SaLF: Sparse Local Fields for Multi-Sensor Rendering in Real-Time

Yun Chen[1,2*] Matthew Haines [1,3*] JingKang Wang[1,2]
Krzysztof Baron-Lis[1] Sivabalan Manivasagam[1,2] Ze Yang[1,2] Raquel Urtasun[1,2]
[1]Waabi [2] University of Toronto [3]University of Waterloo
{ychen, jwang, siva, klis, zyang, urtasun}@waabi.ai, m4haines@uwaterloo.ca

Abstract

High-fidelity sensor simulation of light-based sensors such as cameras and LiDARs is critical for safe and accurate autonomy testing. Neural radiance field (NeRF)-based methods that reconstruct sensor observations via ray-casting of implicit representations have demonstrated accurate simulation of driving scenes, but are slow to train and render, hampering scalability. 3D Gaussian Splatting (3DGS) has demonstrated faster training and rendering times through rasterization, but is primarily restricted to pinhole camera sensors, preventing usage for realistic multi-sensor autonomy evaluation. Moreover, both NeRF and 3DGS couple the representation with the rendering procedure (implicit networks for ray-based evaluation, particles for rasterization), preventing interoperability, which is key for general usage. In this work, we present Sparse Local Fields (SaLF), a novel volumetric representation that supports rasterization and raytracing. SaLF represents volumes as a sparse set of 3D voxel primitives, where each voxel is a local implicit field. SaLF has fast training (<30 min) and rendering capabilities (50+ FPS for camera and 600+ FPS for LiDAR), has adaptive pruning and densification to easily handle large scenes, and can support non-pinhole cameras and spinning LiDARs. We demonstrate that SaLF delivers realism comparable to existing self-driving sensor simulation methods while improving efficiency and enhancing capabilities, thereby enabling more scalable simulation. Please visit our project page for more results: https://waabi.ai/salf

1. Introduction

Closed-loop simulation has become an integral part of testing self-driving vehicles. In order to test the full autonomy system, modern simulators are equiped with the ability to simulate the sensors (e.g. LiDAR, camera) that the self-driving vehicle utilizes to perceive the world. Such a multi-modal sensor simulation system should be realistic to accurately measure autonomy performance, and be very efficient to enable scalable testing and training.

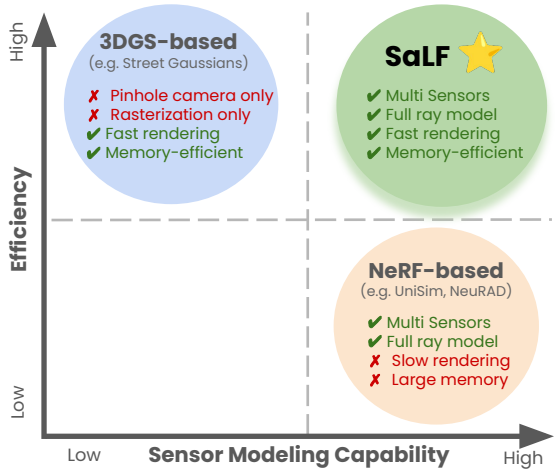

Figure 1. **SaLF combines high efficiency with advanced sensor modeling capabilities for self-driving simulations.**

Neural Radiance Field (NeRF)-based representations [28] have made significant progress in building realistic 3D multi-sensor simulators for self-driving [11, 31, 42, 43, 48, 51]. These methods represent the driving scene as a composable set of dynamic actors and static background that are modelled as 3D implicit representations. Each sensor is modelled as a set of outgoing rays into the scene representation according to its extrinsincs and intrinsics. NeRF-based methods then generate sensor data by querying the representations in space and time for the outgoing radiance along each ray and performing volume rendering that models how light interacts with the scene, resulting in high-fidelity outputs. However, their computational demands in training (multiple hours per scene) as well as rendering (1-2 FPS) have limited their use for scalable real-time simulation, especially when more than 20 sensors may be present on the vehicle [45]. Several methods have improved NeRF rendering efficiency by baking to accelerated representations suitable for fast rendering, such as meshes [5, 12, 25, 54] or

*Equal contributions.

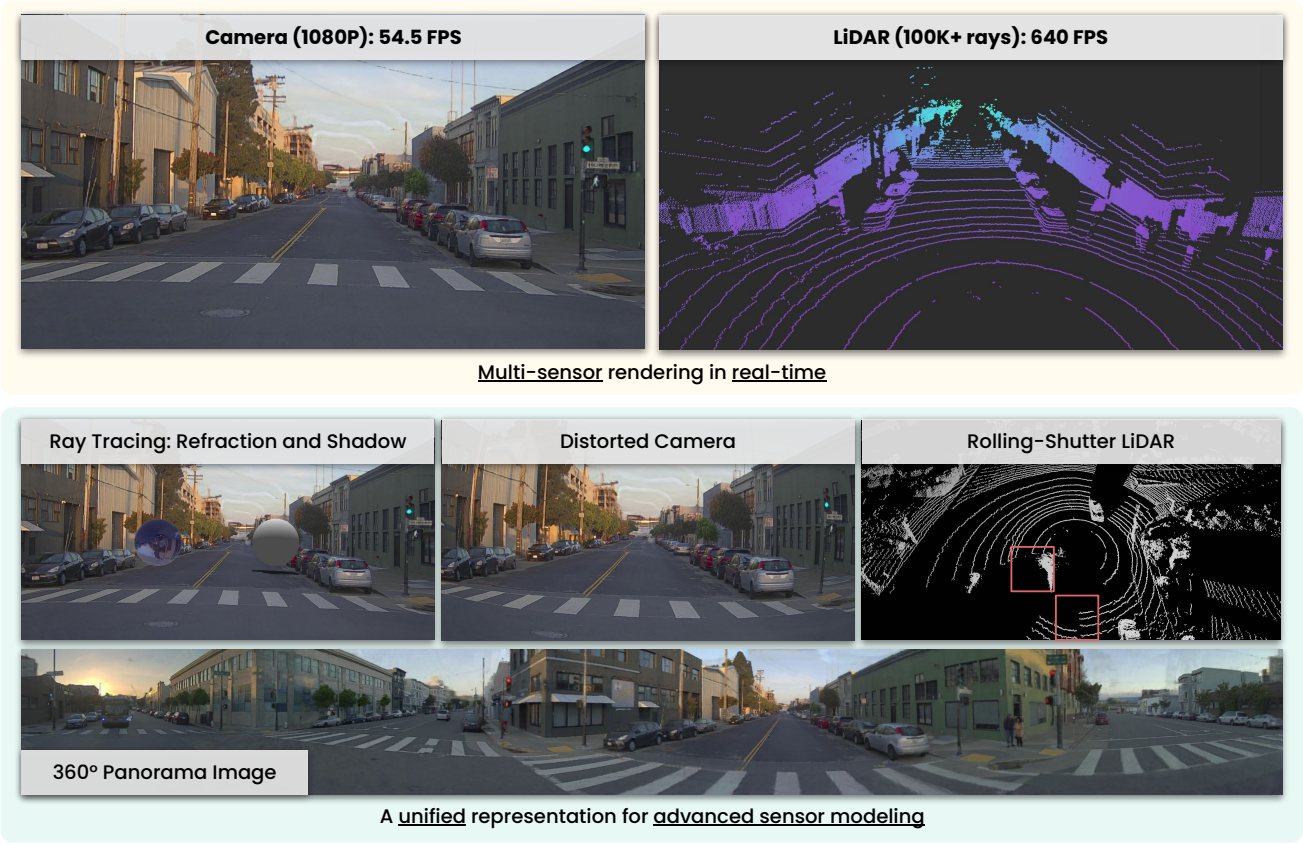

Figure 2. **Real-time self-driving sensor simulation with SaLF representation.** Our method achieves high-performance rendering for both camera and LiDAR, and supports advanced features including secondary effects (*e.g.*, refraction, reflection and shadow) and complex sensor models (*e.g.*, fisheye, rolling-shutter and panoramic cameras). This is made possible by an efficient and unified representation that supports both rasterization and ray-tracing.

grid look-ups [35, 56]. However, these methods still suffer from time-consuming training and complicated baking procedures that can be of lower quality compared to the original trained representation.

3D Gaussian Splatting (3DGS) [17, 50, 61] has recently demonstrated fast training and real-time rendering for camera images. This approach models the static and dynamic parts of the scene as a large set of explicit 3D Gaussian particles, assumes a pinhole camera model, and performs rasterization, "splatting" the particles and alpha-compositing them onto the image plane to generate the final image. Like other rasterization-based approaches, 3DGS does not directly support "ray-based" rendering, which is required for more complex sensor models such as rolling-shutter LiDARs or fish-eye cameras that are commonly used in self-driving. Moreover, it is difficult to accurately model sensor phenomena such as motion blur and secondary lighting effects like refraction, which are crucial for realistic testing of robust self-driving systems [27, 32, 44]. These limitations restrict its use in comprehensive self-driving testing despite its real-time rendering capabilities. Furthermore, similar to how meshes in graphics are compatible with both raytracing and rasterization, we argue that there should be a learnable volumetric representation that supports both as well.

In this work, we propose a novel representation called Sparse Local Field (SaLF). which seamlessly supports efficient raytracing and rasterization. SaLF is composed of a set of voxel primitives, where each voxel is a local implicit field that maps spatial coordinates to geometry and appearance. Similar to NeRF, SaLF can be volume rendered in a ray-casting fashion and can accurately model LiDAR and complex cameras. Without any baking, SaLF's voxels can natively be backed by an octree, directly supporting accelerated raytracing. Similar to 3DGS, SaLF can be rasterized efficiently using a tile-based rasterizer, enabling fast pinhole camera rendering. Additionally, SaLF supports adaptive voxel pruning and densification, creating compact representations and enabling modeling of large scenes.

SaLF efficiently and unifiedly supports a comprehensive set of sensor models and complex phenomena such

as rolling-shutter effects and refraction, which makes SaLF particularly valuable for scalable sensor simulation in autonomous driving. Experiments on the public self-driving dataset demonstrate that compared to prior work for self-driving sensor simulation, SaLF is more efficient in training and rendering while exhibiting comparable realism. We also showcase raytracing-based applications of SaLF, highlighting its potential for fast and versatile sensor simulation.

2. Related Work

Efficient NeRFs: Neural Radiance Fields (NeRF) [28] have become the foundation for photorealistic 3D scene reconstruction from 2D images. However, the vanilla NeRF formulation is computationally intensive, often requiring days of training and significant inference time, limiting its practicality for real-time applications. Recent works have focused on improving training [1, 2, 20, 24, 30, 39, 57] and rendering [5, 10, 12, 34, 35, 54] efficiency. DVGO [39] and Plenoxels [57] replace the global MLP in NeRF with a sparse 3D grid and tiny global MLP or explicit spherical harmonics. Instant-NGP [30] further adopts multi-resolution hash encoding for compactness. To enable real-time rendering, "baking" methods pre-compute and store the neural field properties to efficient representations such as sparse voxel grids [12], triplanes [35], or hierarchical data structures (octree [56] and VDB [10]). Another line of work aims to extract explicit meshes [5, 25, 54] and leverage the real-time rasterization pipeline. However, these approaches either cannot be applied to large scenes [5, 12, 54, 56] due to intractable memory usage, or degrade in quality when baked [10, 25, 56], and typically have long training times [5, 12, 54]. Concurrently, SVR [40] perform voxel rasterization but focuses on camera without supporting ray-based sensor models. In contrast, SaLF optimizes and renders efficiently for multi-sensor in large scenes without baking.

3DGS: 3D Gaussian Splatting (3DGS) [17] generates photorealistic renderings by representing scenes with oriented 3D Gaussians. 3DGS enables real-time rendering through point splatting and tile-based rasterization, greatly reducing training and inference time. 3DGS has been applied to 3D generation [55, 58] and camera simulation [4, 6, 16, 50, 61]. However, like other rasterization-based approaches, 3DGS assumes a pinhole camera model and does not support flexible "ray-based" rendering like NeRF, which limits its applicability in simulating rolling-shutter LiDARs and fish-eye cameras. Several concurrent works [3, 13, 22, 23, 36, 47] have extended 3DGS to support non-pinhole sensors by reformulating projection models or approximating sensor-specific effects. However, their sensor-specific solutions hinder generalization to new sensor configurations. Other concurrent works [29, 60] apply ray-tracing on 3DGS but lack compatibility with rasterization, In contrast, our unified representation supporting both rasterization and ray-tracing, enabling real-time rendering for various sensors.

Data-driven Sensor Simulation for Self-Driving: Traditional graphics-based [8, 37] simulation via game engines have been widely used for autonomy development, but have high domain gap due to differences in content, scene geometry, appearance, and rendering fidelity. Data-driven neural rendering approaches [31, 51] have attracted significant attention due to their photorealism and ability to reconstruct from diverse real world scenes. [21, 31, 42, 43, 51] use NeRFs to build compositional digital twins and decompose background (*e.g.*, road, building), actors and sky as separate MLPs, enabling realistic and controllable camera and LiDAR [42, 46, 51] simulation with single-pass driving data. However, these works usually require hours of GPU training for one 10 sec. driving snippet, and cannot perform real-time rendering. To address this issue, recent works [6, 50, 61] leverage a compositional 3DGS representation for real-time camera simulation but are restricted to pinhole cameras and cannot directly model ray-based phenomena. In contrast, we build the first self-driving neural sensor simulator that supports real-time rendering for complex cameras and LiDAR.

3. SaLF Sparse Local Fields

We present SaLF (Sparse Local Fields), a novel volumetric representation that supports efficient tile-based rasterization and flexible, high-fidelity ray-casting of complex scenes. In this section, we detail our scene representation (Sec. 3.1) and our rasterization and ray-casting rendering algorithms (Sec. 3.2), the coarse-to-fine densification strategy for compactness and efficient training (Sec. 3.3), and discuss how SaLF compares to NeRF and 3DGS (Sec. 3.4).

3.1. Representation

SaLF represents scenes using a sparse grid of local implicit fields that map global 3D coordinates and view-directions to spatial properties such as density and color. Let $\mathcal{V} \subset \mathbb{R}^3$ be a three-dimensional volume in an axis-aligned bounding box (AABB) with dimensions (V_h, V_w, V_d). We partition $\mathcal{V}$ into a regular grid $\mathcal{G}$ of dimensions $\lceil V_h/s_0 \rceil \times \lceil V_w/s_0 \rceil \times \lceil V_d/s_0 \rceil$, where each voxel is a cube with edge length s_0. Each voxel supports recursive sub-division into 8 smaller voxels, up to K levels $(s_0 \dots s_k)$. To efficiently handle large-scale scenes, we employ a sparse representation that stores only non-empty voxels.

Each voxel is characterized by its static geometric parameters: position $p \in \mathbb{R}^3$, scale $s \in \mathbb{R}$, and rotation $q \in \mathbb{R}^4$. q represents the orientation relative to the global

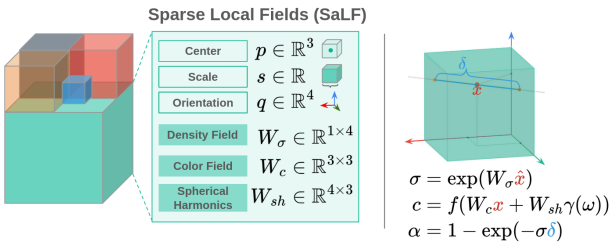

Figure 3. **SaLF Representation. Left:** SaLF models scenes using an adaptive sparse voxel grid with variable scales. Each voxel is characterized by static parameters and learnable parameters. **Right:** Within a voxel, for any point with normalized coordinates x, the density σ and color c values are derived from W_σ and W_c along with the encoded view direction $\gamma(\omega)$ modulating W_{sh}. The opacity α of a ray is calculated using the density at the intersection midpoint and the traversal distance δ.

coordinate system, with all voxels initialized to the identity quaternion in the global frame. Each voxel also contains a geometry field $W_\sigma \in \mathbb{R}^{1\times4}$ and a color field $W_c \in \mathbb{R}^{3\times3}$, along with 2nd order spherical harmonics $W_{sh} \in \mathbb{R}^{3\times4}$ for view-dependent lighting effects. For any point inside the voxel, let $\mathbf{x} \in [-1,1]^3$ denote its normalized local coordinates, and $\hat{\mathbf{x}} = [\mathbf{x}, 1]$ be its homogeneous representation. The geometry field $f_\sigma(\mathbf{x}; W_\sigma) : [-1,1]^3 \to \mathbb{R}_+$ computes the density as:

$$\sigma = f_\sigma(\mathbf{x}; W_\sigma) = \exp(W_\sigma \hat{\mathbf{x}}^T). \tag{1}$$

Similarly, the color field $f_c(\mathbf{x}, \omega; W_c, W_{\text{sh}}) : [-1,1]^3 \times \mathbb{S}^2 \to [0,1]^3$ computes the color value as:

$$c = \text{sigmoid}(W_c \mathbf{x}^T + W_{\text{sh}}\gamma(\omega)) \tag{2}$$

where $\omega \in \mathbb{S}^2$ is the view direction and $\gamma : \mathbb{S}^2 \to \mathbb{R}^4$ maps the view direction to spherical harmonics basis coefficients.

Volume rendering: We render the color $\hat{C}$ for each ray by accumulating the color and opacity values of intersected voxels along the ray, following the volume rendering equation as in NeRF:

$$\hat{C} = \sum_{i=1}^{N_c} T_i \alpha_i c_i, \tag{3}$$

where N_c is the number of intersected voxels along the ray, T_i represents the accumulated transmittance $T_i = \prod_{j=1}^{i-1}(1-\alpha_j)$ and α_i denotes the opacity computed from the density value σ_i and ray segment length δ_i within the i-th voxel:

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i). \tag{4}$$

The density σ_i and color c_i values are sampled at the midpoint of each ray-voxel intersection segment using Eq. (1) and Eq. (2), respectively.

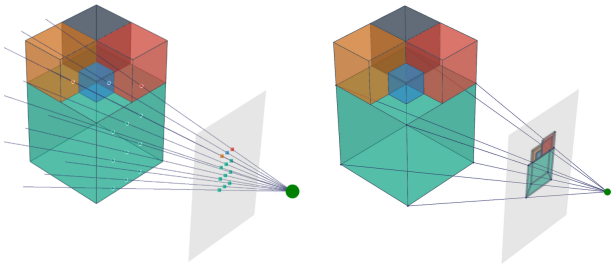

Figure 4. **SaLF can be efficiently rendered by ray-casting (left) and rasterization (right).** Ray casting is more flexible and can handle more complex physics phenomena, while rasterization is more efficient for pinhole cameras. Both follow the same rendering equation.

Surface parameterization: Autonomous driving scenes require high-quality surfaces to enable accurate LiDAR simulation and secondary ray effects such as reflection. Instead of directly representing density as W_σ, we follow previous approaches [11, 42, 51] to adopt a signed distance function (SDF) for better surface parameterization. The geometry SDF field denoted as W_s, quantifies the signed distance ($s_\pm$) between a point $\mathbf{x}$ and the surface as:

$$s_\pm = W_s \hat{\mathbf{x}}^T \tag{5}$$

We transform $s_\pm$ to density σ following VolSDF [38, 53]:

$$\sigma = \frac{a}{2} + \frac{a}{2}\text{sign}(s_\pm)\left(1 - e^{-|s_\pm|/b}\right). \tag{6}$$

The final learnable parameters of each voxel are the geometry field W_s, the color field W_c, the spherical harmonics W_{sh}, and the SDF-to-density parameters a, b.

3.2. Efficient Rendering of SaLF

As shown in Fig. 4, SaLF is interoperable and can be rendered via ray-casting through the volume and computing ray-voxel intersections, or by splatting and compositing voxels onto the image plane (*i.e.*, rasterization). Both can be implemented following the volume rendering equation in Eq. (3), with rasterization being faster but making more approximations in the image formation process. The ray-casting approach is more flexible and can handle more complex physics phenomena and sensor models. We now discuss in more detail our efficient implementations of each.

Ray-Casting with Octree Acceleration: SaLF can be rendered as NeRF by casting rays through the voxels from the sensor origin, sampling points, and accumulating their color and opacity. To accelerate ray-marching and ray-voxel intersection checks, we employ an octree data structure. The octree recursively partitions the volume into eight sub-volumes, where non-leaf nodes maintain pointers to their sub-volumes, and leaf nodes either store -1 for empty

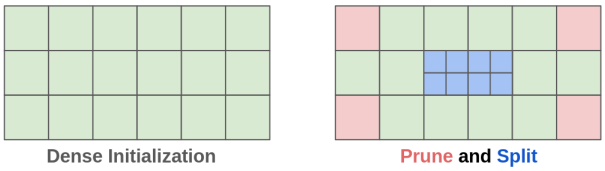

Figure 5. **Initialization and Densification.** SaLF initializes the scene representation with a coarse regular grid partitioning, then adaptively prune empty region while densifying regions that need fine-details.

space or a pointer to the corresponding voxel. This hierarchical structure enables fast ray traversal through the volume. Through a single ray-box intersection test, empty regions can be bypassed. Upon encountering non-empty nodes, traversal selectively descends into intersected child octants with logarithmic complexity.

Tile-based Splatting: SaLF can also be rendered by splatting, which projects voxels onto the image plane, compositing them with alpha blending. Following 3DGS, we divide the image plane into a grid of 16×16 tiles. For each tile, we perform view-frustum culling to identify and sort relevant voxels. Each tile is rendered by a thread block to iterate over the relevant voxels. Crucially, we preload these voxels into shared memory to reduce global memory access, which is a key optimization that significantly accelerates rendering. The opacity for each voxel is computed based on current pixel's ray travel distance within each voxel using Eq. (4), and the color is sampled from the color field at the intersection point. For a pinhole camera with primary rays, the splatting achieves significant acceleration through tile-based processing and shared memory. Please refer to supp. for more details.

3.3. Initialization and Densification

Naive uniform voxelization of large-scale scenes at high resolution leads to prohibitive memory consumption that scales cubically with scene size. To address this challenge, we adopt a coarse-to-fine approach as shown in Fig. 5. During training, we initialize the scene with a coarse representation and apply an adaptive densification and pruning strategy. Voxels exhibiting significant color field gradients are subdivided into eight child voxels in an octree-aligned manner. Voxels with negligible opacity values are removed from the sparse set. Densification and pruning together enables preserving of fine details while maintaining a compact memory footprint.

3.4. Comparison with 3DGS and NeRF

SaLF represents implicit scenes using discrete volumes like other voxel-based NeRF variants such as DVGO [39] and Plenoxels [57]. However, these methods are memory-intensive and slow in high-resolution (*e.g.*, 1920×1080) rendering due to dense voxel grids and global MLPs, compared to SaLF with its more compact sparse representation, efficient splatting, and octree-accelerated structure. We also extend these voxel-like representations to rasterization.

Both SaLF and 3DGS perform efficient splatting through sparse scene representations. But 3DGS does not support ray-based rendering directly because multiple Gaussians can contribute to the same point when overlapped. In contrast, SaLF defines distinct implicit functions for non-overlapping regions, enabling straightforward ray-voxel intersection and property evaluation. Furthermore, 3DGS often struggles with limited surface quality due to disconnected semi-transparent Gaussian primitives and often requires significant regularization [7, 15], while SaLF can leverage established NeRF techniques such as SDF for surface reconstruction. The initialization and densification strategies also differ: 3DGS requires sparse points as initialization and prunes, splits or clones existing Gaussians to cover both spatial extent and fine details. In contrast, SaLF is initialized coarsely yet densely, and its hierarchically subdividing voxels automatically capture fine details while maintaining spatial coverage.

4. Self-driving Sensor Simulation with SaLF

SaLF's efficient and versatile rendering capabilities are particularly well-suited for self-driving sensor simulation, which involves large scenes and dynamic actors, while demanding real-time rendering and multi-sensor support. We now present how to utilize SaLF to construct a lightweight simulator that achieves real-time rendering for camera and LiDAR sensors.

4.1. Compositional Scene Representation

Dynamic Scene Modelling: Following previous work [31], we model dynamic actors and backgrounds as distinct bounding volumes that we compose into a global frame for rendering. For dynamic actors, we initialize voxel sets by partitioning each actor's canonical bounding box. For rasterization, we use actor labels to transform these dynamic voxels to the global coordinate system at each timestamp, combine them with the static voxels, and project all voxels to the image plane. For ray-casting, we construct a separate octree for each dynamic actor. We precompute ray-box intersections between rays and the bounding boxes of dynamic actors, sorting the entry and exit points by camera distance. These precomputed intersections allow us to determine the traversal order between the static scene's octree and the dynamic actors' octrees, enabling efficient ray marching.

Multi-scale Static Scene Initialization: The outdoor driving environment necessitates efficient representation of

large-scale scenes (*e.g.*, sky, far-away buildings). Our initialization strategy begins by identifying a core region of interest that the self-driving vehicle will traverse, which we discretize at a base resolution. Recognizing that distant scene elements do not require high-resolution voxels, we surround the core static foreground region with increasingly coarser outer regions. These outer regions extend at $2\times$, $4\times$, $8\times$, and $16\times$ the base volume, with their voxel sizes scaling proportionally, naturally matching the diminishing detail requirements of distant scene elements. We leverage LiDAR point clouds to optimize scene initialization through a three-step process: inner region voxel pruning based on point absence, subdivision of point-containing voxels, and high-opacity initialization for point-occupied voxels. Please refer to the supp. for more details.

4.2. Learning

The scene representation is optimized through the following loss function:

$$\mathcal{L} = \mathcal{L}_{\text{color}} + \lambda_1 \mathcal{L}_{\text{depth}} + \lambda_2 \mathcal{L}_{\text{reg}} \tag{7}$$

where $\mathcal{L}_{\text{color}}$ and $\mathcal{L}_{\text{depth}}$ measure the ℓ_1 distance between rendered and ground-truth images and LiDAR depth, respectively. $\mathcal{L}_{\text{reg}}$ are regularization terms including enforcing spatial consistency between adjacent voxels by maintaining smooth SDF and color transitions and reduced opacity in outer regions to facilitate subsequent pruning. Please refer to the supp. for more details.

5. Experiments

5.1. Experimental Setup

Dataset and Evaluation Protocol: We evaluate our method on the PandaSet dataset [49], which consists of 103 driving scenes captured at 1920×1080 resolution, with 80 frames per scene and 360-degree LiDAR data. PandaSet has diverse and complex urban scenes. Following previous works [42, 51], we use the same evaluation set containing 10 logs, with 40 frames for training and 40 frames for evaluation per log. This 50%/50% split is more challenging than the typical 90%/10% setting in other methods [18, 61]. We use even frames for training and odd frames for evaluation. We report photorealism on the front camera via PSNR, SSIM, and LPIPS (VGG-backbone [59]) metrics. For LiDAR evaluation, we measure the median distance error (L_1) between the predicted and ground-truth depth[1]. The training and rendering speed are reported on an RTX 3090 averaged over the evaluation set.

Implementation Details of SaLF: We leverage the parallel computing framework `Taichi` [14, 41] to implement efficient rasterization and ray-casting operations. For training, we employ the Adam optimizer [19] with an initial learning rate of 0.01 and apply a decay factor of 0.8 every 800 iterations, for a total of 3200 iterations. The maximum number of voxels allowed is 2.5 million. We also train "SaLF (large)," with more voxels (5 million) and iterations (4500) for better performance. Please see supp. for details.

Comparison with SoTA methods: We compare our approach against several SoTA methods in self-driving sensor simulation including UniSim [52], NeuRAD [42] and Street Gaussian [61]. UniSim leverages compositional neural feature fields to model dynamic scenes for controllable camera and LiDAR simulation. NeuRAD further extends it to handle more complex sensor phenomena (*e.g.*, anti-aliasing, rolling-shutter, ray-dropping). Street Gaussian replaces NeRFs with compositional 3DGS and achieves real-time camera simulation, but does not support LiDAR.

5.2. Fast and Realistic Multi-Sensor Simulation

Comparison to NeRF-based sensor simulation: We compare SoTA sensor simulation approaches on PandaSet, as shown in Tab. 1, our method achieves the best balance between rendering realism, rendering speed, and reconstruction time, while also supporting ray-based sensor models. We achieve real-time rendering for both camera and LiDAR with comparable fidelity to SoTA simulators, and accelerate reconstruction speed by at least $5\times$. NeuRAD has slightly higher camera realism, as it leverages a post-processing CNN, which provides additional model capacity. We find that for NeuRAD without the CNN, SaLF achieves similar camera realism, demonstrating that the sparse voxel representation is as powerful as the NeRF-based neural feature grids, while being faster. Please refer to supp. for more details. Visual comparisons presented in Fig. 6 show that SaLF achieves similar realism compared to previous SOTA methods. We do note that the baselines have better visual quality for dynamic actors, potentially due to their actor pose optimization during training. For LiDAR, SaLF only has 2.5cm higher median error compared to NeuRAD, while being *100× faster*. Fig. 7 shows qualitatively similar point clouds w.r.t. the ground-truth.

Comparison to 3DGS-based sensor simulation: As shown in Fig. 9, our method can reconstruct distant regions more accurately compared to Street Gaussian. This is because Street Gaussian relies on sparse Structure-from-Motion points and LiDAR for initialization, which does not cover distant regions well. In contrast, our method's multi-scale initialization of coarse voxels provides more complete coverage.

Efficiency Analysis: We analyze SaLF's performance characteristics across different operating conditions in

[1] NeuRAD [42] reports LPIPS(AlexNet) and median L_2 LiDAR error.

Table 1. **Comparison to SoTA sensor simulation methods.** Our approach achieves real-time (> 30 FPS) camera and LiDAR rendering, and accelerates the reconstruction process by at least 5× while achieving comparable realism compared to baselines. Street Gaussian leverages 3DGS and does not support LiDAR simulation (✗). NeuRAD leverages NeRF and adopts additional CNN decoder for higher-quality. Only compared with the best-performing methods and ignore the rest (e.g. NSG, vanilla iNGP and 3DGS) for brevity. We highlight first , second , third .

Models	Rendering FPS ↑		Recon Time ↓	Rendering Realism		
	Camera	LiDAR	RTX-3090 hour	PSNR↑	SSIM↑	LiDAR-L1↓
Street Gaussian [50]	**115.5**	✗	2.26	25.65	**0.777**	✗
UniSim [51]	1.3	11.8	1.67	25.63	0.745	0.100
NeuRAD [42] (2x)	1.7	3.79	3.48	**26.60**	0.770	**0.085**
SaLF (base)	54.5	**640**	0.31	25.48	0.744	0.142
SaLF (large)	34.3	430	0.48	25.78	0.762	0.111

Figure 6. **Qualitative comparison on camera novel view synthesis.** We achieve comparable photorealism compared to SoTA approaches.

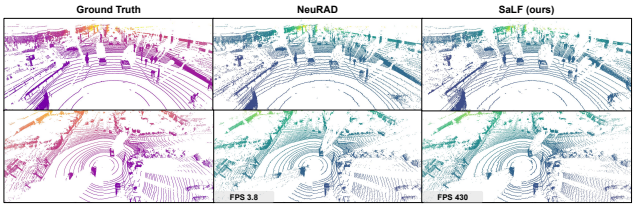

Figure 7. **Qualitative comparison on LiDAR novel view synthesis.** Our method achieves comparable LiDAR rendering performance compared to NeuRAD, while being *100 × faster* in rendering.

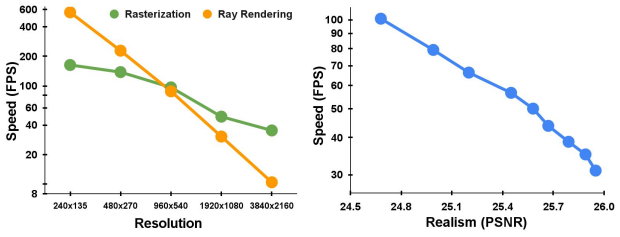

Figure 8. **Efficiency analysis.** Left: rendering speed comparison of rasterization *vs.* ray rendering across resolutions. Right: trade-off between rendering speed and realism.

Fig. 8. Our analysis reveals that at lower resolutions (below 960 × 540), ray rendering significantly outperforms rasterization due to the latter's resolution-independent overhead from projection and sorting operations. As resolution increases, rasterization becomes the more efficient approach, demonstrating the complementary nature and practical value of supporting both rendering paradigms within a single representation. We further analyze the relationship between rendering efficiency and quality, finding a consistent trade-off where higher FPS results in slight PSNR reduction, enabling users to select optimal configurations based on their application requirements.

Table 2. **Ablation study on SaLF components.**

Models	PSNR↑	SSIM↑	LPIPS↓
Ours	**25.48**	**0.744**	**0.373**
– Densification	23.19	0.670	0.474
– Field matrices	25.11	0.735	0.386

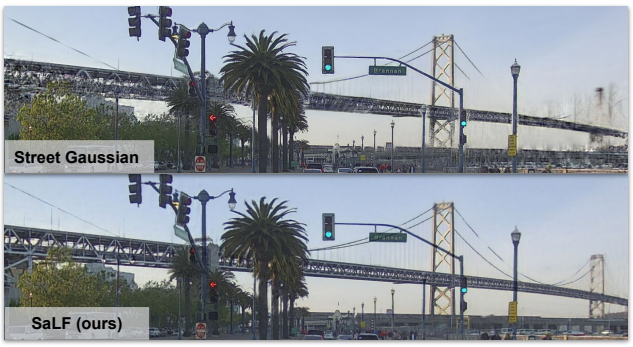

Figure 9. **Comparison with StreetGaussian on distant regions.** Our method provides more accurate reconstructions for distant regions (e.g. bridge) where LiDAR and SfM points are not sufficient.

Ablation Study: Two key aspects of SaLF are its local implicit field representation as a matrix compared to a fixed scalar value per voxel, and its adaptive voxel pruning and densification during optimization. Tab. 2 reports the camera realism results, demonstrating the value of both choices.

5.3. Applications and Extensions

Versatile rendering capabilities. To demonstrate our method's versatile rendering capabilities, we showcase several key capabilities in Fig. 2. These include rolling shutter LiDAR effects, ray-based light phenomena such as refraction and shadows of inserted objects, and panorama camera images. These demonstrations highlight the versatility of our approach in handling various sensor rendering tasks.

Rolling-shutter. In high-speed driving scenarios, the temporal motion of the self-driving vehicle and other actors during sensor capture can significantly impact perception [27]. In Fig. 10, we demonstrate our method's ability to accurately simulate LiDAR and camera rolling-shutter effects, a crucial capability for simulation.

Further extension. SaLF provides the foundation for a performant data-driven multi-sensor renderer that can be further built upon for downstream simulation, where additional features are needed. To demonstrate this, we show initial demonstrations of extending SaLF to support additional LiDAR features such as raydrop and intensity simulation, and multi-camera rendering for panorama image generation. For more details, please refer to the supp. We believe SaLF can further extend to support more sophisticated

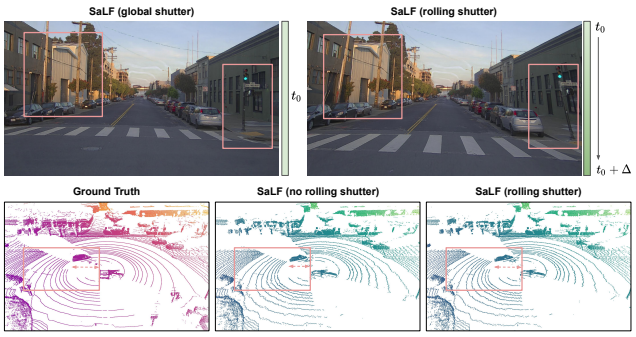

Figure 10. **Rolling-shutter simulation via efficient ray-based rendering. Top:** We render the same view using global shutter and rolling-shutter camera models (see highlighted distorted region). **Bottom:** SaLF simulates rolling-shutter effect and accurately match ground truth point clouds (see lidar sweep seam and relative position for the dynamic actor in the highlighted region.)

features like beam-divergence, actor-label and sensor pose refinement, providing exciting directions for future work for the community.

5.4. Limitations

Our method typically requires a higher number of voxels compared to 3DGS-based StreetGaussian to achieve comparable rendering quality. This stems from our voxels having fixed size, position, and orientation, in contrast to 3DGS's adaptive Gaussian primitives that can dynamically adjust their shape to efficiently represent regions of similar appearance. We also note that additional modifications are required to support non-rigid and temporal changes in our scene representation [6, 9]. While our method supports full raytracing, and we demonstrate phenomena such as shadow in Fig. 2, we train SaLF using primary rays only.

6. Conclusion

In this work, we tackled the problem of developing a multi-sensor simulation system that is fast to train, realistic, and efficient to render with. Towards this goal, we proposed a novel representation, SaLF, which consists of a set of sparse voxels, and where each voxel defines a local implicit field. Importantly, we design our representation to support both rasterization and raycasting, enabling support of ray-based phenomenon such as rolling-shutter, shadows and refraction, as well as sensors with distorted lenses, which was previously difficult to achieve with 3DGS. We enhance SaLF for driving scenes via multi-scale voxel initialization, adaptive pruning and densification, and dynamic actor modelling. We demonstrated that SaLF achieves comparable LiDAR and camera realism to existing neural rendering simulation methods, while being much faster to train (up to 5×) and render with (over 100× for LiDAR), enabling more scalable sensor simulation for autonomy development.

Acknowledgement

We thank the Waabi team for their valuable assistance and support. Especially Rui Hu on the support of GPU programming and Andrei Bârsan on the discussion and feedback on the paper.

References

[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. 3

[2] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *ECCV*, 2022. 3

[3] Qifeng Chen, Sheng Yang, Sicong Du, Tao Tang, Peng Chen, and Yuchi Huo. LiDAR-GS: Real-time LiDAR re-simulation using gaussian splatting. *arXiv*, 2024. 3

[4] Yurui Chen, Chun Gu, Junzhe Jiang, Xiatian Zhu, and Li Zhang. Periodic vibration gaussian: Dynamic urban scene reconstruction and real-time rendering. *arXiv*, 2023. 3

[5] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv*, 2022. 1, 3

[6] Ziyu Chen, Jiawei Yang, Jiahui Huang, Riccardo de Lutio, Janick Martinez Esturo, Boris Ivanovic, Or Litany, Zan Gojcic, Sanja Fidler, Marco Pavone, et al. OmniRe: Omni urban scene reconstruction. *arXiv*, 2024. 3, 8

[7] Kai Cheng, Xiaoxiao Long, Kaizhi Yang, Yao Yao, Wei Yin, Yuexin Ma, Wenping Wang, and Xuejin Chen. Gaussianpro: 3D gaussian splatting with progressive propagation. *arXiv*, 2024. 5

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *Conference on robot learning*, 2017. 3

[9] Tobias Fischer, Lorenzo Porzi, Samuel Rota Bulo, Marc Pollefeys, and Peter Kontschieder. Multi-level neural scene graphs for dynamic urban environments. In *CVPR*, 2024. 8

[10] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-fidelity neural rendering at 200fps. *ICCV*, 2021. 3

[11] Jianfei Guo, Nianchen Deng, Xinyang Li, Yeqi Bai, Botian Shi, Chiyu Wang, Chenjing Ding, Dongliang Wang, and Yikang Li. Streetsurf: Extending multi-view implicit surface reconstruction to street views. *arXiv*, 2023. 1, 4

[12] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021. 1, 3

[13] Georg Hess, Carl Lindström, Maryam Fatemi, Christoffer Petersson, and Lennart Svensson. Splatad: Real-time lidar and camera rendering with 3d gaussian splatting for autonomous driving. *arXiv preprint arXiv:2411.16816*, 2024. 3

[14] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *TOG*, 2019. 6

[15] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 Conference Papers*, 2024. 5

[16] Nan Huang, Xiaobao Wei, Wenzhao Zheng, Pengju An, Ming Lu, Wei Zhan, Masayoshi Tomizuka, Kurt Keutzer, and Shanghang Zhang. S3gaussian: Self-supervised street gaussians for autonomous driving. *arXiv*, 2024. 3

[17] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *TOG*, 2023. 2, 3

[18] Mustafa Khan, Hamidreza Fazlali, Dhruv Sharma, Tongtong Cao, Dongfeng Bai, Yuan Ren, and Bingbing Liu. Autosplat: Constrained gaussian splatting for autonomous driving scene reconstruction. *arXiv preprint arXiv:2407.02598*, 2024. 6

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 6

[20] Juuso Korhonen, Goutham Rangu, Hamed R Tavakoli, and Juho Kannala. Efficient NeRF optimization–not all samples remain equally hard. *arXiv*, 2024. 3

[21] 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 *CVPR*, 2022. 3

[22] Pou-Chun Kung, Xianling Zhang, Katherine A Skinner, and Nikita Jaipuria. Lihi-gs: Lidar-supervised gaussian splatting for highway driving scene reconstruction. *arXiv preprint arXiv:2412.15447*, 2024. 3

[23] Zimu Liao, Siyan Chen, Rong Fu, Yi Wang, Zhongling Su, Hao Luo, Li Ma, Linning Xu, Bo Dai, Hengjie Li, et al. Fisheye-gs: Lightweight and extensible gaussian splatting module for fisheye cameras. *arXiv preprint arXiv:2409.04751*, 2024. 3

[24] Haotong Lin, Sida Peng, Zhen Xu, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Efficient neural radiance fields for interactive free-viewpoint video. In *SIGGRAPH Asia 2022 Conference Papers*, 2022. 3

[25] Jeffrey Yunfan Liu, Yun Chen, Ze Yang, Jingkang Wang, Sivabalan Manivasagam, and Raquel Urtasun. Real-time neural rasterization for large scenes. In *ICCV*, 2023. 1, 3

[26] Alexander Mai, Peter Hedman, George Kopanas, Dor Verbin, David Futschik, Qiangeng Xu, Falko Kuester, Jonathan T Barron, and Yinda Zhang. Ever: Exact volumetric ellipsoid rendering for real-time view synthesis. *arXiv preprint arXiv:2410.01804*, 2024. 20

[27] 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*, 2023. 2, 8

[28] 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. *ECCV*, 2020. 1, 3

[29] 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. In *SIGGRAPH Asia 2024*, 2024. 3

[30] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. 2022. 3

[31] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. *CVPR*, 2021. 1, 3, 5

[32] Ava Pun, Gary Sun, Jingkang Wang, Yun Chen, Ze Yang, Sivabalan Manivasagam, Wei-Chiu Ma, and Raquel Urtasun. Neural lighting simulation for urban scenes. In *NeurIPS*, 2023. 2

[33] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering. *ACM Transactions on Graphics (TOG)*, 43(4):1–17, 2024. 20

[34] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. *ICCV*, 2021. 3

[35] Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P. Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T. Barron, and Peter Hedman. MERF: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *arXiv*, 2023. 2, 3

[36] Yuan Ren, Guile Wu, Runhao Li, Zheyuan Yang, Yibo Liu, Xingxin Chen, Tongtong Cao, and Bingbing Liu. Unigaussian: Driving scene reconstruction from multiple camera models via unified gaussian representations. *arXiv preprint arXiv:2411.15355*, 2024. 3

[37] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, 2018. 3

[38] Yawar Siddiqui, Tom Monnier, Filippos Kokkinos, Mahendra Kariya, Yanir Kleiman, Emilien Garreau, Oran Gafni, Natalia Neverova, Andrea Vedaldi, Roman Shapovalov, et al. Meta 3d assetgen: Text-to-mesh generation with high-quality geometry, texture, and pbr materials. *arXiv*, 2024. 4

[39] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. *CVPR*, 2022. 3, 5

[40] Cheng Sun, Jaesung Choe, Charles Loop, Wei-Chiu Ma, and Yu-Chiang Frank Wang. Sparse voxels rasterization: Real-time high-fidelity radiance field rendering. *arXiv preprint arXiv:2412.04459*, 2024. 3, 20

[41] Kuangyuan Sun. Taichi 3D Gaussian Splatting, 2023. 6, 12

[42] Adam Tonderski, Carl Lindström, Georg Hess, William Ljungbergh, Lennart Svensson, and Christoffer Petersson. NeuRAD: Neural rendering for autonomous driving. In *CVPR*, 2024. 1, 3, 4, 6, 7, 15

[43] Haithem Turki, Jason Y Zhang, Francesco Ferroni, and Deva Ramanan. Suds: Scalable urban dynamic scenes. In *CVPR*, 2023. 1, 3

[44] Zian Wang, Wenzheng Chen, David Acuna, Jan Kautz, and Sanja Fidler. Neural light field estimation for street scenes with differentiable virtual object insertion. *ECCV*, 2022. 2

[45] Waymo. Meet the 6th generation waymo driver, 2024. 1

[46] Hanfeng Wu, Xingxing Zuo, Stefan Leutenegger, Or Litany, Konrad Schindler, and Shengyu Huang. Dynamic lidar resimulation using compositional neural fields. In *CVPR*, 2024. 3

[47] Qi Wu, Janick Martinez Esturo, Ashkan Mirzaei, Nicolas Moenne-Loccoz, and Zan Gojcic. 3dgut: Enabling distorted cameras and secondary rays in gaussian splatting. *arXiv preprint arXiv:2412.12507*, 2024. 3

[48] Zirui Wu, Tianyu Liu, Liyi Luo, Zhide Zhong, Jianteng Chen, Hongmin Xiao, Chao Hou, Haozhe Lou, Yuantao Chen, Runyi Yang, et al. MARS: An instance-aware, modular and realistic simulator for autonomous driving. *arXiv*, 2023. 1

[49] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *ITSC*, 2021. 6

[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. *arXiv*, 2024. 2, 3, 7, 15

[51] 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. 1, 3, 4, 6, 7, 15

[52] 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. 6

[53] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *NeurIPS*, 2021. 4

[54] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. BakedSDF: Meshing neural SDFs for real-time view synthesis. *arXiv*, 2023. 1, 3

[55] Taoran Yi, Jiemin Fang, Junjie Wang, Guanjun Wu, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Qi Tian, and Xinggang Wang. GaussianDreamer: Fast generation from text to 3D gaussians by bridging 2D and 3D diffusion models. In *CVPR*, 2024. 3

[56] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. *ICCV*, 2021. 2, 3

[57] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *CVPR*, 2022. 3, 5

[58] Kai Zhang, Sai Bi, Hao Tan, Yuanbo Xiangli, Nanxuan Zhao, Kalyan Sunkavalli, and Zexiang Xu. GS-LRM: Large reconstruction model for 3D gaussian splatting. In *ECCV*, 2025. 3

[59] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CVPR*, 2018. 6

[60] Chenxu Zhou, Lvchang Fu, Sida Peng, Yunzhi Yan, Zhanhua Zhang, Yong Chen, Jiazhi Xia, and Xiaowei Zhou. Lidar-rt: Gaussian-based ray tracing for dynamic lidar re-simulation. *arXiv preprint arXiv:2412.15199*, 2024. 3

[61] Xiaoyu Zhou, Zhiwei Lin, Xiaojun Shan, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. DrivingGaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes. In *CVPR*, 2024. 2, 3, 6

	3D Gaussian Splatting	SaLF
Parameters		
Center	$\mu \in \mathbb{R}^3$	$p \in \mathbb{R}^3$ (not learnable)
Rotation	$q \in \mathbb{R}^4$ (quaternion)	$q \in \mathbb{R}^4$ (quaternion, not learnable)
Scale	$s \in \mathbb{R}^3$ (anisotropic)	$s \in \mathbb{R}$ (not learnable)
Opacity/density	$\alpha \in \mathbb{R}$ (direct opacity)	$W_\sigma \in \mathbb{R}^{1\times4}$ (density field)
Appearance	SH coefficients for color	$W_c \in \mathbb{R}^{3\times3}$ (color field) + $W_{sh} \in \mathbb{R}^{3\times4}$ (SH)
Opacity Computation		
Computation	$\alpha \cdot \exp(-\frac{1}{2}x^T\Sigma'^{-1}x)$	$\alpha = 1 - \exp(-\sigma\delta)$
Input	- α: learned opacity - x: 2D offset from projected center - Σ': projected covariance	- $\sigma = \exp(W_\sigma x^T)$: density from field - δ: ray-voxel intersected segment - x: intersected midpoint
Dependency	Varies with distance from gaussian center	Varies with ray-voxel intersected point and length
Color Computation		
Base Formula	$c = \text{SH}(\gamma(\omega))$	$c = \text{sigmoid}(W_c x^T + W_{sh}\gamma(\omega))$
Inputs	- $\gamma(\omega)$: encoded viewdir - SH: spherical harmonic coefficients	- x: local coordinates - $\gamma(\omega)$: encoded viewdir
Structure	Single SH evaluation	Combines local color field + SH
Initialization and Densification		
Initialization Strategy	Start from sparse points	Dense coarse voxel grid
Densification Criteria	- Large position gradients - View-space gradient magnitude - Size of Gaussians	Color field gradients
Densification Operations	- Clone: copy Gaussian and move along gradient - Split: divide large Gaussians with reduced scale	Split into 8 children
Pruning Strategy	Remove low opacity Gaussians	Remove low opacity voxels

Table 3. **Comparison between 3DGS and SaLF.**

Appendix

In this appendix, we provide the implementation details of our method and the baselines, additional qualitative results and more discussions of limitations of SaLF. We first describe the implementation details of our approach in Sec. A and baselines in Sec. B. We further showcase additional qualitative examples in Sec. C. Finally, we analyze the limitations of our model and discuss the future works in Sec. D.

A. SaLF Details

Implementation Details for Tile-based Rasterization and comparison to 3DGS: For the tile-based rasterization, our implementation is built on taichi-gaussian-splatting [41]. The key implementation difference between 3DGS and SaLF lies in their opacity computation mechanisms. In 3DGS, opacity is computed using a gaussian function based on the distance between each 2D pixel and the projected gaussian center. In contrast, SaLF converts each pixel into a 3D ray and computes ray-voxel intersections to obtain the traversal distance (interval δ) and midpoint. The density is then computed using this midpoint, and opacity is derived using main paper Equation (4). Our rasterization speed matches the original 3DGS when handling a similar number of voxels. However, SaLF typically requires more voxels to achieve comparable rendering quality, which impacts overall rendering speed. This difference arises because 3DGS can fit scenes more efficiently with fewer gaussians due to its additional learnable parameters (rotation, scale, center) that adapt to scene complexity. In contrast, SaLF uses predefined static size, position, and orientation parameters, necessitating more voxels for high-fidelity scene representation. Additionally, while we maintain a dense coarser voxel grid for distant regions that contains relatively few voxels, each of these voxels has a larger size. This results in more projected voxels per tile during rasterization, affecting performance. We compare the difference between SaLF and 3DGS in Tab. 3.

Implementation Details for Ray-casting Octree-based ray marching combines hierarchical space partitioning with efficient memory layout for fast volume rendering. This includes two aspects: the linear octree storage format and the hierarchical ray marching algorithm. The algorithms boxes below outline the core components of octree-based ray marching.

Algorithm 1 Octree-based Ray Marching

Require: Ray origin $\mathbf{o}$, direction $\mathbf{d}$, octree buffer $\mathcal{B}$
Ensure: Final color and opacity (C_{out}, α_{out})
Initialize $\alpha_{accumulated} = 0$, $C_{out} = 0$, $\epsilon = 1e^{-4}$
Initialize current_pos $\leftarrow \mathbf{o}$
while $\alpha_{accumulated} < 0.99$ and ray in volume **do**
    $node \leftarrow$ QueryOctree($\mathcal{B}$, current_pos)
    if $node.is\_empty$ **then**
        $t_{exit} \leftarrow$ ComputeExitDist(current_pos, $\mathbf{d}$, voxel_bounds) ▷ Ray-node intersection
        current_pos $\leftarrow$ current_pos + (t_{exit} + ϵ)$\mathbf{d}$ ▷ Small offset ϵ to go out of current voxel
    else
        $(C_{voxel}, \alpha_{voxel}) \leftarrow$ GetVoxelData($node.id$) ▷ compute color and opacity
        $C_{out} \leftarrow C_{out} + (1 - \alpha_{accumulated})C_{voxel}\alpha_{voxel}$
        $\alpha_{accumulated} \leftarrow \alpha_{accumulated} + (1 - \alpha_{accumulated})\alpha_{voxel}$
        $t_{exit} \leftarrow$ ComputeExitDist(current_pos, $\mathbf{d}$, voxel_bounds)
        current_pos $\leftarrow$ current_pos + (t_{exit} + ϵ)$\mathbf{d}$
    end if
end while
return $(C_{out}, \alpha_{accumulated})$

Algorithm 2 Linear Octree Buffer Representation

Require: Input voxel data with positions and values
Ensure: Linear buffer $\mathcal{B}$ storing octree nodes
$\mathcal{B}[0] \leftarrow$ root node ▷ Each node stores (id_or_offset, is_leaf)
for each node **do**
    if node is empty **then**
        Store $(-1, -1)$ ▷ Empty node
    else if node has children **then**
        Store $(offset, 0)$ ▷ offset points to 8 children block
        Store 8 children in consecutive memory at $offset$
    else
        Store $(voxel\_id, 1)$ ▷ Leaf node with actual data
    end if
end for
return $\mathcal{B}$

Algorithm 3 QueryOctree

Require: Octree buffer $\mathcal{B}$, position $\mathbf{p}$
Ensure: Leaf node information
$node\_idx \leftarrow 0$ ▷ Start from root
while $node \leftarrow \mathcal{B}[node\_idx]$ is not leaf **do**
    if $node.offset = -1$ **then**
        return EmptyNode
    end if
    $child\_idx \leftarrow$ ComputeChildIndex($\mathbf{p}$) ▷ Based on octant position
    $node\_idx \leftarrow node.offset + child\_idx$
end while
return $node$

Algorithm 2 describes the linear octree buffer structure, which enables efficient memory access. Algorithm 1 shows the main ray marching process, which efficiently skips empty space. Algorithms 3 and 4 detail the octree traversal mechanics.

Algorithm 4 ComputeChildIndex

Require: Position $\mathbf{p}$ in current node's local space
Ensure: Child index in range [0,7]
$offset \leftarrow \mathbf{p} \geq 0.5$ ▷ Binary vector for each axis
return $offset.x + 2(offset.y + 2(offset.z))$ ▷ Convert to linear index

Initialization and Adaptive Densification: We first obtain an Axis-Aligned Bounding Box (AABB) that encompasses all actor trajectories in the world coordinate frame. We extend this boundary by 10m upward, 5m downward, and 40m laterally to define an inner region that typically contains dense LiDAR coverage. We partition this inner region into 1-meter voxels and extend it hierarchically by factors of 2x, 4x, 8x, and 16x to create outer regions, with voxel sizes scaled proportionally. To optimize the scene initialization, we leverage the aggregated static LiDAR point cloud data through a two-step process: first pruning voxels in the inner region that contain no LiDAR points, then subdividing voxels containing LiDAR points, thereby creating a multi-scale scene representation. For parameter initialization, voxels containing at least one LiDAR point are assigned high opacity values by setting parameter a in Equation (6) to 2.0, while voxels without LiDAR points are initialized with $a = 0.1$. Parameter b is initialized to 0.2 across all voxels. The learnable parameters (SDF Field, RGB Field, Spherical Harmonics) are randomly initialized using PyTorch's default settings in `nn.Linear`. The adaptive splitting and pruning process maintains a budget of M total voxels, with N voxels at the current step. We first prune N_{prune} voxels whose opacity remains below 0.005. We then use accumulated training gradients as a measure of local geometric complexity to identify the top $(M + N_{\text{prune}} - N)/(8 \times 5)$ voxels for splitting. The factor of 8 accounts for subdivision into sub-voxels, while the factor of 5 ensures gradual splitting to prevent premature budget allocation. Upon splitting, all eight sub-voxels inherit identical color and geometry fields from their parent voxel.

Training: Our loss formulation comprises of an L_1 color loss (weight: 1.0) and an L_1 depth loss (weight: 10.0), along with regularization terms. Our regularization framework consists of multiple weighted loss terms, all utilizing the L_1 norm. The Eikonal loss (weight: 0.1) enforces the SDF field property by constraining the norm of the first three W_s components to unity (for $f = ax + by + cz + d$, ensuring $a^2 + b^2 + c^2 = 1$). The smoothness loss (weight: 3.0) minimizes the difference between color and geometry fields of neighboring voxel pairs along their shared plane. For opacity regularization (weight: 10.0), we enforce opacity values close to 1 for LiDAR points by computing opacity at 20cm traversal distance using queried SDF values. The empty space loss (weight: 0.1) encourages the lowest 20% of voxel opacity values in the outer region to approach 0 at maximum traversal distance.

Implementation Details for Applications: For *panoramic visualization*, we first train our model using all 6 cameras and then render 360° views through ray-casting. *Rolling shutter* and *motion blur* effects occur when the sensor is moving with respect to parts of the scene. We simulate those by computing individual per-ray origin points based on the sensor's velocity and the exact timestamp when each ray of the sensor is registered. We also interpolate the poses of dynamic objects according to the individual ray timestamps. Our method's ability to cast individual camera rays also allows us to inject new objects into the scene with *reflections*, *refraction*, and *shadows*. For the glass sphere demo, we cast additional refracted and reflected rays, then blend their contributions according to the Fresnel formula. The scene lighting present in the training images is baked into our representation and disentangling lighting from surface material parameters is outside the scope of this paper. Nevertheless, we are able to simulate a shadow of a synthetically injected object if we assume that most of the light comes from the sun located high above the scene. For the shadow demo, we first use the per-ray depth to determine the 3D position of the scene surface. If the shadow ray cast from the surface to the sun intersects the injected object, we darken the original output color. Please note that the injected spheres in the demonstrations are not themselves represented using SaLF, but instead we solve the ray-sphere intersection equation. The SaLF-represented scene itself is not reflective, as estimating the material properties is outside the scope of this paper.

Efficiency Analysis. For the efficiency-resolution analysis, we evaluate the SaLF-base model (with 2.2 million voxels) on log *001*. The ray-rendering throughput remains relatively consistent across resolutions, with only slight reduction at lower resolutions due to the fixed overhead of preprocessing dynamic objects. In contrast, rasterization throughput decreases more significantly at lower resolutions because operations like projection and sorting incur resolution-independent overhead, becoming proportionally more expensive for smaller images. For the efficiency-realism analysis, we evaluate log *001* with

varying model capacities (1.2, 1.6, 2.2, 3.0, 3.8, 4.7, 6.2, 8.2, 10.7 million voxels) to measure the quality-performance tradeoff.

Storage and Memory. SaLF-big requires 807MB of disk storage, comparable to StreetGaussian (760MB). SaLF's sparse multi-scale voxel approach provides better memory efficiency than dense single-scale methods like Plenoxels (which use sparse features but still require dense grids for indexing). Despite using different primitive types, SaLF and StreetGaussian have similar parameter counts (181M vs. 192M).

Hardware Performance. On an RTX-3090 rendering 1080P images, SaLF-large achieves 16.7 FPS with 1.6GB memory using ray-tracing (approximately 10× faster than NeRF-based approaches due to our accelerated structure), and 34.3 FPS with 3.1GB memory using rasterization. Interestingly, on newer hardware NVIDIA L40S, ray-tracing actually outperforms rasterization (61.6 FPS vs. 47.9 FPS). This performance inversion likely results from the L40S having superior compute capabilities but lower memory bandwidth compared to the 3090. Rasterization is more memory-bound due to operations like tile creation, culling, duplication, and sorting, while our ray-tracing implementation benefits from kernel fusion that fully utilizes compute resources.

B. Baseline Implementation Details

UniSim [51]: UniSim is a state-of-the-art (SoTA) neural sensor simulator which builds modifiable digital twins with compositional neural radiance fields. It achieves SoTA novel view synthesis and simulation performance in both camera and LiDAR simulation and has been applied in closed-loop evaluation. We follow the same evaluation setting as [51] and copy the numbers from the original paper. We note that UniSim [51] speed numbers are measured on RTX-A5000, which is approximately 1.2x slower than RTX-3090 used in this paper.

NeuRAD [42]: NeuRAD extends UniSim to handle more complex sensor phenonmena (*e.g.*, rolling-shutter, ray-dropping and beam divergence) and achieves superior performance in camera and LiDAR simulation with 2× training time. We adopt the public implementation[1]the continuously improving version in the codebase. For NeuRAD w/o CNN comparison, we replace the CNN decoder with a three-layer MLP with ReLU activation. Note that we have some minor difference compared to NeuRAD in the settings and evaluation metric. First, we split data in odd/even frames for training/validation (0,2,4,...76,78) VS (1,3,9,....,77,79), while NeuRAD has slightly difference for the last frame (0,2,4,...,76, **79**) VS (1,3,5,...,77, **78**). Second, we use LPIPS with VGG backbone for camera evaluation, while NeuRAD uses AlexNet backbone. Third, we use median L_1 LiDAR error (m) for evaluation, while NeuRAD uses median L_2 LiDAR error (m^2). So we rerun the evaluation with the same settings.

Street Gaussian [50]: Street Gaussian replaces NeRFs in UniSim or NeuRAD with compositional 3DGS and achieves real-time camera simulation, but does not support LiDAR. We adopt the public implementation[2] and train the models for 30,000 iterations with the default hyperparameters (*e.g.*, density control, learning rate schedule). We use 800,000 downsampled aggregated LiDAR points and random 200,000 points for the initialization of 3D Gaussians.

C. Additional Experiments

C.1. Representation Comparison

We further analyze the performance differences between different representation approaches for multi-sensor rendering. Table 4 compares our SaLF (large) with NeuRAD (without CNN post-processing) and Street Gaussian to isolate the impact of the underlying representations rather than additional enhancements. This comparison highlights the trade-offs between different scene representations in terms of rendering efficiency, reconstruction time, and rendering quality.

As shown in Table 4, SaLF offers significant advantages in reconstruction time (7.1× faster than NeuRAD and 4.7× faster than Street Gaussian) while maintaining competitive rendering realism. For camera rendering speed, SaLF falls between the highly efficient Street Gaussian and the slower NeRF-based NeuRAD, while providing LiDAR simulation capabilities that Street Gaussian lacks. This analysis demonstrates that our sparse voxel-based representation with local implicit fields effectively bridges the gap between explicit particle-based methods (3DGS) and pure implicit field approaches (NeRF).

[1] https://github.com/georghess/neurad-studio
[2] https://github.com/ziyc/drivestudio

Table 4. **Comparison of different scene representations for multi-sensor rendering.**

Models	Rendering FPS ↑		Recon Time ↓	Rendering Realism		
	Camera	LiDAR	RTX-3090 hour	PSNR↑	SSIM↑	LiDAR-L1↓
Street Gaussian	**115.5**	✗	2.26	25.65	**0.777**	✗
NeuRAD (w/o CNN)	0.2	3.80	3.42	25.64	0.719	**0.090**
SaLF (large)	34.3	**430**	**0.48**	**25.78**	0.762	0.111

Models	Intensity	Raydrop Acc	Speed (fps)
SaLF-large	0.069	92.7	**350**
NeuRAD	**0.062**	**96.2**	3.79
UniSim	0.085	91.0	11.8

Table 5. **Extension SaLF for LiDAR Raydrop and intensity.**

Models	PSNR	SSIM	LPIPS
StreetGS	24.73	0.745	0.314
NeuRAD(w/ CNN)	**25.80**	**0.753**	**0.250**
UniSim	23.12	0.682	0.360
SaLF-large	24.80	0.732	0.351

Models	FID
NeuRAD w/ CNN	**29.52**
NeuRAD w/o CNN	40.84
StreetGS	37.92
SaLF-large	41.22

Table 6. **Multi-camera performance** and **extrapolation**.

C.2. Extension for LiDAR raydrop and intensity

For more LiDAR feature like raydrop and intensity simulation, we extend SaLF with a 8-channel feature for each voxel. The alpha-blended feature, along with depth and viewdir, is converted to intensity and raydrop probability with a linear layer. Preliminary results show SaLF has reasonable performance while being significantly faster (Tab. 5). We also extended SaLF to support multi-camera rendering, which can then be used to generate panorama images. More sophisticated features like beam-divergence, actor-label refinement and multi-sensor calibration can be built on top of SaLF for various applications scenarios, which we leave for future work.

C.3. Evaluation for multi-cameras and extrapolation

Multi-camera reconstruction in Pandaset presents challenges due to significant exposure differences across cameras. To address this issue, we implement per-camera spherical harmonics while maintaining shared geometry information. Table 6 presents our preliminary results, which demonstrate performance comparable to StreetGaussian, though still behind NeuRAD with its CNN-enhanced capacity. Future work could incorporate more sophisticated modeling approaches, including exposure correction and camera calibration, to further enhance performance. We also evaluate extrapolation performance by displacing the camera $\pm$2m along the XY-axis and computing the FID between rendered and source images. As shown in Table 6, SaLF and StreetGS achieve similar extrapolation quality, while NeuRAD demonstrates greater robustness due to its CNN post-processing.

C.4. Additional Qualitative Results

Our method achieves the best balance between rendering realism, rendering speed, and reconstruction time, while also supporting ray-based sensor models. As shown in Fig. 11 and 12, we achieve real-time rendering for both camera and LiDAR with comparable fidelity to SoTA simulators on a wide variety of driving scenarios.

C.5. Additional Examples for Controllable Simulation

We further showcase the capacities of SaLF in controllable simulation. In Fig. 13, we show examples of removing all dynamic actors seamlessly from the original scenes. In Fig. 14, we show high-fidelity simulation examples of SDV manipulation by moving the front camera left or right. These results demonstrate SaLF has comparable realism and capacities as SoTA

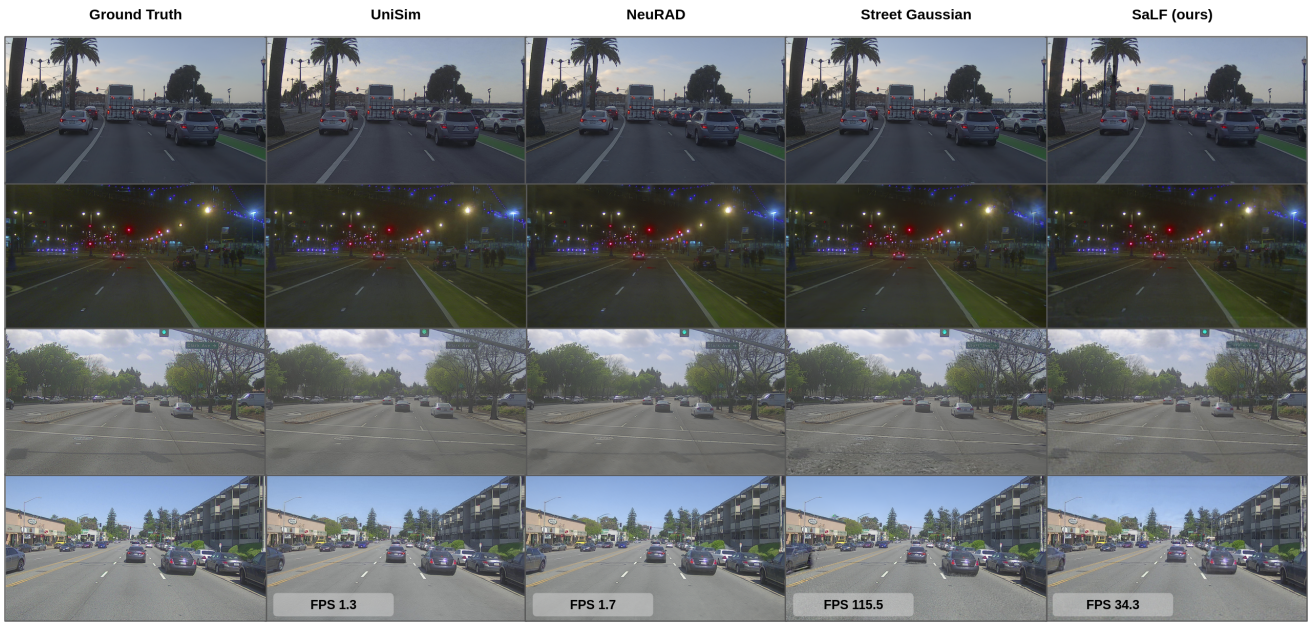

Figure 11. **Additional qualitative comparison on camera novel view synthesis.**

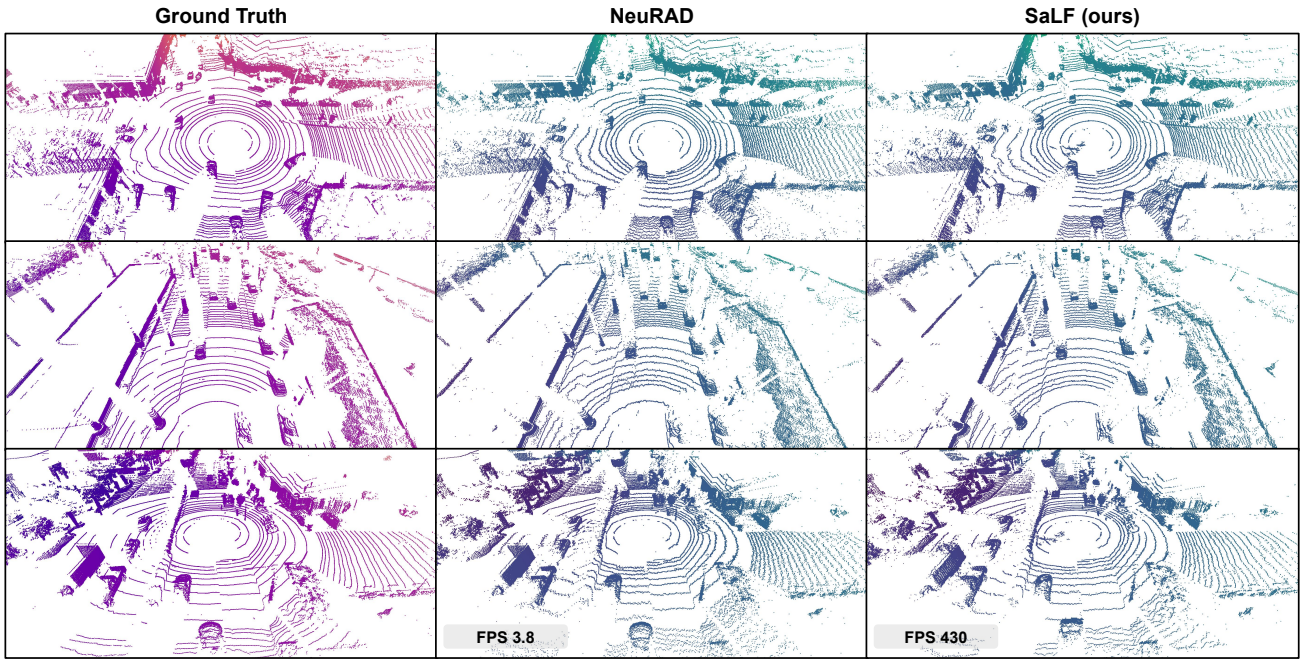

Figure 12. **Additional qualitative comparison on LiDAR novel view synthesis.**

self-driving multi-modal sensor simulators (*e.g.*, UniSim, NeuRAD) while improving efficiency, enabling more scalable simulation.

C.6. Additional Examples for Complex Sensor Modelling and Secondary Effects

We provide additional examples of ray-based rendering which 3DGS does not support directly in Fig. 15, 16 and 17. In Fig. 15, we show an example of simulating ray-tracing based effects including refraction/reflection (with variable index of refraction) and shadows. In Fig. 16, we simulate a fish-eye camera with different distortion parameters. In Fig. 17, we provide

Figure 13. **Controllable simulation: actor removal.**

Figure 14. **Controllable simulation: SDV manipulation.**

additional examples of simulating rolling-shutter effects for both camera and LiDAR.

C.7. Surface Normal Visualization

Figure 18 demonstrates surface rendering for SaLF where we show surface normal visualization from our SaLF representation. SaLF maintains smooth, continuous surface transitions through its effective SDF representation. More regularization can be applied to further improve surface consistency, particularly in regions with complex geometry or sparse training views.

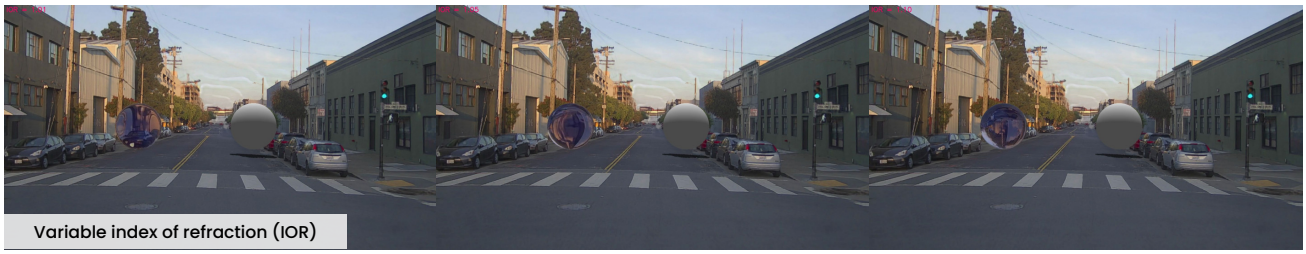

Figure 15. **Simulating ray-tracing based effects.**

Figure 16. **Simulating distorted fish-eye cameras.**

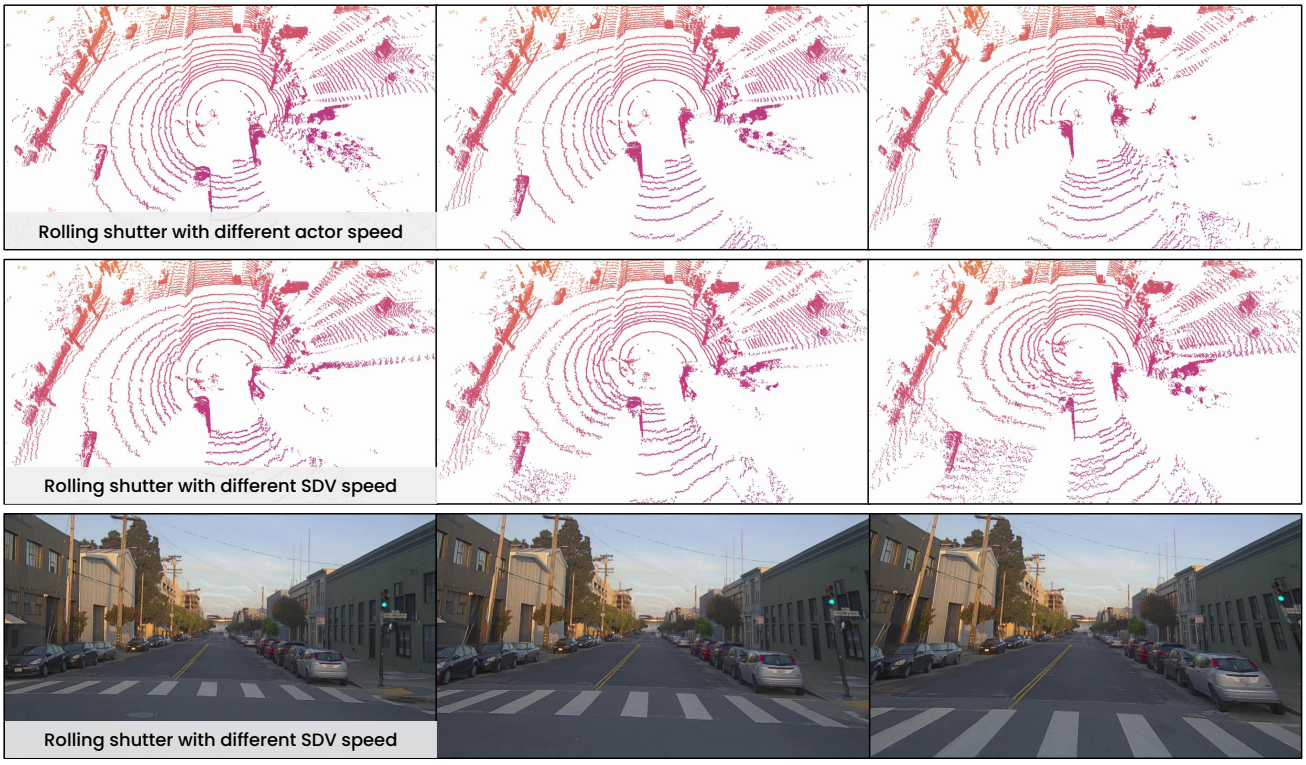

Figure 17. **Simulating rolling-shutter LiDAR and camera.**

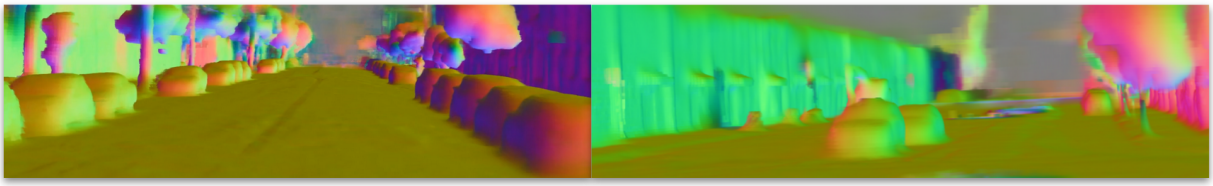

Figure 18. **Rendered Surface Normal.**

D. Discussions

D.1. Sorting Ambiguities in Rasterization

In voxel-based rasterization, a potential issue arises when determining front-to-back ordering. When sorting voxels by their center distances to the camera, a voxel whose center is farther away might have its front edge or corner positioned closer to the camera than another voxel with a nearer center. This sorting ambiguity could theoretically lead to incorrect alpha compositing and visual artifacts, particularly concerning for physically-based sensor simulation. Contrary to expectation, our analysis shows this issue has minimal practical impact for two key reasons. First, the volume rendering equation inherently mitigates the problem: when a ray intersects an edge or corner of a voxel, the small traversal distance (δ) results in a negligible opacity contribution ($\alpha = 1 - \exp(-\sigma\delta)$), regardless of density values. This effectively downweights contributions from potentially ambiguous intersections. Second, SaLF's adaptive densification process tends to produce nearby voxels of similar sizes, further reducing sorting ambiguities. During coarse-to-fine optimization, voxels exhibiting similar color field gradients in proximity undergo comparable levels of subdivision. With similarly-sized neighboring voxels, the maximum possible distance between a voxel's center and its boundary becomes more constrained. This significantly reduces cases where a farther voxel's corner could extend substantially in front of a nearer voxel's boundary, making center-based sorting a better approximation of the correct front-to-back order. Our quantitative comparison between images by rasterization (using center-based sorting) and ray-casting (with correct traversal ordering) confirms this effect, showing an average L_1 difference of only 0.013 — considerably smaller than the minimum distinguishable difference in 8-bit color ($1/256 \approx 0.0039$). This demonstrates that potential artifacts from sorting ambiguities remain mostly imperceptible and below numerical precision thresholds. It is worth noting that this sorting ambiguity is not unique to our voxel-based approach, but is a common issue in all splatting-based methods with spatially-extended primitives [26, 33, 40].

D.2. Artifacts and potential enhancements

In Fig. 19, we show some artifacts of our method. For dynamic actors such as vehicles and pedestrians, our method produces lower visual quality compared to baselines, likely due to their explicit actor pose optimization during training, which can be added to SaLF as well. Our method also struggles with very strong view-dependent effects, which may need higher resolution voxels and more spherical harmonics to capture. Additionally, we observe occasional transparency artifacts in our renderings, which might be addressed by implementing more robust near-surface rendering techniques. Finally, our current implementation has difficulty accurately representing fine geometric details, such as individual tree leaves, due to the inherent resolution limitations of our voxel-based representation and noisy data.

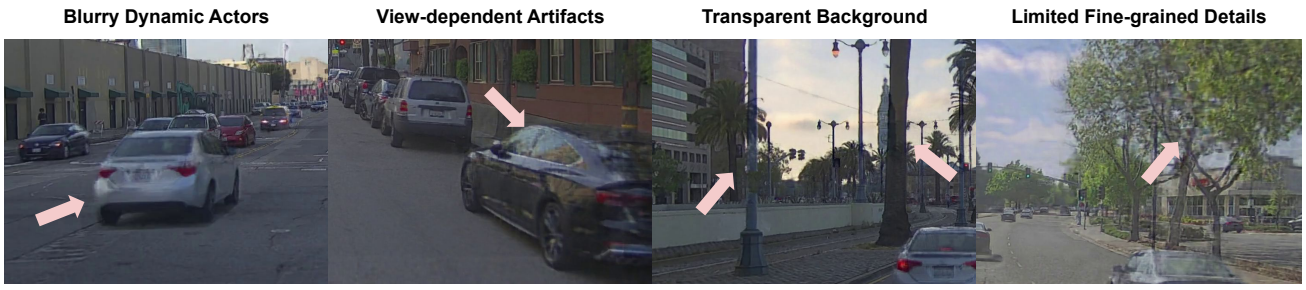

Figure 19. **Artifacts.**

D.3. Limitations

As discussed in Appendix A, alimitation of SaLF is the requirement for a significantly higher number of voxels compared to 3DGS-based Street Gaussian to achieve comparable rendering quality, which impacts the overall rendering performance. Additionally, our choice of using axis-aligned, non-overlapping voxels, while efficient for sensor simulation, presents challenges for generalizable scene reconstruction from images. Unlike methods that predicts a set of 3D Gaussians, which can flexibly adapt their orientation and overlap to match scene geometry, our sparse set of non-overlapping and axis-aligned voxels with fixed resolutions enforces strict spatial constraints that may be difficut to adhere to during generation. These limitations highlight potential directions for future improvements in our approach.

D.4. Future Work

While we demonstrate SaLF's effectiveness in autonomous driving sensor simulation, our representation has potential for other applications such as virtual reality. Multi-scale voxel representation in SaLF aligns perfectly with level-of-detail rendering techniques, similar to MipMap structures, enabling efficient rendering across varying distances and viewpoints. Moreover, the local nature of our representation enables efficient streaming and management of large-scale scenes, similar to open-world video games, where content can be dynamically loaded based on camera position and scene importance. These extensions would maintain SaLF's core advantages of efficiency and flexibility while broadening its impact.

D.5. Broader Impact and Data Privacy

We evaluate SaLF on the PandaSet dataset, which contains 2D/3D sensor data from real-world driving scenes. This dataset has been properly vetted for ethical considerations in data collection and usage. While the dataset includes street scenes with pedestrians, it contains no personally identifiable information or sensitive content.

SaLF advances autonomous vehicle development by enabling efficient, realistic, and real-time sensor simulation. Its ability to accurately simulate various sensor effects (e.g., rolling shutter, refraction) and support interactive scene manipulation can significantly improve the testing and validation of autonomous systems.