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

Yun Chen[1,2*] Matthew Haines[1,3*†] Jingkang Wang[1,2] Sahil Jain[1]
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, sjain, klis, siva, 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 for unified multi-sensor simulation. 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 has similar realism as existing self-driving sensor simulation methods while improving efficiency and enhancing capabilities, enabling more scalable simulation.**

I. Introduction

Closed-loop simulation is an integral part of testing self-driving vehicles [1], [2]. To evaluate the full autonomy system, modern simulators must simulate the sensors (e.g., LiDAR, camera) that the vehicle utilizes for perception. Such multi-modal systems must be realistic to accurately measure autonomy performance, and highly efficient to enable scalable testing and training.

Neural Radiance Field (NeRF) representations [3] have made significant progress in realistic 3D multi-sensor self-driving simulation [4], [5]. These methods model the scene as composable dynamic actors and static backgrounds using 3D implicit representations. By casting rays according to sensor intrinsics and extrinsics, NeRF-based methods generate high-fidelity sensor data via volume rendering. However, high computational demands during training (hours per scene) and rendering (1-2 FPS) limit their scalability for real-time simulation, especially given modern vehicles utilize over 20 sensors [6]. While several methods improve rendering efficiency by baking into meshes [7], [8], [9], [10] or grid look-ups [11], [12], they still suffer from time-consuming training, complex baking procedures, and potential quality degradation compared to the original representation.

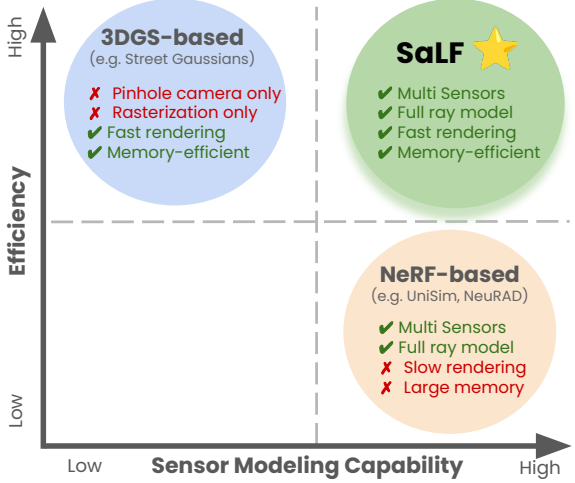

Fig. 1. **SaLF combines high efficiency with advanced sensor modeling capabilities for self-driving simulations.**

3D Gaussian Splatting (3DGS) [13], [14], [15] enables fast training and real-time pinhole camera rendering by modeling scenes as explicit 3D Gaussian particles and rasterizing them onto the image plane. However, like other rasterization-based approaches, 3DGS lacks native support for "ray-based" rendering, which is required for complex sensor models such as rolling-shutter LiDARs or fish-eye cameras. It also struggles to accurately model phenomena like motion blur and secondary lighting effects (e.g., refraction), which are crucial for robust testing [16], [17]. These limitations restrict its use in comprehensive testing despite its real-time capabilities. Furthermore, similar to how graphics meshes support both raytracing and rasterization, we argue for a unified, learnable volumetric representation. While specialized LiDAR or camera models may achieve high performance in their respective domains, maintaining disjoint representations for the same scene adds prohibitive memory overhead and pipeline complexity. Furthermore, a shared volumetric foundation is essential to guarantee physical consistency across simulated sensors, ensuring that downstream autonomy systems do not fail due to misalignment.

In this work, we present Sparse Local Fields (SaLF), a volumetric representation supporting both rasterization and raytracing for multi-sensor simulation. SaLF consists of voxel primitives, each acting as a local implicit field mapping

*Equal contributions.
†Work done while a research intern at Waabi.

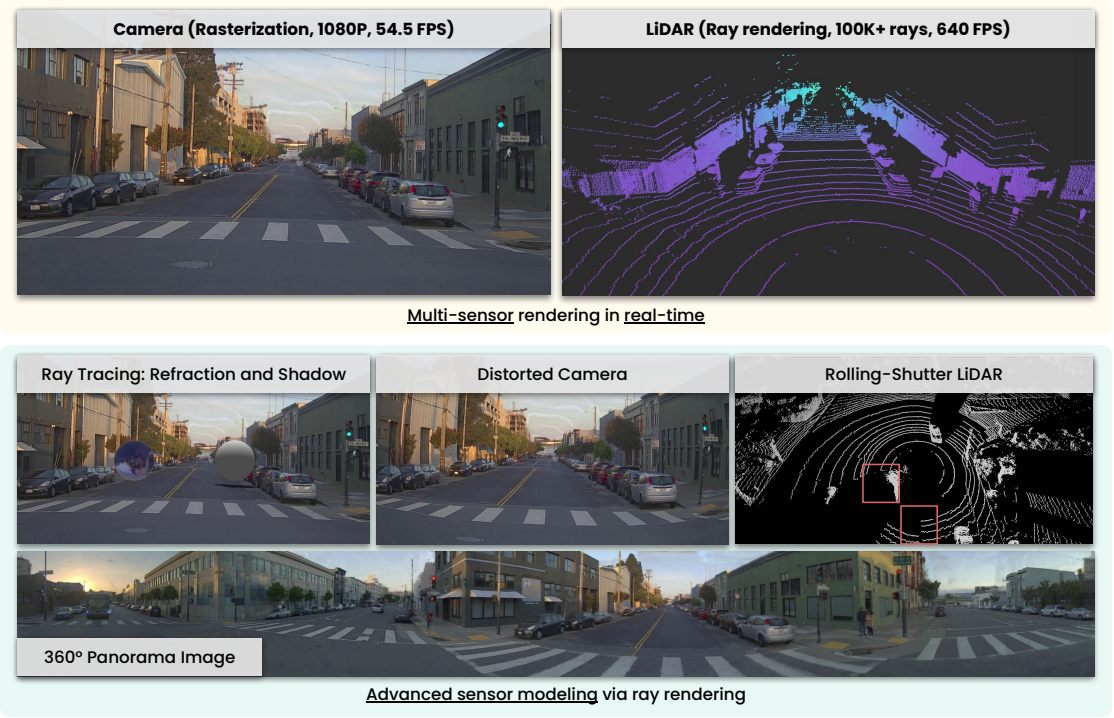

Fig. 2. **Real-time self-driving sensor simulation with SaLF representation.** Our method achieves high-performance rendering for both camera and LiDAR, and supports advanced features including secondary effects (e.g., refraction, reflection, and shadow) and complex sensor models (e.g., fisheye, rolling-shutter,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

spatial coordinates to geometry and appearance. Like NeRF, SaLF supports volume rendering via ray-casting to accurately model LiDAR and complex cameras. Without any baking, SaLF's voxels are natively backed by an octree, directly supporting accelerated raytracing. Like 3DGS, SaLF can be efficiently rasterized for fast pinhole camera rendering. SaLF also supports adaptive voxel pruning and densification, creating compact representations for large scenes.

By unifying support for diverse sensor models and complex phenomena such as rolling-shutter effects and refraction, SaLF is highly valuable for scalable autonomous driving simulation. Experiments on a public self-driving dataset demonstrate that SaLF achieves comparable realism to prior works while being significantly more efficient to train and render. We showcase its raytracing capabilities for fast and versatile sensor simulation, and demonstrate that SaLF achieves a smaller domain gap for downstream perception, prediction, and planning tasks.

II. Related Work

a) Efficient NeRFs: Neural Radiance Fields (NeRF) [3] provide a foundation for photorealistic 3D reconstruction, yet the vanilla formulation remains computationally intensive. Recent works address these efficiency challenges. DVGO [18] replaces the global MLP with a sparse 3D grid for faster convergence, while Plenoxels [19] utilizes explicit spherical harmonics in a sparse grid, removing neural networks entirely. Instant-NGP [20] further adopts multi-resolution hash encoding for compactness. To enable real-time rendering, "baking" methods pre-compute neural field properties into efficient representations like sparse voxel grids [9], triplanes [11], octrees [12], or VDB [21]. Other works extract explicit meshes [7], [8], [10] for standard rasterization pipelines. However, these approaches often face intractable memory usage in large scenes, quality degradation when baked [10], [21], [12], or long training times [9], [7], [8]. SVR [22] performs voxel rasterization but focuses strictly on cameras without supporting ray-based sensor models. In contrast, SaLF optimizes and renders efficiently for multi-sensor simulation in large scenes without baking.

b) 3DGS: 3D Gaussian Splatting (3DGS) [13] enables real-time photorealistic rendering by representing scenes with oriented 3D Gaussians and tile-based rasterization. 3DGS has been applied to 3D generation [23] and camera simulation [15], [14], [24], [25]. However, like other rasterization-based approaches, 3DGS assumes a pinhole camera model and lacks the flexible ray-based rendering of NeRF, limiting its use for rolling-shutter LiDARs and fisheye cameras. Several concurrent works [26], [27], [28], [29], [30], [31] have extended 3DGS to non-pinhole sensors by approximating sensor-specific effects, but these solutions often hinder generalization. Concurrent works enable 3DGS ray tracing by wrapping Gaussians in proxy geometries for BVH traversal [32], [33]. This introduces significant structural overhead compared to SaLF, which natively supports both paradigms within a unified representation.

c) Data-driven Sensor Simulation for Self-Driving: Traditional graphics-based simulators [34], [35] face significant domain gaps in geometry and appearance. Data-driven neural rendering [36], [4] has attracted attention for its photo-

realism and ability to reconstruct real-world scenes. Methods like [36], [4], [5], [37] use NeRFs to build digital twins, decomposing scenes into backgrounds and actors to enable realistic camera and LiDAR [4], [38], [5] simulation. However, these typically require hours of training per 10-second clip and cannot perform real-time rendering. To address this, recent works [14], [15], [39] leverage compositional 3DGS for real-time camera simulation but remain restricted to pinhole models. In contrast, we build the first self-driving neural sensor simulator supporting real-time rendering for complex cameras and LiDAR.

III. 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. III-A) and our rasterization and ray-casting rendering algorithms (Sec. III-B), the coarse-to-fine densification strategy for compactness and efficient training (Sec. III-C), and discuss how SaLF compares to NeRF and 3DGS (Sec. III-D).

A. 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 $\mathscr{V} \subset \mathbb{R}^3$ be a three-dimensional volume in an axis-aligned bounding box (AABB) with dimensions (V_h, V_w, V_d). We partition $\mathscr{V}$ into a regular grid $\mathscr{G}$ 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 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.

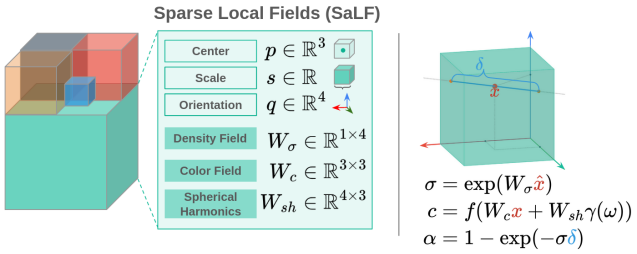

Fig. 3. **SaLF Representation. Left:** SaLF models scenes using an adaptive sparse voxel grid with variable scales. Each voxel is characterized by static parameters and learnable parameters. **Right:** Within a voxel, for any point with normalized coordinates $\mathbf{x}$, the density σ and color c values 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.

a) 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 Eqn. (1) and Eqn. (2), respectively.

b) 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 [4], [5] 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 [40], [41]:

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

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.

B. 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 Eqn. (3), with rasterization being faster but making more

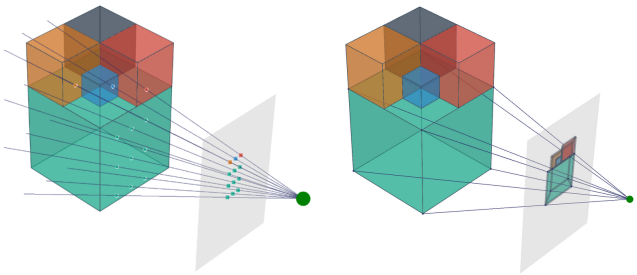

Fig. 4. **SaLF can be efficiently rendered by ray casting (left) and rasterization (right).** Ray casting is more flexible and can handle more complex physics phenomena, while rasterization is more efficient for pinhole cameras. Both follow the same rendering equation.

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.

a) 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 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.

b) 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 Eqn. (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.

C. 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 (evaluated as the L_1 norm of the loss gradient with respect to the color

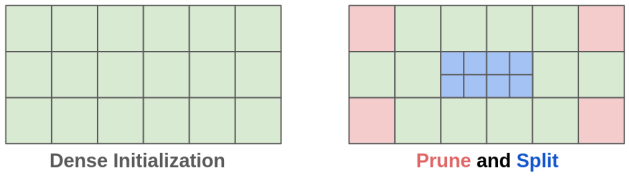

Fig. 5. **Initialization and Densification.** SaLF initializes the scene representation with a coarse regular grid partitioning, then adaptively prune empty region while densifying regions that need fine-details.

field parameters W_c) are subdivided into eight child voxels in an octree-aligned manner. Voxels with negligible opacity ($\alpha < 0.001$) are removed from the sparse set. Densification and pruning together enables preserving of fine details while maintaining a compact memory footprint.

D. Comparison with 3DGS and NeRF

SaLF represents implicit scenes using discrete volumes like other voxel-based NeRF variants such as DVGO [18] and Plenoxels [19]. Unlike these methods, whose uniform grids cause an $O(N^3)$ memory explosion, our approach ensures memory efficiency at scale by allocating finer voxels exclusively to complex geometry via gradient-based octree subdivision.

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 [42], [43], 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.

IV. 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.

A. Compositional Scene Representation

a) Dynamic Scene Modelling: Following previous work [36], 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.

b) 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.

B. Learning

The scene representation is optimized through the following loss function:

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

where $\mathscr{L}_{\text{color}}$ and $\mathscr{L}_{\text{depth}}$ measure the ℓ_1 distance between rendered and ground-truth images and LiDAR depth, respectively. $\mathscr{L}_{reg}$ enforces spatial consistency by summing the penalized differences in local SDF s, opacity α, and surface normals $\mathbf{n}$ across all adjacent voxel pairs $(i,j) \in \mathscr{N}$:

$$\mathscr{L}_{reg} = \sum_{(i,j)\in\mathscr{N}} \left(\lambda_s |s_i - s_j| + \lambda_\alpha |\alpha_i - \alpha_j| + \lambda_n (1 - \mathbf{n}_i \cdot \mathbf{n}_j)\right).$$

V. Experiments

A. Experimental Setup

a) Dataset and Evaluation Protocol: We evaluate our method on the PandaSet dataset [44], which consists of 103 driving scenes captured at 1920×1080 resolution, with 80 frames per scene and 360-degree LiDAR data. Following previous works [4], [5], we use the standard 10-log evaluation set, splitting the 80 frames per log into even (training) and odd (evaluation) frames. This 50%/50% split is more challenging than the typical 90%/10% setting in other methods [15]. We report photorealism on the front camera via PSNR, SSIM and LPIPS (VGG-backbone) [45] 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. For clarity during evaluation, we explicitly divide our unified representation into two inference modes: **SaLF-Raster** for real-time pinhole rendering, and **SaLF-Ray** (ray casting) for complex sensors (e.g., fisheye, LiDAR).

b) Implementation Details of SaLF: We implement efficient rasterization and ray-casting operations using `Taichi` [46], [47]. For training, we employ the Adam optimizer 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. A high-capacity "SaLF (large)" is also trained (5M voxels, 4500 iterations) for better performance.

c) Comparison with state-of-the-art (SoTA) methods: We compare our approach against several SoTA methods in self-driving sensor simulation including UniSim [4], NeuRAD [5] and Street Gaussian [15]. 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.

B. Fast and Realistic Multi-Sensor Simulation

a) Comparison to baseline methods: We compare SoTA sensor simulation approaches on PandaSet, as shown in Table I. Through **SaLF-Raster**, our method achieves real-time rendering for both camera and LiDAR with comparable fidelity to SoTA simulators, and accelerates reconstruction speed by at least $5\times$ compared to NeRF based methods. Simultaneously, **SaLF-Ray** extends this shared volumetric representation to support complex, ray-based sensor models where physical accuracy is important. NeuRAD achieves slightly higher camera realism due to a post-processing CNN. However, without this CNN, SaLF achieves similar realism while being significantly faster, demonstrating the efficiency of our sparse voxel representation (Fig. 6). 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. Furthermore, our method's multi-scale initialization of coarse voxels reconstructs distant regions more accurately than 3DGS-based methods like Street Gaussian (Fig. 9), which struggle due to their reliance on sparse Structure-from-Motion and LiDAR initialization.

b) Extrapolation: To further evaluate robustness for novel view synthesis, we assess extrapolation performance by displacing the camera $\pm$2m along the XY-axis and computing the Fréchet Inception Distance (FID) between the rendered and source images. As reported in Table V, SaLF and StreetGS achieve comparable extrapolation quality. While NeuRAD demonstrates greater robustness, this is primarily

[1]NeuRAD [5] reports LPIPS(AlexNet) and median L_2 LiDAR error.

TABLE I

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. WE HIGHLIGHT FIRST, SECOND, THIRD.

Models	Rendering FPS ↑		Recon Time ↓	Rendering Realism			
	Camera (SaLF-Raster)	LiDAR (SaLF-Ray)	RTX-3090 hour	PSNR↑	SSIM↑	LPIPS↓	LiDAR-L1↓
Street Gaussian [14]	**115.5**	✗	2.26	25.65	**0.777**	0.307	✗
UniSim [4]	1.3	11.8	1.67	25.63	0.745	**0.288**	0.100
NeuRAD [5] (2x)	1.7	3.79	3.48	**26.60**	0.770	0.297	**0.085**
SaLF (base)	54.5	**640**	0.31	25.48	0.744	0.373	0.142
SaLF (large)	34.3	430	0.48	25.78	0.762	0.344	0.111

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

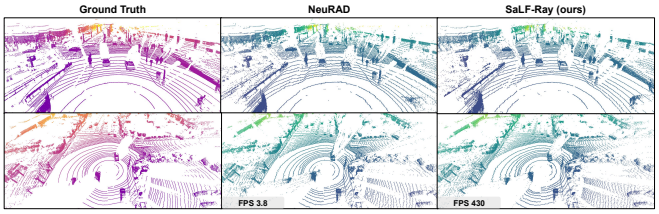

Fig. 7. **Qualitative comparison on LiDAR novel view synthesis.** Our method achieves comparable LiDAR rendering performance compared to NeuRAD, while being *100 × faster* in rendering.

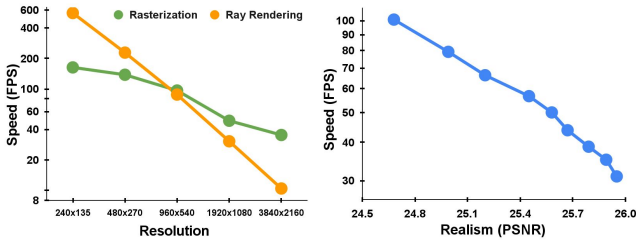

Fig. 8. **Efficiency.** Left: rendering speed of rasterization (SaLF-Raster) *vs.* ray-tracing (SaLF-Ray) across resolutions. Right: trade-off between rendering speed and realism.

attributable to its CNN post-processing; notably, NeuRAD without its CNN decoder performs similarly to our method.

c) Efficiency analysis: Our analysis (Fig. 8) reveals that at lower resolutions ($< 960 \times 540$), ray rendering significantly outperforms rasterization due to the latter's resolution-independent projection overhead. 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. More subdivision iterations increase voxel density and detail at the expense of inference speed.

d) 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. Table II reports the camera realism, demonstrating the value of both choices.

C. Simulation Capabilities and Extensions

SaLF establishes a foundation for a performant, data-driven multi-sensor renderer that can be readily extended to various downstream simulation needs. As showcased in Fig. 2, our method supports ray-based light phenomena (e.g., refraction, shadows of inserted objects) and generalized camera models like panoramic rendering, demonstrating its versatility for diverse sensor rendering tasks. Furthermore, because temporal motion during capture significantly impacts perception [16], SaLF explicitly simulates both LiDAR and camera rolling-shutter effects (Fig. 10). Building on these ray-based capabilities, we also incorporate LiDAR-specific features such as raydrop and intensity simulation (Table III)

Fig. 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.

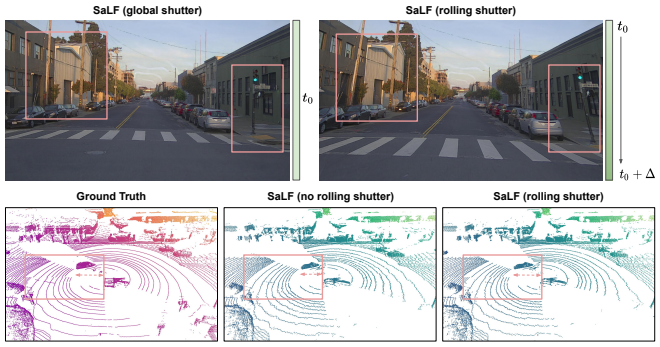

Fig. 10. **Rolling-shutter simulation via SaLF-Ray. Top:** We render the same view using global shutter and rolling-shutter camera models (see highlighted distorted region). **Bottom:** SaLF simulates rolling-shutter effect and accurately match ground truth point clouds (see lidar sweep seam and relative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

Beyond these, future directions include incorporating beam divergence, actor-label refinement, and sensor pose refinement, offering exciting avenues for the community.

D. Downstream Applications

In addition to photometric and geometric realism, the fidelity of a sensor simulator must be evaluated by how accurately it replicates the sensor data as perceived by the downstream autonomy system. We measure the open-loop domain gap on a 187-sequence highway dataset using a modern autonomy stack with a joint detection/prediction transformer [48] and a trajectory planner [49]. For each sequence, we run the autonomy on paired real and simulated data, and compare the downstream outputs using three metrics: (1) *detection agreement*: average precision (AP) matching simulated to real detections (IoU ≥ 0.5); (2) *prediction ADE*: minimum average displacement error between forecasted and ground-truth trajectories; and (3) *planning consistency*: endpoint deviation between 5s plans generated from real versus simulated inputs. As shown in Table IV, SaLF achieves lower domain gap across detection, prediction, and planning, outperforming UniSim on all downstream tasks.

TABLE II

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

TABLE III

SALF EXTENSION FOR LIDAR RAYDROP AND INTENSITY.

Models	Intensity	Raydrop Acc.	Speed (FPS)
NeuRAD [5]	**0.062**	**96.2**	3.79
UniSim [4]	0.085	91.0	11.8
SaLF-large	0.069	92.7	**350**

TABLE IV

DOWNSTREAM DOMAIN GAP EVALUATION.

	Det. Agg. ↑	Pred. ADE ↓	Plan Cons. ↓
UniSim [4]	0.74	0.63	0.99
Ours	**0.78**	**0.52**	**0.83**

TABLE V

EXTRAPOLATION PERFORMANCE. FID SCORES FOR NOVEL VIEW SYNTHESIS WHEN DISPLACING THE CAMERA $\pm$2M ALONG THE XY-AXIS.

Models	FID ↓
NeuRAD w/ CNN	**29.52**
NeuRAD w/o CNN	40.84
Street Gaussian	37.92
SaLF-large	41.22

E. 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, rather than adaptive primitives (like 3DGS) that dynamically adjust their shape to local structures. We also note that additional modifications are required to support non-rigid and temporal changes in our scene representation [39]. While our method supports full raytracing, and we demonstrate phenomena such as shadow in Fig. 2, we train SaLF using primary rays only.

VI. 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.

VII. Acknowledgements

We thank Ioan-Andrei Barsan and Yasasa Abeysirigoonawardena for constructive discussion and feedback. We thank the Waabi team for their valuable assistance and support.

References

[1] J. Wang, A. Pun, J. Tu, S. Manivasagam, A. Sadat, S. Casas, M. Ren, and R. Urtasun, "Advsim: Generating safety-critical scenarios for self-driving vehicles," in *CVPR*, 2021.

[2] J. Sarva, J. Wang, J. Tu, Y. Xiong, S. Manivasagam, and R. Urtasun, "Adv3d: Generating safety-critical 3d objects through closed-loop simulation," in *7th Annual Conference on Robot Learning*, 2023.

[3] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *ECCV*, 2020.

[4] Z. Yang, Y. Chen, J. Wang, S. Manivasagam, W.-C. Ma, A. J. Yang, and R. Urtasun, "Unisim: A neural closed-loop sensor simulator," in *CVPR*, 2023.

[5] A. Tonderski, C. Lindström, G. Hess, W. Ljungbergh, L. Svensson, and C. Petersson, "NeuRAD: Neural rendering for autonomous driving," in *CVPR*, 2024.

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

[7] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi, "MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures," *arXiv*, 2022.

[8] L. Yariv, P. Hedman, C. Reiser, D. Verbin, P. P. Srinivasan, R. Szeliski, J. T. Barron, and B. Mildenhall, "BakedSDF: Meshing neural SDFs for real-time view synthesis," *arXiv*, 2023.

[9] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," in *ICCV*, 2021.

[10] J. Y. Liu, Y. Chen, Z. Yang, J. Wang, S. Manivasagam, and R. Urtasun, "Real-time neural rasterization for large scenes," in *ICCV*, 2023.

[11] C. Reiser, R. Szeliski, D. Verbin, P. P. Srinivasan, B. Mildenhall, A. Geiger, J. T. Barron, and P. Hedman, "MERF: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes," *arXiv*, 2023.

[12] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "Plenoctrees for real-time rendering of neural radiance fields," *ICCV*, 2021.

[13] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D gaussian splatting for real-time radiance field rendering," *TOG*, 2023.

[14] Y. Yan, H. Lin, C. Zhou, W. Wang, H. Sun, K. Zhan, X. Lang, X. Zhou, and S. Peng, "Street gaussians for modeling dynamic urban scenes," *arXiv*, 2024.

[15] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.-H. Yang, "DrivingGaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes," in *CVPR*, 2024.

[16] S. Manivasagam, I. A. Bârsan, J. Wang, Z. Yang, and R. Urtasun, "Towards zero domain gap: A comprehensive study of realistic LiDAR simulation for autonomy testing," in *ICCV*, 2023.

[17] A. Pun, G. Sun, J. Wang, Y. Chen, Z. Yang, S. Manivasagam, W.-C. Ma, and R. Urtasun, "Neural lighting simulation for urban scenes," in *NeurIPS*, 2023.

[18] C. Sun, M. Sun, and H.-T. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," *CVPR*, 2022.

[19] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," *CVPR*, 2022.

[20] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," in *SIGGRAPH*, 2022.

[21] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "FastNeRF: High-fidelity neural rendering at 200fps," *ICCV*, 2021.

[22] C. Sun, J. Choe, C. Loop, W.-C. Ma, and Y.-C. F. Wang, "Sparse voxels rasterization: Real-time high-fidelity radiance field rendering," in *CVPR*, 2025.

[23] T. Yi, J. Fang, J. Wang, G. Wu, L. Xie, X. Zhang, W. Liu, Q. Tian, and X. Wang, "GaussianDreamer: Fast generation from text to 3D gaussians by bridging 2D and 3D diffusion models," in *CVPR*, 2024.

[24] Y. Chen, J. Wang, Z. Yang, S. Manivasagam, and R. Urtasun, "G3R: Gradient guided generalizable reconstruction," in *ECCV*, 2024.

[25] J. Wang, H. Che, Y. Chen, Z. Yang, L. Goli, S. Manivasagam, and R. Urtasun, "Flux4d: Flow-based unsupervised 4d reconstruction," in *NeurIPS*, 2025.

[26] Q. Wu, J. M. Esturo, A. Mirzaei, N. Moenne-Loccoz, and Z. Gojcic, "3dgut: Enabling distorted cameras and secondary rays in gaussian splatting," in *CVPR*, 2025.

[27] Q. Chen, S. Yang, S. Du, T. Tang, P. Chen, and Y. Huo, "LiDAR-GS: Real-time LiDAR re-simulation using gaussian splatting," *arXiv*, 2024.

[28] Y. Ren, G. Wu, R. Li, Z. Yang, Y. Liu, X. Chen, T. Cao, and B. Liu, "Unigaussian: Driving scene reconstruction from multiple camera models via unified gaussian representations," *arXiv preprint arXiv:2411.15355*, 2024.

[29] P.-C. Kung, X. Zhang, K. A. Skinner, and N. Jaipuria, "Lihi-gs: Lidar-supervised gaussian splatting for highway driving scene reconstruction," in *CVPR*, 2025.

[30] G. Hess, C. Lindström, M. Fatemi, C. Petersson, and L. Svensson, "Splatad: Real-time lidar and camera rendering with 3d gaussian splatting for autonomous driving," in *CVPR*, 2025.

[31] Z. Liao, S. Chen, R. Fu, Y. Wang, Z. Su, H. Luo, L. Ma, L. Xu, B. Dai, H. Li, *et al.*, "Fisheye-gs: Lightweight and extensible gaussian splatting module for fisheye cameras," *arXiv*, 2024.

[32] N. Moenne-Loccoz, A. Mirzaei, O. Perel, R. de Lutio, J. M. Esturo, G. State, S. Fidler, N. Sharp, and Z. Gojcic, "3D gaussian ray tracing: Fast tracing of particle scenes," in *SIGGRAPH Asia 2024*, 2024.

[33] C. Zhou, L. Fu, S. Peng, Y. Yan, Z. Zhang, Y. Chen, J. Xia, and X. Zhou, "Lidar-rt: Gaussian-based ray tracing for dynamic lidar re-simulation," *arXiv preprint arXiv:2412.15199*, 2024.

[34] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *Conference on robot learning*, 2017.

[35] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, 2018.

[36] J. Ost, F. Mannan, N. Thuerey, J. Knodt, and F. Heide, "Neural scene graphs for dynamic scenes," *CVPR*, 2021.

[37] Z. Yang, J. Wang, H. Zhang, S. Manivasagam, Y. Chen, and R. Urtasun, "Genassets: Generating in-the-wild 3d assets in latent space," 2025.

[38] H. Wu, X. Zuo, S. Leutenegger, O. Litany, K. Schindler, and S. Huang, "Dynamic lidar re-simulation using compositional neural fields," in *CVPR*, 2024.

[39] Z. Chen, J. Yang, J. Huang, R. de Lutio, J. M. Esturo, B. Ivanovic, O. Litany, Z. Gojcic, S. Fidler, M. Pavone, *et al.*, "OmniRe: Omni urban scene reconstruction," *arXiv*, 2024.

[40] L. Yariv, J. Gu, Y. Kasten, and Y. Lipman, "Volume rendering of neural implicit surfaces," *NeurIPS*, 2021.

[41] Y. Siddiqui, T. Monnier, F. Kokkinos, M. Kariya, Y. Kleiman, E. Garreau, O. Gafni, N. Neverova, A. Vedaldi, R. Shapovalov, *et al.*, "Meta 3d assetgen: Text-to-mesh generation with high-quality geometry, texture, and pbr materials," *arXiv*, 2024.

[42] K. Cheng, X. Long, K. Yang, Y. Yao, W. Yin, Y. Ma, W. Wang, and X. Chen, "Gaussianpro: 3D gaussian splatting with progressive propagation," *arXiv*, 2024.

[43] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, "2d gaussian splatting for geometrically accurate radiance fields," in *ACM SIGGRAPH 2024 Conference Papers*, 2024.

[44] P. Xiao, Z. Shao, S. Hao, Z. Zhang, X. Chai, J. Jiao, Z. Li, J. Wu, K. Sun, K. Jiang, *et al.*, "Pandaset: Advanced sensor suite dataset for autonomous driving," in *ITSC*, 2021.

[45] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," *CVPR*, 2018.

[46] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, "Taichi: a language for high-performance computation on spatially sparse data structures," *TOG*, 2019.

[47] K. Sun, "Taichi 3D Gaussian Splatting," 2023.

[48] S. Casas, B. Agro, J. Mao, T. Gilles, A. Cui, T. Li, and R. Urtasun, "Detra: A unified model for object detection and trajectory forecasting," in *ECCV*, 2024.

[49] A. Sadat, M. Ren, A. Pokrovsky, Y.-C. Lin, E. Yumer, and R. Urtasun, "Jointly learnable behavior and trajectory planning for self-driving vehicles," in *IROS*, 2019.