FGGS-LiDAR: Ultra-Fast, GPU-Accelerated Simulation from General 3DGS Models to LiDAR

Junzhe Wu[1*], Yufei Jia[2*], Yiyi Yan[3], Zhixing Chen[1], Tiao Tan[1], Zifan Wang[4], Guangyu Wang[1], BoKui Chen[1†], Guyue Zhou[4†]

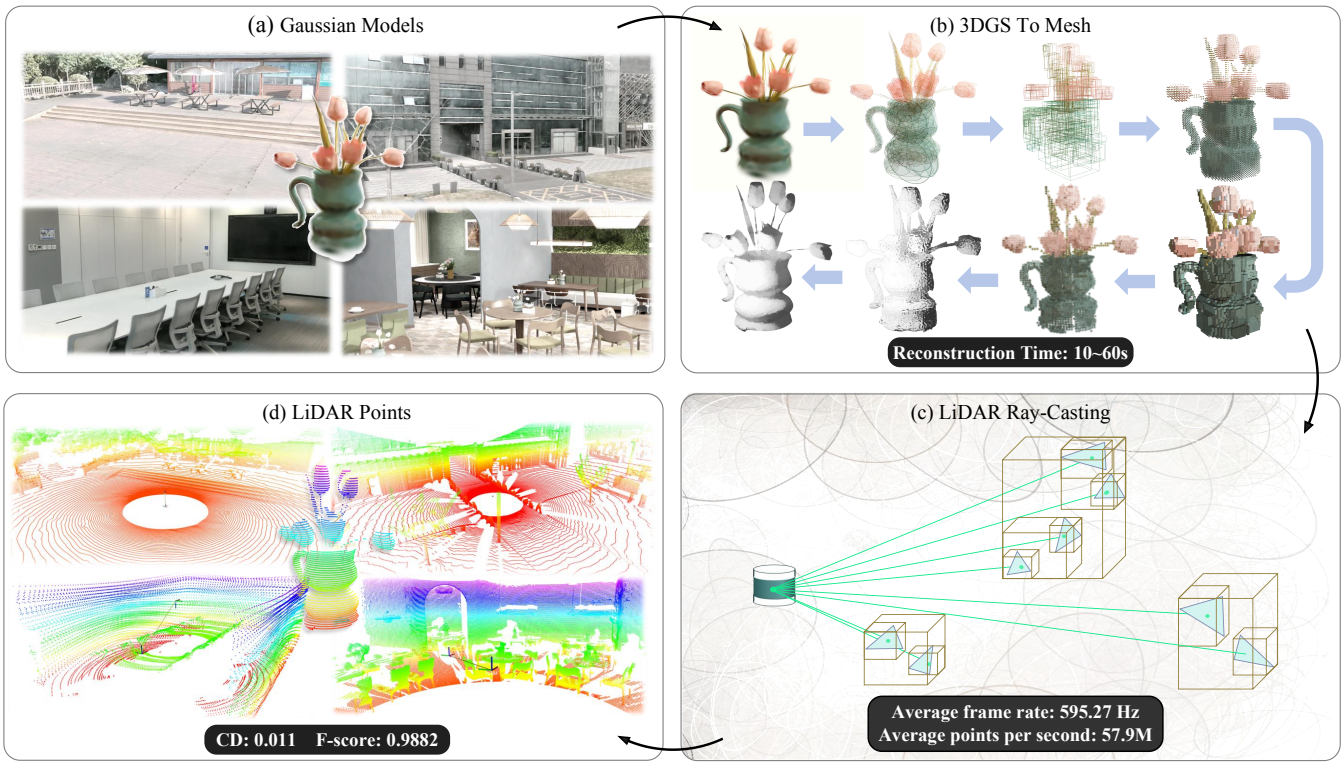

Fig. 1: **Overview of FGGS-LiDAR framework.** (a) Visualization of a generic 3DGS dataset. (b) Schematic illustration of our 3DGS2Mesh conversion pipeline, which transforms Gaussian primitives into mesh-based scene representations. (We use the vase as an example for the illustration.) (c) Conceptual diagram of our LiDAR ray-casting procedure, simulating range measurements through efficient GPU-based traversal. (d) Visualization of rendered LiDAR scans.

***Abstract*— While 3D Gaussian Splatting (3DGS) has revolutionized photorealistic rendering, its vast ecosystem of assets remains incompatible with high-performance LiDAR simulation, a critical tool for robotics and autonomous driving. We present FGGS-LiDAR, a framework that bridges this gap with a truly plug-and-play approach. Our method converts *any* pretrained 3DGS model into a high-fidelity, watertight mesh without requiring LiDAR-specific supervision or architectural alterations. This conversion is achieved through a general pipeline of volumetric discretization and Truncated Signed Distance Field (TSDF) extraction. We pair this with a highly optimized, GPU-accelerated ray-casting module that simulates LiDAR returns at over 500 FPS. We validate our approach on indoor and outdoor scenes, demonstrating exceptional geometric fidelity; By enabling the direct reuse of 3DGS assets for geometrically accurate depth sensing, our framework extends their utility beyond visualization and unlocks new capabilities for scalable, multi-modal simulation. Our open-source implementation is available at `https://github.com/TATP-233/FGGS-LiDAR`.**

*Equal contribution; †Corresponding Authors.
[1]Tsinghua University, Shenzhen, China. {wjz25, chenzx24, tt23, wanggy24}@mails.tsinghua.edu.cn, chenbk@tsinghua.edu.cn
[2]Department of Electronic Engineering, Tsinghua University, Beijing, China. jyf23@mails.tsinghua.edu.cn
[3]DISCOVER Robotics. albertyanyy@gmail.com
[4]Hong Kong University of Science and Technology (Guangzhou). wang_zifan@outlook.com
[5]The Institute for AI Industry Research (AIR), Tsinghua University, Beijing, China. zhouguyue@air.tsinghua.edu.cn

I. INTRODUCTION

LiDAR is a cornerstone modality for 3D perception, underpinning autonomous driving, localization, odometry, mapping, and indoor navigation [1], [2], [3], [4], [5]. To mitigate the prohibitive expense and logistical challenges of curating large-scale real-world datasets, simulation offers a controllable and reproducible source of data for training and benchmarking perception algorithms. Consequently, the

fidelity, efficiency, and scalability of LiDAR simulators are critical factors that directly determine their utility in both research and real-world deployment.

The landscape of LiDAR simulation has historically been defined by a trade-off between asset creation and rendering performance. Traditional simulation pipelines, built on explicit mesh representations, can produce geometrically accurate and controllable data but are fundamentally bottlenecked by the need for high-quality, often manually created, 3D assets [6], [7], [8], [9]. This dependency on specialized assets limits their scalability and adaptability to diverse, real-world environments. In response, recent advances in neural fields, particularly Neural Radiance Fields (NeRF), have enabled the reconstruction of scenes directly from sensor data [10]. However, while NeRF-based LiDAR simulators achieve impressive fidelity, their reliance on implicit representations and exhaustive volumetric ray marching renders them computationally intensive, with slow training times and low inference throughput that preclude real-time applications [11], [12], [13], [14], [15].

The emergence of 3D Gaussian Splatting (3DGS) promised to resolve this impasse, offering a transformative representation that combines photorealistic quality with real-time rendering speeds [16], [17], [18], [19]. Yet, the standard 3DGS formulation, optimized for visual fidelity, is ill-suited for geometric sensing tasks. Its rendering process often produces blurred surfaces and incoherent depth estimates, failing to model the precise first-return mechanics of a LiDAR sensor [20], [21], [22], [23], [24]. Recognizing this, a new class of specialized methods has sought to adapt the Gaussian framework specifically for LiDAR simulation [25], [26]. These approaches achieve high physical realism by integrating new, learnable attributes for intensity and ray-drop directly into the Gaussian primitives. However, this specialization comes at a cost: they require direct supervision from ground-truth LiDAR data and modify the core 3DGS architecture. This dependency on LiDAR-specific training data fundamentally limits their generality. As a result, the vast and rapidly expanding ecosystem of pre-existing, photometrically-trained 3DGS assets remains incompatible with LiDAR simulation, creating a significant ”generality gap” in the field.

To bridge this gap, we introduce FGGS-LiDAR, a framework that establishes a new paradigm for LiDAR simulation. Instead of specializing the scene representation, we propose a general-purpose pipeline that converts **arbitrary, off-the-shelf 3DGS assets** into a simulation-ready format. Our approach is founded on two core contributions. First, we develop a novel **3DGS-to-Mesh conversion process** that reliably produces watertight and topologically-consistent meshes from any pretrained 3DGS model, crucially without requiring dataset-specific supervision, architectural changes, or external priors like COLMAP [27]. Second, we contribute a **highly optimized, GPU-accelerated ray-casting module** that simulates LiDAR returns from these generated meshes at speeds exceeding 500 FPS, even on complex scenes with millions of triangles. By decoupling the 3DGS scene representation from the LiDAR simulation, our framework provides a truly ”plug-and-play” solution.

Our contributions can be summarized as follows: Our contributions can be summarized as follows:

- We introduce a general, prior-free ”3DGS2Mesh” pipeline that converts any pretrained 3DGS model into a high-quality, watertight mesh suitable for geometric simulation.
- We release a high-performance, open-source LiDAR ray-casting module that achieves ultra-fast simulation speeds (>500 FPS) through hardware-conscious optimizations.
- We demonstrate state-of-the-art geometric fidelity, achieving millimeter-level accuracy in reconstructing scene geometry from 3DGS assets, validated through direct comparison with ground-truth scans.

II. Related works

A. 3DGS for novel view synthesis

3DGS[16] introduced a differentiable splatting algorithm to render Gaussian primitives, establishing a practical and effective framework for novel view synthesis. Subsequent studies[24], [28], [21] extended this paradigm with improvements in anti-aliasing[24] and view-consistent rendering [21]. For example, PGSR [20] flattens 3D Gaussians into planar forms combined with unbiased depth rendering, while incorporating both single and multiple view regularizations to enforce global geometric consistency, thereby improving surface fidelity without compromising efficiency. Other methods such as 2D Gaussian Splatting [22] project volumetric Gaussians into oriented planar disks, offering a more compact and geometrically precise formulation.

B. Mesh reconstruction from 3DGS

Mesh reconstruction from 3DGS has been investigated as a means to extend Gaussian representations toward geometry-oriented applications. Existing methods can be broadly divided into two categories. One line of work, exemplified by frameworks such as GS2Mesh [29] and MILo [30], reconstructs meshes from Gaussian primitives through volumetric sampling, signed distance field (SDF) extraction, and meshing algorithms. Although these pipelines can generate watertight surfaces, they typically rely on auxiliary reconstructions such as COLMAP [31] [32] or multiview stereo to provide camera poses and depth priors. This dependency couples them tightly to the image acquisition and 3DGS training process, preventing direct application to arbitrary pretrained 3DGS models.

C. Gaussian-based LiDAR simulation

Recent studies have also explored extending Gaussian representations for LiDAR simulation. LiDAR-RT [25] proposed a Gaussian-based ray tracing framework, introducing learnable LiDAR-specific attributes (e.g., reflection intensity and ray-drop probability) into Gaussian primitives. In parallel, GS-LiDAR [26] adopts a panoramic Gaussian projection

approach, representing the scene with oscillatory 2D Gaussian primitives and modeling explicit ray–Gaussian interactions for LiDAR novel view synthesis. It further incorporates spherical harmonic coefficients to encode LiDAR-specific properties, yielding higher efficiency and fidelity than NeRF-based approaches.

However, these efforts face several limitations. They remain anchored to autonomous-driving datasets and require supervision from raw LiDAR scans, narrowing transferability to other domains.They also alter the standard 3DGS formulation with task-specific extensions, undermining drop-in compatibility with large pretrained 3DGS models. In addition, LiDAR-RT's ray-traced pipeline is computationally heavy, whereas GS-LiDAR's rasterization lacks an explicit time-of-flight model. Collectively, these factors limit scalability and plug-and-play use , motivating a general, efficient, and fully 3DGS compatible solution.

III. PRELIMINARIES

A. 3D Gaussian Splatting

3D Gaussian Splatting represents a 3D scene as a collection of anisotropic Gaussian primitives [16]. Each primitive G_i is defined by a set of learnable attributes: a mean position $\mu_i \in \mathbb{R}^3$, a 3D covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$, an opacity $\sigma_i \in (0,1)$, and spherical harmonics (SH) coefficients to model view-dependent appearance. The influence of the i-th Gaussian on a spatial location $x \in \mathbb{R}^3$ is described by an unnormalized Gaussian function:

$$G_i(x) = \exp\left(-\frac{1}{2}(x-\mu_i)^\top \Sigma_i^{-1}(x-\mu_i)\right), \qquad (1)$$

where the covariance matrix Σ_i is factorized as $\Sigma_i = R_i S_i S_i^\top R_i^\top$, using a scaling matrix S_i and a rotation matrix R_i. Our work takes such a collection of primitives, derived from a pretrained 3DGS model, as the geometric foundation for our simulation pipeline.

B. Truncated Signed Distance Functions

A Truncated Signed Distance Function (TSDF) is a volumetric data structure used to represent an implicit surface, making it an effective bridge between discrete point or voxel data and a continuous mesh representation [33]. Given a surface $\partial\Omega$, the signed distance for any point x in space is defined as:

$$d(x) = \operatorname{sgn}(x) \min_{y\in\partial\Omega} \|x-y\|, \qquad (2)$$

where the sign is negative if x is inside the surface and positive if it is outside. In practice, computing this distance field across an entire volume is computationally expensive and often unnecessary. The TSDF improves efficiency by restricting the computation to a narrow band of width $\pm\tau$ around the surface:

$$d_\tau(x) = \max(-\tau, \min(\tau, d(x))). \qquad (3)$$

This truncation stabilizes the representation by ignoring voxels far from the object boundary. The zero-level set of the resulting TSDF volume, $\{x | d_\tau(x) = 0\}$, corresponds to the object's surface. This implicit surface can then be efficiently converted into an explicit, watertight polygonal mesh using an isosurface extraction algorithm such as Marching Cubes [34].

IV. METHODS

Our objective is to develop a pipeline that converts any pretrained 3DGS model $\mathcal{G}$ into a high-quality, watertight mesh $\mathcal{M}$ suitable for ultra-fast LiDAR simulation. Our approach is a three-stage process: (1) We first discretize the continuous Gaussian representation into a structured, binary occupancy volume, a process accelerated by a Bounding Volume Hierarchy (Sec. IV-A). (2) From this volumetric data, we reconstruct a smooth and topologically consistent implicit surface using a narrow-band TSDF (Sec. IV-B). (3) Finally, we extract the explicit mesh $\mathcal{M}$ and perform LiDAR simulation using a highly optimized, GPU-accelerated ray-casting engine (Sec. IV-C).

A. Gaussian to BVH

Overview. To enable geometric reasoning from Gaussian primitives, we discretize continuous representations into voxel grids. It partitions the 3D space $\Omega \subset \mathbb{R}^3$ into a regular lattice of cubic cells with spacing h, where the center of a voxel is v_{ijk}. It is then assigned a binary occupancy volume $V(i,j,k)$, yielding a structured volumetric approximation of the Gaussian scene. For each voxel center v_{ijk}, we estimate density $D(v_{ijk})$ with an opacity weighting function $f(\sigma_i)$ and then evaluate its occupancy with the surface mask $\mathrm{Surf}(i,j,k)$.

BVH Construction Given a pretrained 3DGS $\mathcal{G} = \{(\mu_i, q_i, s_i, \alpha_i)\}_{i=1}^N$, we build an BVH over per-Gaussian axis-aligned bounding box (AABB). Each Gaussian G_i (center μ_i, orientation $R_i \equiv R(q_i)$, scale s_i) is conservatively enclosed by an AABB $b_i = (b_i^{\min}, b_i^{\max})$ with half-extents

$$r_i = \kappa\,|R_i|\,s_i, \qquad b_i^{\min} = \mu_i - r_i, \quad b_i^{\max} = \mu_i + r_i, \quad (4)$$

where $|R_i|$ is taken elementwise (projecting the oriented axes to world axes) and $\kappa \geq 1$ is a padding factor.

We then sort primitives by Morton codes computed from their centers μ_i within a scene box $[o, o+L]$ using b bits per axis:

$$m_i = \mathrm{interleave}\Big(\lfloor 2^b \tfrac{\mu_{i,x}-o_x}{L_x}\rfloor,\ \lfloor 2^b \tfrac{\mu_{i,y}-o_y}{L_y}\rfloor,\ \lfloor 2^b \tfrac{\mu_{i,z}-o_z}{L_z}\rfloor\Big). \quad (5)$$

Morton code generation and radix sort run massively in parallel. Given sorted codes $\{m_i\}$, internal node ranges are determined by the longest common prefix (LCP) of adjacent codes,

$$\mathrm{LCP}(i,j) = \max\{\ell : \text{ the first } \ell \text{ bits of } m_i \text{ and } m_j \text{ match}\}, \quad (6)$$

these values are computed with bitwise operations, and prefix scans form node ranges without recursion. Parent bounds are reduced in parallel as the union of child bounds:

$$\mathcal{B}(n) = \mathcal{B}(n_{\text{left}}) \cup \mathcal{B}(n_{\text{right}}). \quad (7)$$

Fig. 2: **LiDAR visualization across scenes.** A 4×3 panel: each row corresponds to one scene. From left to right, the columns show (1) 3DGS visualization, (2) the voxelized mesh, and (3) the LiDAR point cloud rendered by FGGS-LiDAR.

Node AABBs are reduced bottom-up in parallel, and nodes are laid out in breadth-first or Morton order to maximize coalesced memory access; traversal is stackless, avoiding global stacks. Consequently, BVH construction is near-linear in the number of Gaussians and contributes only a small fraction of the overall voxelization cost, even with millions of primitives.

Grid query and occupancy evaluation. The next step is to convert AABBs in BVH to a binary occupancy volume V for mesh reconstruction. We partition the volumetric grid into tiles of size B^3, and for a tile with AABB $\mathcal{B}_{\text{tile}}$, we cull candidates by intersecting per-Gaussian AABBs b_i:

$$\mathcal{C}_{\text{tile}} = \{\, G_i \mid b_i \cap \mathcal{B}_{\text{tile}} \neq \emptyset \,\}. \tag{8}$$

Each voxel $v \in \mathcal{V}_{\text{tile}}$ accumulates density only over $\mathcal{C}_{\text{tile}}$:

$$D(v) = \sum_{G_i \in \mathcal{C}_{\text{tile}}} \exp\Big(-\tfrac{1}{2}(v-\mu_i)^\top \Sigma_i^{-1}(v-\mu_i)\Big)\, f(\sigma_i). \tag{9}$$

Finally, we create the binary occupancy volume from $D(v)$. A voxel is occupied if

$$V(i,j,k) = \begin{cases} 1, & D(v_{ijk}) > \theta, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

where θ is a user-defined density threshold that controls the trade-off between completeness and sparsity of the voxelized geometry.

We check whether a voxel is interior by looking at its 6-neighborhood:

$$\begin{aligned} \text{Int}(i,j,k) = {} & V_{i,j,k} \wedge V_{i-1,j,k} \wedge V_{i+1,j,k} \\ & \wedge V_{i,j-1,k} \wedge V_{i,j+1,k} \wedge V_{i,j,k-1} \wedge V_{i,j,k+1}, \end{aligned} \tag{11}$$

With it, we create a surface mask to apply over the occupancy volume $V_{i,j,k}$:

$$\text{Surf}(i,j,k) = V_{i,j,k} \wedge \neg\, \text{Int}(i,j,k). \tag{12}$$

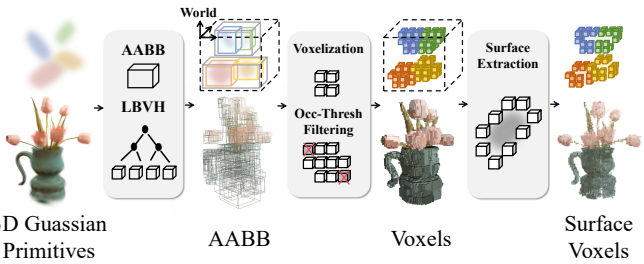

Fig. 3: **From Gaussian primitives to surface voxels.** We first organize 3D Gaussian primitives into an BVH structure for efficient spatial indexing. The scene is then voxelized within the global AABB, followed by occupancy thresholding and filtering to obtain a clean volumetric representation. Finally, surface voxels are extracted to yield contour points that approximate the underlying scene geometry.

B. Mesh Reconstruction

Overview. Given a binary occupancy volume $V : \Omega \to \{0,1\}$ with voxel spacing $\mathbf{s} = (s_x, s_y, s_z)$ and origin $\mathbf{o}$, we reconstruct a watertight surface by (i) denoising and re-thresholding the binary field, (ii) building a

narrow-band TSDF with reliable sign via outside flood-fill, (iii) extracting an isosurface with Marching Cubes, and (iv) performing structure-first simplification followed by non-shrinking smoothing.

Binary denoising and re-thresholding. To suppress salt-and-pepper artifacts while remaining resolution-agnostic, we convolve V with a Gaussian kernel of metric scale σ (expressed in meters and mapped to voxel units via $\mathbf{s}$):

$$V'(x) = (G_\sigma * V)(x). \tag{13}$$

A denoised occupancy $\tilde{V}$ is obtained by either a fixed threshold τ or a quantile threshold q:

$$\tilde{V}_{\text{fixed}}(x) = \begin{cases} 1, & V'(x) \geq \tau, \\ 0, & \text{otherwise}, \end{cases} \tag{14a}$$

$$\tilde{V}_{\text{quant}}(x) = \begin{cases} 1, & V'(x) \geq \text{Quantile}_q(V'), \\ 0, & \text{otherwise}. \end{cases} \tag{14b}$$

Narrow-band TSDF. Given a binary occupancy grid $V : \Omega \to \{0, 1\}$, we construct a signed distance field $\phi : \Omega \to [-r, r]$ in three logical stages. We first assign signs by identifying the outside region: a flood-fill is performed over free voxels, seeded from a padded frame Γ surrounding the domain, so that the 6-connected component $\mathcal{O}$ connected to Γ is labeled as outside. Voxels in $\mathcal{O}$ are assigned $s(x) = +1$, while all others (occupied cells or enclosed voids) are assigned $s(x) = -1$, ensuring stable sign labeling even in the presence of cavities and tunnels.

Next, unsigned distances are computed by layered propagation from the boundary set

$$\mathcal{S}_0 = \{\, x \mid \exists\, y \in \mathcal{N}_6(x),\ V(x) \neq V(y) \,\}, \tag{15}$$

where each expansion shell $\mathcal{S}_m$ grows over the 6-neighborhood and newly visited voxels record their first-arrival shell index $\kappa(x)$. The unsigned distance is then approximated as

$$\delta(x) = \kappa(x)\, v_{\min}, \qquad v_{\min} = \min(s_x, s_y, s_z), \tag{16}$$

which provides a conservative lower bound of the Euclidean distance and prevents diagonal leakage on anisotropic grids.

Finally, the signed distance field is obtained by combining the seeded sign and unsigned distance with truncation,

$$\phi(x) = \text{clip}\big(s(x)\, \delta(x), -r, r\big), \tag{17}$$

which restricts values to the radius-r band and avoids the memory and time overhead of a global Euclidean distance transform. All steps are executed fully on the GPU, where flood-fill, layered expansions, and truncation are carried out in parallel. By leveraging massive parallelism, the method achieves high efficiency and scales effectively to large volumes, avoiding the overhead of full-grid distance transforms.

Isosurface extraction. Next we apply isosurface extraction to get a smooth surface from the grid. The target surface is the level set

$$\mathcal{S} = \{\, x \in \Omega \mid \phi(x) = \text{iso} \,\}, \tag{18}$$

with $\text{iso} = 0$ by default (optional millimeter-scale bias $\pm$iso). We discretize $\mathcal{S}$ via Marching Cubes with step-size parameter `mc_step` and map vertices to world coordinates using $\mathbf{o}$ and $\mathbf{s}$. Per-vertex normals are estimated from $\nabla\phi$ for consistent outward orientation, yielding a watertight raw mesh $\mathcal{M}_{\text{raw}}$.

Mesh optimization. To make the reconstruction scalable for large scenes, we first apply structure-prioritized simplification to obtain $\mathcal{M}_{\text{simp}}$, targeting a prescribed face count or ratio while protecting boundaries and removing tiny components. We then perform non-shrinking smoothing (e.g., Taubin) on $\mathcal{M}_{\text{simp}}$ with parameters (λ, μ) and a small number of iterations, reducing staircase/normal noise while preserving sharp features and thin walls:

$$\mathcal{M}_{\text{final}} = \text{Smooth}_{\lambda,\mu,T}\big(\text{Simplify}(\mathcal{M}_{\text{raw}})\big).$$

Unless stated otherwise, all scale parameters (σ, r, iso) are specified in meters through $\mathbf{s}$, decoupling control from voxel resolution.

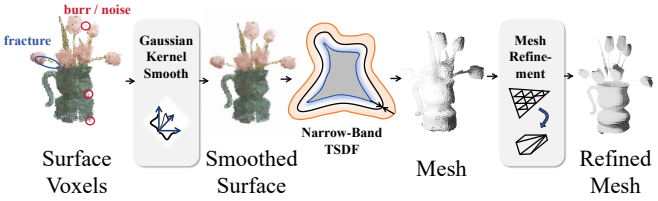

Fig. 4: **From surface voxels to refined mesh reconstruction.** The pipeline starts from surface voxels, which are denoised and smoothed through filtering and thresholding. We then construct a TSDF to extract an isosurface using the Marching Cubes algorithm for generating an initial mesh. Finally, mesh simplification and smoothing are applied to obtain the refined mesh representation.

C. Ray-casting for LiDAR Simulation

Ray to mesh measurement. In a LiDAR scan, the j-th beam is modeled as a ray

$$r_j(t) = x_s + t\, d_j, \quad t \geq 0, \tag{19}$$

where $T_s = \begin{bmatrix} R_s & t_s \\ \mathbf{0}^\top & 1 \end{bmatrix} \in SE(3)$ is the sensor pose in world coordinates, $x_s := t_s$ is the beam origin, and $d_j \in \mathbb{S}^2$ is a unit direction determined by the scanning pattern. The environment is represented as a triangle mesh

$$\mathcal{M} = \{\triangle_k = (v_{k,0}, v_{k,1}, v_{k,2})\}_{k=1}^{T}.$$

For each ray, the LiDAR return corresponds to the nearest intersection

$$t_j^\star = \min\Big\{\, \tau(r_j, \triangle_k) \;\Big|\; 1 \leq k \leq T,\ \tau \in [t_{\min}, t_{\max}] \Big\}, \tag{20}$$

where $\tau(r_j, \triangle_k)$ is the intersection parameter with triangle $\triangle_k$. The measured range equals $\rho_j = t_j^\star$ since $\|d_j\| = 1$.

GPU-accelerated ray-casting. We implement ray-casting entirely on the GPU by assigning one thread to each LiDAR beam, thereby transforming the inherently independent nature of beam propagation into massive parallelism. In our design, every thread traverses the preconstructed BVH and typically visits only $\mathcal{O}(\log T)$ nodes before reaching a small set of candidate triangles. Traversal is strictly guided by the best-so-far depth $t_j^\star$: nodes whose entry distance exceeds $t_j^\star$ are discarded together with their subtrees, and candidate triangles lying beyond this threshold are likewise excluded. This early-termination mechanism ensures that computation remains focused only on geometrically relevant regions.

Formally, the per-ray work can be written as

$$\mathrm{cost}(r_j) = C_{\mathrm{trav}}\, N_{\mathrm{nodes}}(r_j) + C_{\mathrm{tri}}\, K_j, \tag{21}$$

where C_{trav} and C_{tri} are the costs of a ray–AABB and ray–triangle test, $N_{\mathrm{nodes}}(r_j)$ is the number of BVH nodes visited, and K_j is the number of triangles tested at leaves. Summing over all rays gives

$$\mathcal{C}_{\mathrm{GPU}} = \sum_{j=1}^{N_r} \mathrm{cost}(r_j) \;\approx\; \mathcal{O}(N_r \cdot (\log T + \overline{K})), \tag{22}$$

with $\overline{K} = \frac{1}{N_r}\sum_j K_j \ll T$ in practice. This stands in contrast to the naive baseline

$$\mathcal{C}_{\mathrm{naive}} = \mathcal{O}(N_r \cdot T),$$

and explains why the GPU design achieves near-logarithmic scaling and millisecond-level simulation time even on million-triangle meshes.

Beyond hierarchical pruning, we incorporate several architectural optimizations to maximize GPU efficiency. Mesh vertices, triangle indices, and BVH bounds are organized in a structure-of-arrays layout, enabling coalesced memory accesses across warps. Frequently reused node bounds are cached in shared memory, which substantially reduces global memory traffic. Warp-synchronous traversal further enforces execution coherence, mitigating branch divergence among neighboring rays. Since beam queries are fully independent, the nearest-hit depth $t_j^\star$ and the corresponding intersection point $x_j^\star$ are written directly to global output buffers in a lock-free manner. Collectively, these design choices integrate algorithmic pruning with hardware-conscious optimization, ensuring that traversal, intersection, and memory access are all jointly accelerated, and rendering LiDAR-scale simulation feasible at millisecond latency even for million-triangle meshes.

V. EXPERIMENTS

A. Experiment Setup

Datasets. We evaluate on two benchmark scenes: one indoor and one outdoor. For each scene, we acquire a ground-truth (GT) watertight mesh via real LiDAR scanning followed by SLAM-based reconstruction, and we prepare a corresponding 3DGS asset of the same scene. All assets are registered to a common world frame and share identical LiDAR extrinsics. From a fixed sensor pose and scanning pattern, we render one LiDAR frame per scene for each of three sensor configurations (HDL64, OS128, VLP32), matching the beam layouts used in the tables.

Competitors. We compare two sources of geometry. Our method converts off-the-shelf 3DGS assets into a watertight mesh with our pipeline and renders a first-hit LiDAR frame under the fixed pose and scan pattern. The GT baseline renders a LiDAR frame from the mesh reconstructed from real LiDAR scans using the same pose and scan pattern. All point clouds are evaluated in the same coordinate frame.

Metrics. We report symmetric Chamfer Distance (CD; lower is better) and F-score, Precision, and Recall at standard distance thresholds (indoor and outdoor thresholds differ). Results are averaged over the two scenes and over the three LiDAR configurations. Distances are measured in meters.

Implementation details. All experiments are conducted on a workstation with an Intel W3545 CPU at 3.2 GHz and an NVIDIA RTX 4090 GPU. All timings and throughputs reported in this section are measured on this machine.

TABLE I **Quantitative comparison on indoor LiDAR simulation.** Results are averaged over three LiDAR types (HDL64, OS128, VLP32) by comparing simulated and ground-truth point clouds.

LiDAR	CD ↓	F-score ↑	Precision ↑	Recall ↑
HDL64	0.0034	0.9950	0.9985	0.9916
OS128	0.0034	0.9950	0.9956	0.9944
VLP32	0.0053	0.9918	0.9964	0.9872
Avg.	0.0041	0.9939	0.9968	0.9911

TABLE II **Quantitative comparison on outdoor LiDAR simulation**. Results are averaged over three LiDAR types (HDL64, OS128, VLP32) by comparing simulated and ground-truth point clouds.

LiDAR	CD ↓	F-score ↑	Precision ↑	Recall ↑
HDL64	0.0157	0.9816	0.9826	0.9806
OS128	0.0104	0.9867	0.9883	0.9853
VLP32	0.0250	0.9789	0.9844	0.9736
Avg.	0.0170	0.9824	0.9851	0.9798

B. Comparisons with Ground Truth

Indoor. Table I reports results for the indoor scene. Our simulation closely matches the GT-scan mesh, with CD in the 3–5 mm range (avg. 4.07 mm), mean F-score 0.994, Precision 0.997, and Recall 0.991. Among beam layouts, OS128 attains the lowest CD (0.003432 m) and the highest Recall (0.994356), while HDL64 yields the top F-score (0.995043) and Precision (0.9985). VLP32 is consistently weaker—expected from its sparser vertical sampling—yet still remains within a few millimeters of GT.

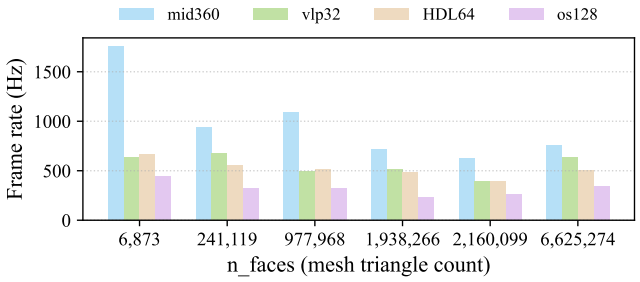

Fig. 5: LiDAR frame rate vs. mesh complexity.

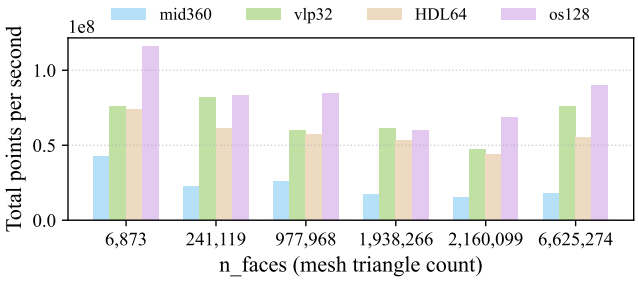

Fig. 6: LiDAR throughput vs. mesh complexity.

Outdoor. Table II summarizes the outdoor scene. Errors increase but remain small: average CD is 17.0 mm with F-score 0.982, Precision 0.985, and Recall 0.980. OS128 dominates across metrics (CD 0.010429 m; F-score 0.986726), reflecting the benefit of denser beams for long-range structure and complex occlusions. Compared with indoor, the gap is mainly due to longer ranges and clutter; nonetheless, the averages indicate our pipeline produces ranges that approach those rendered from meshes reconstructed by real LiDAR scans across distinct LiDAR geometries.

C. LiDAR Simulation Performance

We evaluate the simulation performance of four different LiDAR sensor configurations across six distinct 3DGS scenes; the horizontal axis in the plots corresponds to the number of faces (triangles) in the converted mesh model.

Frame rate. As shown in Fig. 5, owing to the BVH acceleration structure, across all evaluated 3DGS scenes (up to 6M Gaussian primitives) we observe no systematic degradation of simulation frame rate with increasing primitive count. This indicates that, within this scale, performance is effectively decoupled from the raw number of primitives. Instead, traversal efficiency is primarily governed by spatial distribution characteristics—such as clustering patterns, local density heterogeneity, and occlusion layering—rather than absolute cardinality. This demonstrates that our work achieves *far beyond real-time performance*, ranking as the fastest among peer methods, and highlights the efficiency advantage of the proposed pipeline.

Throughput. Fig. 6 reports point throughput as a function of mesh complexity. Despite lower frame rates, denser sensors (OS128, VLP32) in our work achieve significantly higher throughput, exceeding 10^8 points/s on lightweight meshes and sustaining $> 7.5 \times 10^7$ points/s even at multi-million triangle scales. This indicates that our method fully exploits BVH-accelerated parallelism and maintains high utilization across diverse LiDAR types.

D. Comparison with 3DGS Depth

Comparison of Depth and LiDAR Imaging Principles. Depth in 3D Gaussian Splatting is computed as an opacity-weighted expectation along the camera ray, yielding view-dependent averages rather than true geometric intersections. This causes two characteristic artifacts: edges and thin structures are blurred or widened due to splat blending, and under-constrained regions—such as back-facing or rarely observed surfaces, low-texture areas, and glossy materials—manifest as dropouts or holes in rendered depth maps (see Fig. 7). LiDAR, in contrast, reports the first-return geometric distance, a physically grounded metric that directly corresponds to scene geometry and avoids averaging artifacts.

Computational Efficiency. Depth rendering in 3D Gaussian Splatting requires per-pixel accumulation over many anisotropic Gaussians, with cost and memory scaling with both image resolution and splat footprint, making high-resolution rendering computationally expensive. LiDAR rendering instead performs BVH-accelerated ray–triangle intersections on a mesh. Its cost grows mainly with beam and triangle count, and the independence of rays allows highly efficient parallelization.

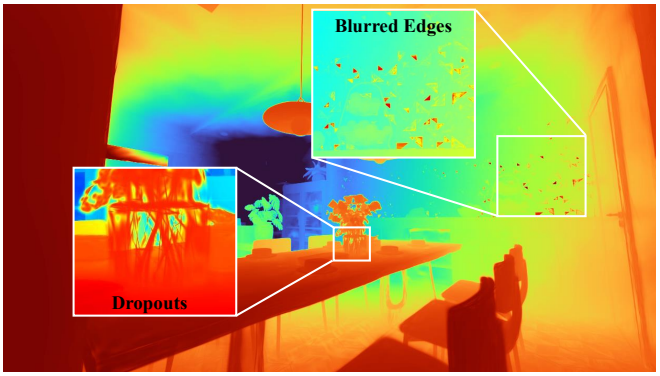

Fig. 7: **Edge blurring and depth dropouts in 3DGS depth maps.** The figure illustrates blurred object boundaries and missing depth values in 3DGS-rendered maps, indicating limitations in geometric accuracy and stability.

VI. Conclusion

We present FGGS-LiDAR, an ultra-fast GPU-accelerated LiDAR simulation framework for general 3DGS assets. Our fully GPU-resident framework operates directly on off-the-shelf 3DGS models without LiDAR supervision or COLMAP-style priors by passing 3DGS assets through BVH-based volumetric discretization and narrow-band TSDF, followed by isosurface extraction to create a watertight surface. We then perform BVH-accelerated per-ray

first-hit ranging in the LiDAR spherical image and achieve over 500 FPS for 200k+ rays in a 6M+ triangle scene. In both indoor and outdoor scenarios, LiDAR simulations based on meshes modeled by our method exhibit strong agreement with those derived from real-scanned meshes, as reflected by average Chamfer Distances of 4.07 mm indoors and 17.0 mm outdoors, alongside mean F-scores of 0.994 and 0.982, respectively. In addition, our method can serve as a plug-in LiDAR module for 3DGS-based simulators, integrating into existing pipelines to support large-scale sensor data generation.

Remaining limitations include residual internal cavities at high voxel resolutions, depending on the quality of the 3DGS assets, and high GPU memory usage; future work will improve memory scalability, robustness to low-quality assets, and add more realistic LiDAR sensor physics with closed-loop evaluation for dynamic scenes.

REFERENCES

[1] Y. Zhang, P. Shi, and J. Li, "Lidar-based place recognition for autonomous driving: A survey," *ACM Computing Surveys*, vol. 57, no. 4, pp. 1–36, 2024.

[2] H. Yin, X. Xu, S. Lu, X. Chen, R. Xiong, S. Shen, C. Stachniss, and Y. Wang, "A survey on global lidar localization: Challenges, advances and open problems," *International Journal of Computer Vision*, vol. 132, no. 8, pp. 3139–3171, 2024.

[3] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022.

[4] A. Charroud, K. El Moutaouakil, V. Palade, A. Yahyaouy, U. Onyekpe, and E. U. Eyo, "Localization and mapping for self-driving vehicles: a survey," *Machines*, vol. 12, no. 2, p. 118, 2024.

[5] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun, "Minos: Multimodal indoor simulator for navigation in complex environments," *arXiv preprint arXiv:1712.03931*, 2017.

[6] S. Manivasagam, S. Wang, K. Wong, W. Zeng, M. Sazanovich, S. Tan, B. Yang, W.-C. Ma, and R. Urtasun, "Lidarsim: Realistic lidar simulation by leveraging the real world," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11167–11176, 2020.

[7] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics: Results of the 11th international conference*, pp. 621–635, Springer, 2017.

[8] C. Li, Y. Ren, and B. Liu, "Pcgen: Point cloud generator for lidar simulation," *arXiv preprint arXiv:2210.08738*, 2022.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*, pp. 1–16, PMLR, 2017.

[10] 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," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[11] Z. Zheng, F. Lu, W. Xue, G. Chen, and C. Jiang, "Lidar4d: Dynamic neural fields for novel space-time view lidar synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5145–5154, 2024.

[12] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1389–1399, 2023.

[13] W. Xue, Z. Zheng, F. Lu, H. Wei, G. Chen, *et al.*, "Geonlf: Geometry guided pose-free neural lidar fields," *Advances in Neural Information Processing Systems*, vol. 37, pp. 73672–73692, 2024.

[14] J. Zhang, F. Zhang, S. Kuang, and L. Zhang, "Nerf-lidar: Generating realistic lidar point clouds with neural radiance fields," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 7178–7186, 2024.

[15] S. Huang, Z. Gojcic, Z. Wang, F. Williams, Y. Kasten, S. Fidler, K. Schindler, and O. Litany, "Neural lidar fields for novel view synthesis," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 18236–18246, 2023.

[16] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 139:1–139:14, 2023.

[17] Y. Liu, C. Luo, L. Fan, N. Wang, J. Peng, and Z. Zhang, "Citygaussian: Real-time high-quality large-scale scene rendering with gaussians," in *European Conference on Computer Vision*, pp. 265–282, Springer, 2024.

[18] G. Feng, S. Chen, R. Fu, Z. Liao, Y. Wang, T. Liu, B. Hu, L. Xu, Z. Pei, H. Li, *et al.*, "Flashgs: Efficient 3d gaussian splatting for large-scale and high-resolution rendering," in *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 26652–26662, 2025.

[19] M. T. I. SpatialVerse Research Team, "Interiorgs: A 3d gaussian splatting dataset of semantically labeled indoor scenes." `https://huggingface.co/datasets/spatialverse/InteriorGS`, 2025.

[20] D. Chen, H. Li, W. Ye, Y. Wang, W. Xie, S. Zhai, N. Wang, H. Liu, H. Bao, and G. Zhang, "Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[21] L. Radl, M. Steiner, M. Parger, A. Weinrauch, B. Kerbl, and M. Steinberger, "Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering," *ACM Transactions on Graphics (TOG)*, vol. 43, no. 4, pp. 1–17, 2024.

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

[23] Z. Qian, S. Wang, M. Mihajlovic, A. Geiger, and S. Tang, "3dgs-avatar: Animatable avatars via deformable 3d gaussian splatting," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5020–5030, 2024.

[24] Z. Yu, A. Chen, B. Huang, T. Sattler, and A. Geiger, "Mip-splatting: Alias-free 3d gaussian splatting," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 19447–19456, 2024.

[25] 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," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

[26] J. Jiang, C. Gu, Y. Chen, and L. Zhang, "Gs-lidar: Generating realistic lidar point clouds with panoramic gaussian splatting," in *International Conference on Learning Representations (ICLR)*, 2025.

[27] J. L. Schönberger, T. Price, T. Sattler, J.-M. Frahm, and M. Pollefeys, "A vote-and-verify strategy for fast spatial verification in image retrieval," in *Asian Conference on Computer Vision (ACCV)*, 2016.

[28] Y. Xu, K. Ye, T. Shao, and Y. Weng, "Animatable 3d gaussians for modeling dynamic humans," *Frontiers of Computer Science*, vol. 19, no. 9, p. 199704, 2025.

[29] Y. Wolf, A. Bracha, and R. Kimmel, "Gs2mesh: Surface reconstruction from gaussian splatting via novel stereo views," in *European Conference on Computer Vision*, pp. 207–224, Springer, 2024.

[30] A. GuÃŠdon, D. Gomez, N. Maruani, B. Gong, G. Drettakis, and M. Ovsjanikov, "Milo: Mesh-in-the-loop gaussian splatting for detailed and efficient surface reconstruction," *arXiv preprint arXiv:2506.24096*, 2025.

[31] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[32] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[33] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, 1996.

[34] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Seminal graphics: pioneering efforts that shaped the field*, pp. 347–353, 1998.