L}_{CD} [10, 58] to jointly supervise scene geometry. The CD loss is calculated with the point-wise distance between the rendered and

ground-truth point clouds:

$$\mathcal{L}_{CD} = \frac{1}{K} \left(\sum_{\hat{p}_i \in \hat{S}} \min_{p_i \in S} \|\hat{p}_i - p_i\|_2^2 + \sum_{p_i \in S} \min_{\hat{p}_i \in \hat{S}} \|p_i - \hat{p}_i\|_2^2 \right), \quad (11)$$

where K is $\min(|\hat{S}|, |S|)$, \hat{S} and S are the rendered and ground-truth point clouds, respectively.

Ray-drop Refinement. In driving scenarios, we categorize the ray-drop effects into two types: *scene-level* and *sensor-level*. *Scene-level* ray-drop occurs due to environmental factors like reflective materials and long detection ranges. While *sensor-level* ray-drop is caused by the inherent sensor biases that are independent of the scene representation, such as the hardware noises and limited visibility. Through our practice, we discovered that modeling ray-drops solely with SHs and optimizing them jointly with Gaussian primitives leads to inaccurate estimations in several areas. Inspired by LiDAR4D [58], we utilize a UNet [38] $\mathcal{F}_{\text{refine}}$ to refine the *sensor-level* ray-drop effects after the Gaussian optimization. Contrary to LiDAR4D, we feed the ray origins and ray directions as additional inputs so that the UNet can better percept the spatial movement of the sensor:

$$\beta_{\text{refine}} = \mathcal{F}_{\text{refine}}(d, \zeta, \beta, \mathbf{r}_o, \mathbf{r}_d). \quad (12)$$

We refine the rendered ray-drop mask via binary cross-entropy loss as follows:

$$\mathcal{L}_r = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} \mathcal{L}_{bce}(\hat{P}(\mathbf{r}), P(\mathbf{r})). \quad (13)$$

Moreover, we apply a random horizontal rotation to the sampled sensor pose and ground-truth image to avoid the overfitting to the specific pose.

Table 1. **Quantitative comparison on the Waymo Open Dataset [40]**. The resolution of range image is 64×2650 and metrics are averaged over all sequences. The cell colors present the **best** and the **second best** results, respectively.

Method	Efficiency		Depth					Intensity					Point Cloud	
	FPS	Storage	RMSE↓	MedAE↓	LPIPS↓	SSIM↑	PSNR↑	RMSE↓	MedAE↓	LPIPS↓	SSIM↑	PSNR↑	CD↓	F-score↑
LiDAR-NeRF [42]	0.98	1.6 GB	7.7258	0.0518	0.3414	0.6817	20.5204	0.0659	0.0108	0.1893	0.7492	23.1612	0.1815	0.9184
DyNFL [46]	0.21	14.9 GB	6.9787	0.0388	0.3010	0.7080	21.3094	0.0662	0.0093	0.1382	0.7555	24.0473	0.1182	0.7786
LiDAR4D [58]	0.17	7.7 GB	6.6234	0.0379	0.3406	0.7008	21.8413	0.0634	0.0086	0.1854	0.7756	24.0838	0.1060	0.9437
Ours	20.1	1.37 GB	6.4577	0.0340	0.2918	0.7330	22.1581	0.0627	0.0101	0.1652	0.7828	23.9079	0.1002	0.9458

Table 2. **Quantitative comparison on the KITTI-360 benchmarks [20]**. The resolution of range image is 66×1030 and metrics are averaged over all sequences. The color annotations are consistent with the Table 1 above.

Method	Efficiency		Depth					Intensity					Point Cloud	
	FPS	Storage	RMSE↓	MedAE↓	LPIPS↓	SSIM↑	PSNR↑	RMSE↓	MedAE↓	LPIPS↓	SSIM↑	PSNR↑	CD↓	F-score↑
PCGen [19]	0.05	5.23 GB	5.6853	0.2040	0.5391	0.4903	23.1675	0.1970	0.0763	0.5926	0.1351	14.1181	0.4636	0.8023
LiDARsim [25]	0.9	0.76 GB	6.9153	0.1279	0.2926	0.6342	21.4608	0.1666	0.0569	0.3276	0.3502	15.5853	3.2228	0.7157
LiDAR-NeRF [42]	1.8	1.61 GB	4.0886	0.0556	0.2712	0.6309	26.0590	0.1464	0.0438	0.3590	0.3567	16.7621	0.1502	0.9073
LiDAR4D [58]	0.4	7.38 GB	3.5256	0.0404	0.1051	0.7647	27.4767	0.1195	0.0327	0.1845	0.5304	18.5561	0.1089	0.9272
Ours	42.7	0.41 GB	3.4671	0.0512	0.1016	0.8406	27.6755	0.1115	0.0271	0.1812	0.6077	19.0862	0.1077	0.9255

4. Implementation details

The implementation of LiDAR-RT is based on PyTorch [33] and our custom CUDA kernels [13] with NVIDIA OptiX framework [32]. We train LiDAR-RT for 30000 iterations with Adam optimizers [17] following the configurations of 3DGS [16]. After that, we refine the UNet [38] model for 500 epochs with a learning rate of $1e^{-3}$. In practice, the loss weights λ_d , λ_i , λ_r , and λ_{CD} are set to 0.1, 0.1, 0.01, 0.01 respectively. During ray tracing, we set the chunk size to 16 for the trade-off between memory consumption and efficiency. The near plane is set to 0.2. All the experiments are conducted on one single RTX 4090 GPU.

5. Experiments

5.1. Experimental Setup

Datasets. We conducted experiments on the public Waymo Open Dataset [40] and KITTI-360 benchmarks [20]. Both datasets is equipped with a 64-beam LiDAR sensor and an acquisition frequency of 10Hz. On Waymo dataset, we follow the settings of DyNFL [46], selecting 4 dynamic sequences and 4 static sequences for experiments. Each selected sequence has 50 continuous frames, we sample every 10^{th} frame in the sequence as the test frames and use the remaining for training, the resolution of the range image is 64×2650. On KITTI-360 dataset, we select 6 dynamic sequences and 4 static sequences for experiments and evaluate our method with the settings of LiDAR4D [58]. The sample strategy is as same as the Waymo dataset. However, the KITTI-360 dataset only provides per-frame point clouds instead of the raw range images, so we fuse the multi-frame point clouds and project the fused point clouds to range images with the resolution of 66×1030.

Table 3. **Ablation studies on dynamic sequence of the KITTI-360 dataset [20]**. \mathcal{L}_{CD} : Chamfer Distance loss, \mathcal{I} : Point cloud initialization with normals, \mathcal{M} : Ray-drop modeling. Sec. 5.3 provides a detailed description for each ablation term.

	Depth			Intensity			Point Cloud	
	RMSE↓	SSIM↑	PSNR↑	RMSE↓	SSIM↑	PSNR↑	CD↓	F-score↑
<i>w/o</i> \mathcal{L}_{CD}	3.5053	0.8393	27.5526	0.1120	0.6068	19.0553	0.1192	0.9464
<i>w/o</i> \mathcal{I}	4.0597	0.8343	26.2554	0.1141	0.6024	18.8896	0.1363	0.9389
<i>w/o</i> \mathcal{M}	3.7445	0.7989	26.9473	0.1179	0.5687	18.6051	0.1234	0.9286
Ours	3.4671	0.8406	27.6755	0.1115	0.6077	19.0862	0.1077	0.9255

Baseline Methods. We perform a comprehensive comparison of our method with various types of baselines: (1) LiDARsim [25] and PCGen [19] are mesh-based reconstruction methods, we reproduce the results under the same settings in their papers. (2) LiDAR-NeRF [42] is the first NeRF-based method for LiDAR re-simulation, we use the official implementation for comparison. (3) LiDAR4D [58] and DyNFL [46] extend neural fields to dynamic scenes through 4D hybrid features and tracked bounding boxes, which are the primary methods we compare with. Please refer to the supplementary materials for more implementation details of baseline methods.

Evaluation Metrics. Following the previous works [15, 42, 46, 58], we evaluate the performance of our method with following metrics: We use Chamfer Distance (CD) [10] and F-score (error threshold is 5cm) to quantify the 3D geometric error between the generated and ground-truth point clouds. For evaluating range accuracy, we report the Root Mean Square Error (RMSE) and Median Absolute Error (MedAE), along with the PSNR, SSIM [45], and LPIPS [57] to measure the overall variance. The intensity results are evaluated in the same manner with the range. In addition, we present the efficiency of our method in terms of the inference FPS and training GPU storage.

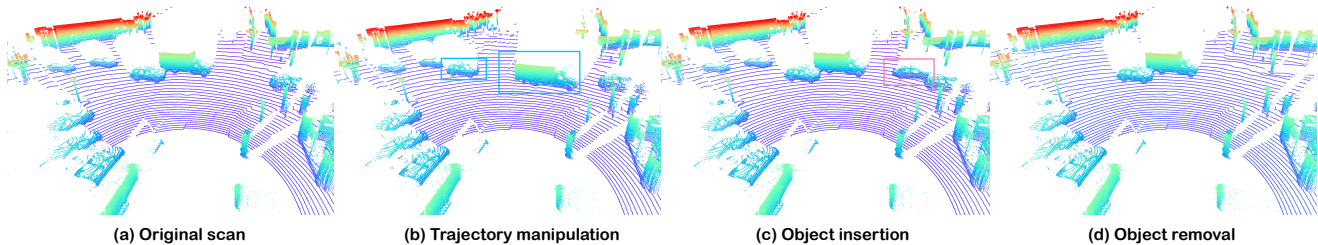


Figure 5. **Scene editing results on the Waymo dataset [40].** Our method supports various scene editing operations, including trajectory manipulation, object insertion, and object removal. The boxes (\square/\square) highlight the edited objects.

5.2. Comparisons with the State-of-the-art

Tabs. 1, 2 and Fig. 4 show the quantitative and qualitative comparisons with the SOTA NeRF-based methods [42, 46, 58] on the Waymo [40] and KITTI-360 [20] datasets. As evident in Tab. 1 and Tab. 2, our method achieves notably better rendering quality compared to the SOTA methods [46, 58], while rendering at two orders of magnitude faster and requiring significantly less training storage. As shown in Fig. 4, LiDAR-NeRF [42] fails to model the dynamic objects, while LiDAR4D [58] suffers from blurry results and ray-drop estimation errors due to the lack of capacity of their model to capture the complex scenes. In contrast, our LiDAR-RT is capable of generating high-fidelity and physically plausible LiDAR views for driving scenarios. More comparison results on the Waymo dataset [40] can be found in supplementary materials.

5.3. Ablations Studies

We perform ablation studies on the dynamic sequence of the KITTI-360 dataset [20]. More qualitative and quantitative results of the ablation studies are provided in our supplementary materials.

Ablation study on loss functions. As shown in Tab. 3 row 1, remove the \mathcal{L}_{CD} terms not only reduces the scene geometric accuracy (CD and F-score), but also leads to the degradation of other performance metrics.

Ablation study on point cloud initialization with normals. The "w/o \mathcal{I} " variant removes the normal-guided initialization for LiDAR point clouds (Sec. 3.2), which significantly impedes the quality of the scene geometry.

Ablation study on ray-drop modeling. The "w/o \mathcal{M} " variant removes the proposed ray-drop modeling module (Eq. 4) and replaces it with a simple ray-drop estimation. As shown in Tab. 3, this variant leads to the ray-drop estimation errors and inferior rendering quality.

5.4. Applications

Scene editing. Our instance-aware scene representation (Sec. 3.2) models each component separately and enables a wide range of scene editing operations. Fig. 5 demonstrates

the high-fidelity scene editing results on the Waymo dataset [40], including trajectory manipulation (b), object insertion (c), and object removal (d).

Sensor re-simulation. Thanks to our effective Gaussian-based ray tracer (Sec. 3.3), we can easily re-simulate LiDAR sensors with varying configurations. As shown in Fig. 6, our method can generate realistic LiDAR point clouds with different sensor poses, beam numbers and FOV settings. This flexibility is particularly beneficial for downstream tasks in autonomous driving.

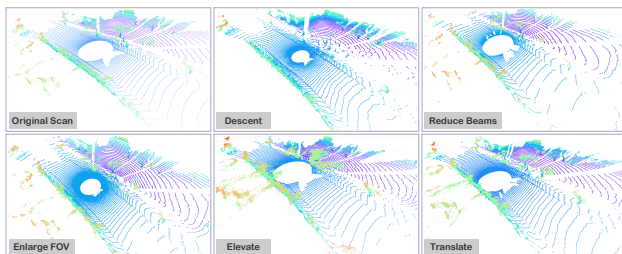


Figure 6. **LiDAR re-simulation results with different sensor configurations on the Waymo dataset [40].** Our LiDAR-RT can generate realistic LiDAR point clouds with varying sensor poses, beam numbers, and FOV settings.

6. Conclusion and Discussion

In this paper, we propose LiDAR-RT, a novel framework designed for real-time, high-fidelity LiDAR re-simulation of dynamic driving scenarios. We decompose dynamic scenes into a static background and multiple moving objects, each represented by a set of Gaussian primitives, and incorporate scene graphs to handle scene dynamics. Furthermore, we develop a differentiable LiDAR rendering pipeline that utilizes the hardware-accelerated ray tracing algorithms to generate realistic novel LiDAR views, also maintaining great performance under various scene editing operations and sensor configurations. Extensive experiments across a variety of public autonomous driving benchmarks have demonstrated that our method achieves state-of-the-art performance in terms of both rendering

quality and efficiency.

However, our method has certain limitations. Firstly, LiDAR-RT cannot accurately model non-rigid objects such as pedestrians and cyclists due to their substantial deformations across frames, which are important for some downstream tasks. Moreover, the performance and rendering speed of LiDAR-RT are impacted when dealing with long driving sequences, as the number of Gaussian primitives increases dramatically. How to model non-rigid objects and improve the efficiency for long driving sequences are remaining challenges for future works.

References

- [1] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphs, 2020. [5](#)
- [2] Benjamin Attal, Jia-Bin Huang, Christian Richardt, Michael Zollhoefer, Johannes Kopf, Matthew O’Toole, and Changil Kim. HyperReel: High-fidelity 6-DoF video with ray-conditioned sampling. In *CVPR*, 2023. [2](#)
- [3] Louis Bavoil, Steven P. Callahan, Aaron Lefohn, João L. D. Comba, and Cláudio T. Silva. Multi-fragment effects on the gpu using the k-buffer. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, page 97–104, New York, NY, USA, 2007. Association for Computing Machinery. [2](#)
- [4] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *CVPR*, pages 4413–4421, 2018. [4](#)
- [5] Hugo Blanc, Jean-Emmanuel Deschaud, and Alexis Paljic. RayGauss: Volumetric gaussian-based ray casting for photo-realistic novel view synthesis, 2024. [3](#), [5](#)
- [6] Lucas Caccia, Herke van Hoof, Aaron Courville, and Joelle Pineau. Deep generative modeling of LiDAR data, 2019. [2](#)
- [7] Yurui Chen, Chun Gu, Junzhe Jiang, Xiatian Zhu, and Li Zhang. Periodic vibration gaussian: Dynamic urban scene reconstruction and real-time rendering. *arXiv:2311.18561*, 2023. [1](#), [3](#)
- [8] Ziyu Chen, Jiawei Yang, Jiahui Huang, Riccardo de Lutio, Janick Martinez Esturo, Boris Ivanovic, Or Litany, Zan Gojcic, Sanja Fidler, Marco Pavone, Li Song, and Yue Wang. OmniRe: Omni urban scene reconstruction. *arXiv preprint arXiv:2408.16760*, 2024. [1](#), [3](#)
- [9] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. [2](#)
- [10] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3D object reconstruction from a single image, 2016. [6](#), [7](#)
- [11] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *CVPR*, 2023. [2](#)
- [12] Jian Gao, Chun Gu, Youtian Lin, Hao Zhu, Xun Cao, Li Zhang, and Yao Yao. Relightable 3d gaussian: Real-time point cloud relighting with brdf decomposition and ray tracing. *arXiv:2311.16043*, 2023. [3](#), [5](#)
- [13] Jaysree Ghorpade. GPGPU processing in CUDA architecture. *Advanced Computing: An International Journal*, 3(1): 105–120, 2012. [7](#)
- [14] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024. [2](#), [3](#), [5](#), [13](#)
- [15] 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 *ICCV*, 2023. [2](#), [4](#), [7](#), [12](#)
- [16] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *TOG*, 42(4):139:1–139:14, 2023. [2](#), [3](#), [4](#), [5](#), [6](#), [7](#), [12](#), [13](#)
- [17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. [7](#)
- [18] Aaron Knoll, R Keith Morley, Ingo Wald, Nick Leaf, and Peter Messmer. Efficient particle volume splatting in a ray tracer. *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*, pages 533–541, 2019. [5](#)
- [19] Chenqi Li, Yuan Ren, and Bingbing Liu. PCGen: Point cloud generator for lidar simulation. In *ICRA*, pages 11676–11682, 2023. [2](#), [7](#), [12](#)
- [20] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3292–3310, 2022. [2](#), [6](#), [7](#), [8](#), [12](#)
- [21] Carl Lindström, Georg Hess, Adam Lilja, Maryam Fatemi, Lars Hammarstrand, Christoffer Petersson, and Lennart Svensson. Are nerfs ready for autonomous driving? towards closing the real-to-simulation gap. In *CVPRW*, 2024. [2](#)
- [22] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *ACM Trans. Graph.*, 40(4), 2021. [5](#)
- [23] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024. [3](#)
- [24] Alexander Mai, Peter Hedman, George Kopanas, Dor Verbin, David Futschik, Qiangeng Xu, Falko Kuester, Jon Barron, and Yinda Zhang. EVER: Exact volumetric ellipsoid rendering for real-time view synthesis, 2024. [3](#), [5](#)
- [25] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. LiDARsim: Realistic lidar simulation by leveraging the real world. In *CVPR*, pages 11167–11176, 2020. [2](#), [7](#), [12](#)
- [26] 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 *ICCV*, pages 8238–8248, 2023. [2](#), [4](#)

- [27] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, page 405–421, 2020. [2](#)
- [28] Nicolas Moenne-Loccoz, Ashkan Mirzaei, Or Perel, Riccardo de Lutio, Janick Martinez Esturo, Gavriel State, Sanja Fidler, Nicholas Sharp, and Zan Gojcic. 3D Gaussian Ray Tracing: Fast tracing of particle scenes. *ACM Transactions on Graphics and SIGGRAPH Asia*, 2024. [3](#), [5](#), [13](#)
- [29] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *CVPR*, pages 2856–2865, 2021. [2](#), [3](#)
- [30] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021. [2](#)
- [31] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. HyperNeRF: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), 2021. [2](#)
- [32] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: a general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4), 2010. [5](#), [7](#)
- [33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library, 2019. [7](#)
- [34] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: surface elements as rendering primitives. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, page 335–342, USA, 2000. ACM Press/Addison-Wesley Publishing Co. [2](#)
- [35] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *CVPR*, 2020. [2](#)
- [36] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted gaussian splatting for view-consistent real-time rendering. *ACM Trans. Graph.*, 43(4), 2024. [3](#), [6](#)
- [37] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, page 497–500, New York, NY, USA, 2001. Association for Computing Machinery. [4](#)
- [38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pages 234–241, Cham, 2015. Springer International Publishing. [6](#), [7](#)
- [39] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles, 2017. [2](#)
- [40] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, pages 2446–2454, 2020. [2](#), [7](#), [8](#), [12](#), [14](#), [15](#)
- [41] Tao Tang, Guangrun Wang, Yixing Lao, Peng Chen, Jie Liu, Liang Lin, Kaicheng Yu, and Xiaodan Liang. AlignMiF: Geometry-aligned multimodal implicit field for lidar-camera joint synthesis. *arXiv preprint arXiv:2402.17483*, 2024. [2](#)
- [42] Tang Tao, Longfei Gao, Guangrun Wang, Yixing Lao, Peng Chen, Hengshuang Zhao, Dayang Hao, Xiaodan Liang, Mathieu Salzmann, and Kaicheng Yu. LiDAR-NeRF: Novel lidar view synthesis via neural radiance fields. *arXiv preprint arXiv:2304.10406*, 2023. [2](#), [4](#), [7](#), [8](#), [12](#), [15](#)
- [43] Adam Tonderski, Carl Lindström, Georg Hess, William Ljungbergh, Lennart Svensson, and Christoffer Petersson. NeuRAD: Neural rendering for autonomous driving. In *CVPR*, 2024. [2](#), [3](#), [4](#)
- [44] Haithem Turki, Jason Y Zhang, Francesco Ferroni, and Deva Ramanan. SUDS: Scalable urban dynamic scenes. In *CVPR*, 2023. [2](#)
- [45] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004. [7](#)
- [46] Hanfeng Wu, Xingxing Zuo, Stefan Leutenegger, Or Litany, Konrad Schindler, and Shengyu Huang. Dynamic lidar re-simulation using compositional neural fields. In *CVPR*, 2024. [2](#), [4](#), [7](#), [8](#), [12](#), [15](#)
- [47] Zirui Wu, Tianyu Liu, Liyi Luo, Zhide Zhong, Jianteng Chen, Hongmin Xiao, Chao Hou, Haozhe Lou, Yuan-tao 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. In *CICAI*, 2023. [3](#)
- [48] Ziyang Xie, Junge Zhang, Wenye Li, Feihu Zhang, and Li Zhang. S-NeRF: Neural radiance fields for street views. In *ICLR 2023*, 2023. [2](#)
- [49] Weiyi Xue, Zehan Zheng, Fan Lu, Haiyun Wei, Guang Chen, and Changjun Jiang. GeoNLF: Geometry guided pose-free neural lidar fields, 2024. [2](#)
- [50] Yunzhi Yan, Haotong Lin, Chenxu Zhou, Weijie Wang, Haiyang Sun, Kun Zhan, Xianpeng Lang, Xiaowei Zhou, and Sida Peng. Street gaussians for modeling dynamic urban scenes. In *ECCV*, 2024. [1](#), [3](#), [4](#), [12](#)
- [51] Jiawei Yang, Boris Ivanovic, Or Litany, Xinshuo Weng, Seung Wook Kim, Boyi Li, Tong Che, Danfei Xu, Sanja Fidler, Marco Pavone, and Yue Wang. EmerNeRF: Emergent spatial-temporal scene decomposition via self-supervision. In *ICLR*, 2024. [3](#)
- [52] Zhenpei Yang, Yuning Chai, Dragomir Anguelov, Yin Zhou, Pei Sun, Dumitru Erhan, Sean Rafferty, and Henrik Kretzschmar. Surfelgan: Synthesizing realistic sensor data for autonomous driving. In *CVPR*, 2020. [2](#)
- [53] 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 *CVPR*, 2023. [2](#), [3](#)

- [54] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. AbsGS: Recovering fine details for 3d gaussian splatting, 2024. [13](#)
- [55] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-Splatting: Alias-free 3d gaussian splatting. *CVPR*, 2024. [3](#), [13](#)
- [56] Junge Zhang, Feihu Zhang, Shaochen Kuang, and Li Zhang. NeRF-LiDAR: Generating realistic lidar point clouds with neural radiance fields, 2024. [2](#)
- [57] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. [7](#)
- [58] Zehan Zheng, Fan Lu, Weiyi Xue, Guang Chen, and Changjun Jiang. LiDAR4D: Dynamic neural fields for novel space-time view lidar synthesis. In *CVPR*, pages 5145–5154, 2024. [1](#), [2](#), [4](#), [6](#), [7](#), [8](#), [12](#), [15](#)
- [59] Hongyu Zhou, Jiahao Shao, Lu Xu, Dongfeng Bai, Weichao Qiu, Bingbing Liu, Yue Wang, Andreas Geiger, and Yiyi Liao. HUGS: Holistic urban 3d scene understanding via gaussian splatting. In *CVPR*, pages 21336–21345, 2024. [1](#), [3](#)
- [60] Xiaoyu Zhou, Zhiwei Lin, Xiaojun Shan, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. Drivinggaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes. *arXiv preprint arXiv:2312.07920*, 2023. [3](#)
- [61] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa volume splatting. In *Proceedings Visualization, 2001. VIS '01.*, pages 29–538, 2001. [4](#)
- [62] Vlas Zyrianov, Xiyue Zhu, and Shenlong Wang. Learning to generate realistic lidar point clouds. In *ECCV*, 2022. [2](#)
- [63] Vlas Zyrianov, Henry Che, Zhijian Liu, and Shenlong Wang. LidarDM: Generative lidar simulation in a generated world, 2024. [2](#)

LiDAR-RT: Gaussian-based Ray Tracing for Dynamic LiDAR Re-simulation

Supplementary Material

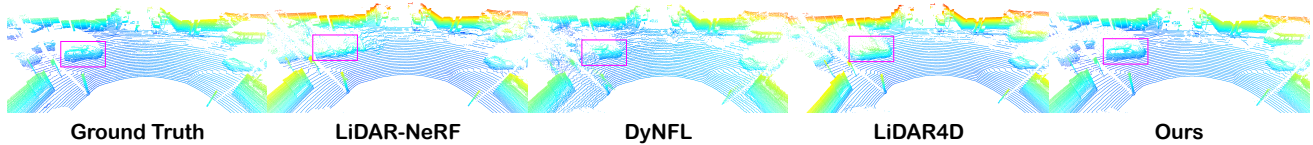


Figure 7. **Qualitative comparison of novel view LiDAR point clouds on Waymo Open Dataset [40].** Our LiDAR-RT generates a realistic novel LiDAR view with accurate scene geometry and high-frequency details of dynamic objects.

In this supplementary material, we begin by presenting additional implementation details about the evaluation datasets and the baseline methods in Sec. A. Then we display the further ablation studies in Sec. B and comparison results on Waymo [40] and KITTI-360 [20] datasets in Sec. C. Subsequently, Sec. D showcases more applications of our method.

A. Implementation details

A.1. Evaluation datasets.

We evaluate our method on the Waymo Open Dataset [40] and the KITTI-360 dataset [20]. Following previous works [42, 46, 58], we select the static and dynamic sequences from the both datasets. The specific selected sequence names and corresponding ids are listed in Tab. 4 and Tab. 5.

Table 4. **The selected sequences from the KITTI-360 [20] dataset for evaluation.** S and D denote static and dynamic sequences, respectively.

Scene name	Type	Sequence id	Start frame	End frame
<i>ks1</i>	S	Seq 1538-1601	1538	1601
<i>ks2</i>	S	Seq 1728-1791	1728	1791
<i>ks3</i>	S	Seq 1908-1971	1908	1971
<i>ks4</i>	S	Seq 3353-3415	3353	3415
<i>kd1</i>	D	Seq 2351-2400	2351	2400
<i>kd2</i>	D	Seq 4951-5000	4951	5000
<i>kd3</i>	D	Seq 8121-8170	8121	8170
<i>kd4</i>	D	Seq 10201-10250	10201	10250
<i>kd5</i>	D	Seq 10751-10800	10751	10800
<i>kd6</i>	D	Seq 11401-11450	11401	11450

Table 5. **The selected sequences from the Waymo Open [40] dataset for evaluation.** S and D denote static and dynamic sequences, respectively.

Scene name	Type	Sequence id	Start frame	End frame
<i>ws1</i>	S	Seg 113792	1	50
<i>ws2</i>	S	Seg 106762	1	50
<i>ws3</i>	S	Seg 177619	1	50
<i>ws4</i>	S	Seg 117240	1	50
<i>wd1</i>	D	Seg 108305	148	197
<i>wd2</i>	D	Seg 132712	51	100
<i>wd3</i>	D	Seg 100721	1	50
<i>wd4</i>	D	Seg 105003	148	197

A.2. Baseline methods

LiDARsim and PCGen. LiDARsim [25] and PCGen [19] are surfel-based reconstruction methods. Since the official implementation is not publicly available, we re-implement these two methods based on the codebase provided by the LiDAR-NeRF [42] and follow the same experimental settings on the KITTI-360 dataset [20].

LiDAR-NeRF. LiDAR-NeRF [42] is the first NeRF-based method for LiDAR re-simulation, we directly adopt the official implementation. For KITTI-360 dynamic sequences and Waymo [40] scenes, we adjust the scene scales and LiDAR resolutions for fair comparison.

LiDAR4D. LiDAR4D [58] utilizes a 4D hybrid representation combined with multi-planar and grid features for LiDAR re-simulation. We adopt the official implementation and follow the same experimental settings as their paper on the KITTI-360 dataset. For the Waymo dataset, we preprocess the dataset following the same procedure as LiDAR4D and adjust the LiDAR resolutions. The ray-drop refinement is also conducted for evaluation sequences.

DyNFL. DyNFL [46] leverages the bounding boxes of moving objects to construct an editable neural field for high-fidelity re-simulation of LiDAR scans. We follow the original implementation based on NFL Studio [15] and the settings for the Waymo dataset.

A.3. Gaussian Densification.

We adopt the adaptive control techniques from 3DGS [16] during optimization, which includes operations such as pruning, cloning, and splitting. However, unlike the vanilla 3DGS [16], which tracks screen-space gradients of particle positions for cloning and splitting decisions, our approach utilizes gradients in 3D world-space. This method is more general and suitable in our ray tracing context since the forward and backward passes are performed in 3D space. Furthermore, to prevent object Gaussians from expanding into occluded areas, we follow the strategy of [50] and sample a set of points for each object model to form a probability distribution function. During optimization,

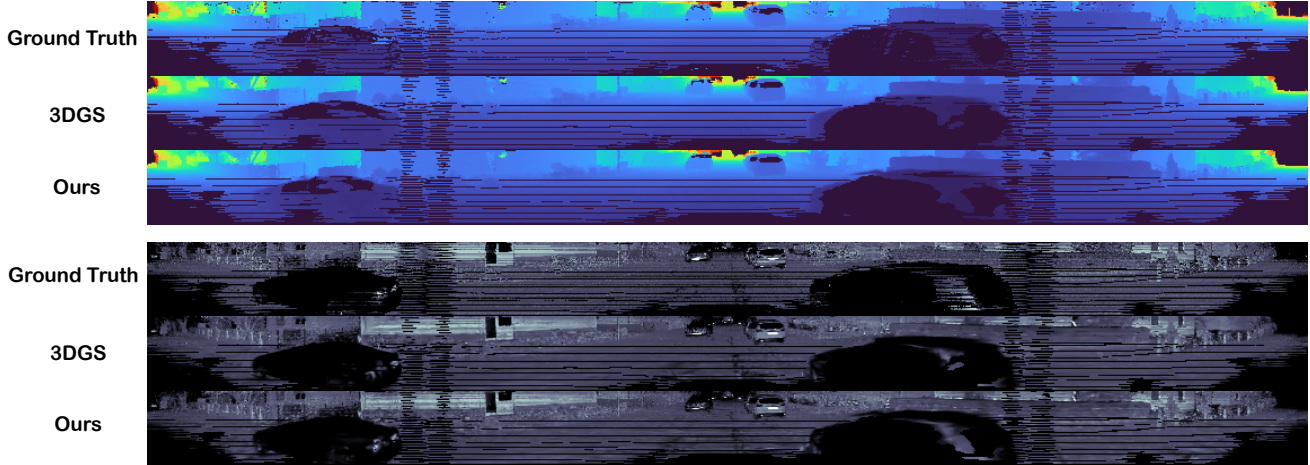


Figure 8. Qualitative results of ablation study on ray tracing with Gaussian variants.



Figure 9. Qualitative results of ablation study on ray-drop modeling and refinement.

Gaussians associated with sampled points that fall outside the bounding box are pruned to avoid excessive growth.

Table 6. Quantitative results of ablation study on ray tracing with Gaussian variants. The cell colors present the best and the second best results, respectively.

Method	FPS \uparrow	RMSE \downarrow	LPIPS \downarrow	PSNR \uparrow	CD \downarrow	F-score \uparrow
3D Gaussians	29	3.6716	0.1145	27.0976	0.3553	0.8899
2D Gaussians (Ours)	42	3.4671	0.1070	27.6755	0.1077	0.9255

Table 7. Quantitative results of ablation study on ray-drop modeling and refinement. The cell colors present the best and the second best results, respectively.

Method	RMSE \downarrow	LPIPS \downarrow	PSNR \uparrow	CD \downarrow	F-score \uparrow
w/o \mathcal{R}_{hit}	4.5482	0.4503	25.2371	0.1592	0.9089
w/o \mathcal{R}_{refine}	4.4635	0.4338	25.3924	0.1485	0.9119
w/o $\mathcal{R}_{spatial}$	3.7571	0.1480	26.9385	0.1247	0.9249
Ours	3.4671	0.1070	27.6755	0.1077	0.9255

B. More Ablation studies

B.1. Impact of ray tracing with Gaussian variants

Tab. 6 and Fig. 8 show the quantitative and qualitative results of the ablation study on ray tracing with Gaussian variants. We adopt 3D Gaussians [16] and 2D Gaussians [14] as our Gaussian primitives for ray tracing separately. As for 3D Gaussians, we construct the corresponding proxy geometry as an icosahedron, proposed by 3DGRT [28]. The results demonstrate that the 2D Gaussians have a slight advantage over 3D Gaussians in terms of rendering quality and efficiency, which means our ray tracer is compatible with various types of Gaussian primitives and other extensions [54, 55] applied on Gaussian primitives can be easily integrated into our framework.

B.2. Impact of ray-drop modeling and refinement

Tab. 7 and Fig. 9 present the quantitative and qualitative results of our detailed ablation study on ray-drop modeling

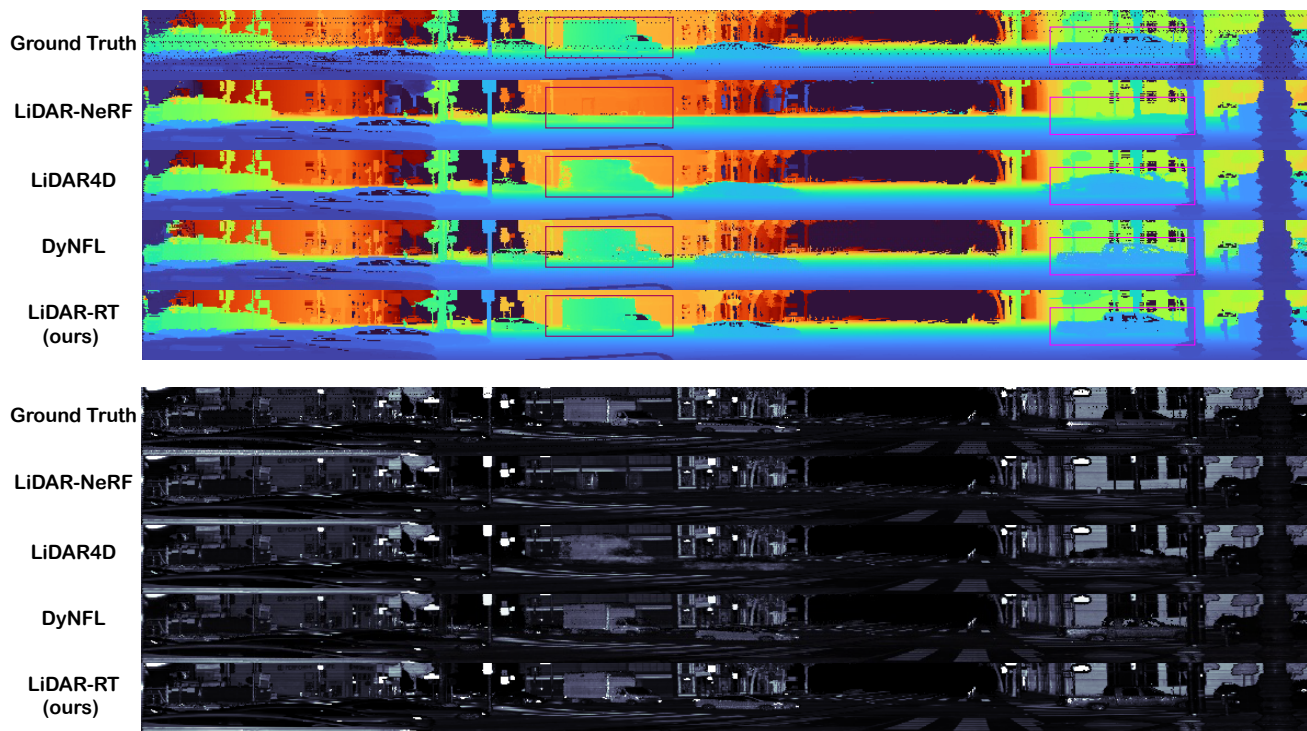


Figure 10. Qualitative comparison of LiDAR range images on Waymo Open Dataset [40] sequence seg-132712.

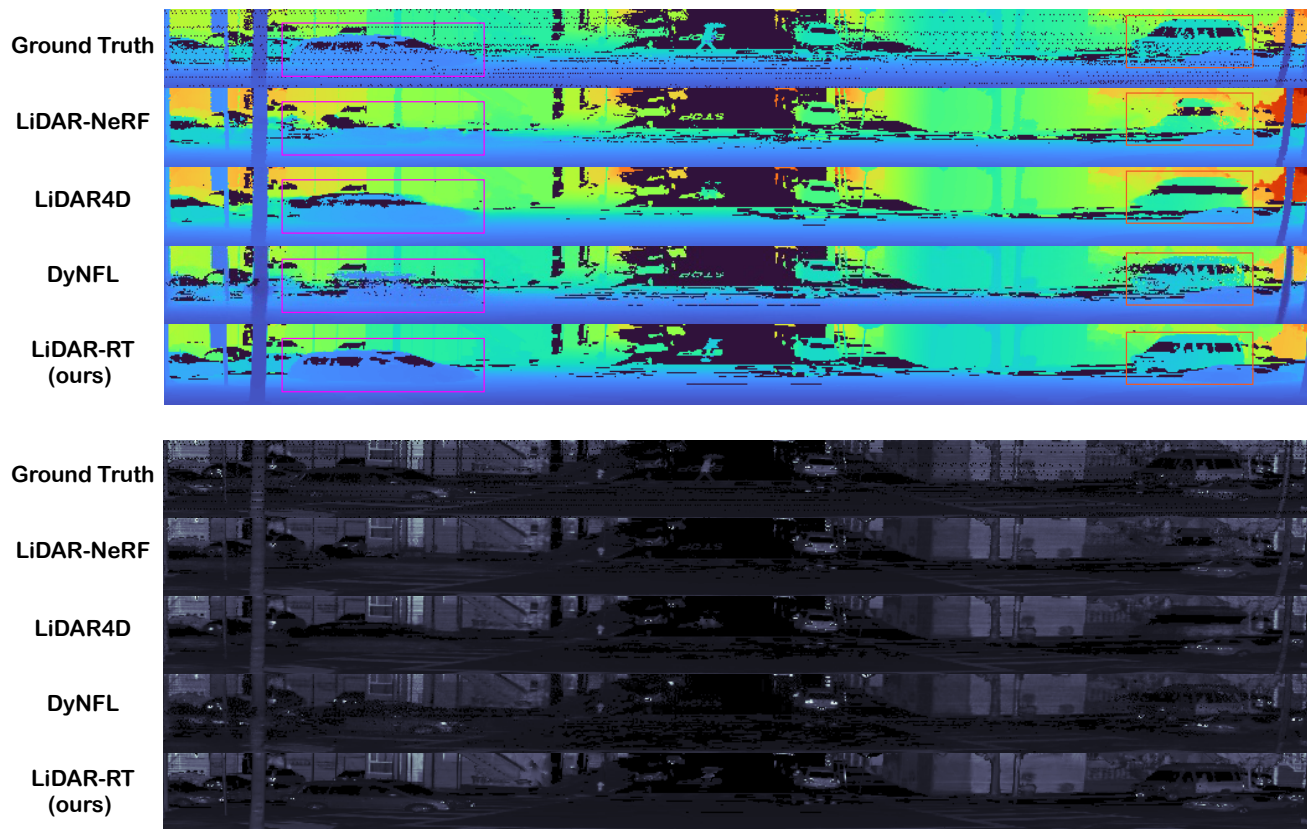


Figure 11. Qualitative comparison of LiDAR range images on Waymo Open Dataset [40] sequence seg-108305.

and refinement. The variant labeled *w/o* \mathcal{R}_{hit} models the ray-drop using only a single logit, results in a significant degradation of rendering quality. The variant *w/o* \mathcal{R}_{ref} omits the refinement stage, consequently failing to capture the *sensor-level* ray-drop patterns. Lastly, the *w/o* $\mathcal{R}_{\text{spatial}}$ variant disregards the ray information $(\mathbf{r}_o, \mathbf{r}_d)$ as UNet inputs, leading to a loss of details on dynamic objects.

C. Additional results

We provide additional qualitative results on the Waymo dataset [40] with multiple baselines, as shown in Fig. 7, Fig. 10, and Fig. 11. The dynamic vehicles are highlighted with colored bounding boxes (\square/\square) for better visualization. Even on the challenging Waymo [40] dataset with multiple moving actors and the complex urban environment, our LiDAR-RT still generates realistic novel LiDAR views with accurate geometry and high-frequency details of dynamic objects. In contrast, LiDAR-NeRF [42] struggles with dynamic objects due to its lack of temporal modeling. LiDAR4D [58] produces blurry and distorted results on this challenging dataset. While DyNFL [46] renders plausible results, also exhibits some artifacts around the dynamic objects due to the inaccurate estimations of ray-drop.

D. Applications

Object decomposition. Fig. 12 illustrates the object decomposition results on Waymo dataset [40]. Our method is capable of decomposing the foreground dynamic objects clearly and produces high fidelity rendering results.



Figure 12. **Decomposition results on Waymo dataset [40].** The points are colorized by intensity values from blue(0) to red (1).