SplatSDF: Boosting SDF-NeRF via Architecture-Level Fusion with Gaussian Splats

Runfa Blark Li[1], Daniel George[1], Keito Suzuki[1], Bang Du[1], K. M. Brian Lee[1], Nikolay Atanasov[1], Truong Nguyen[1]

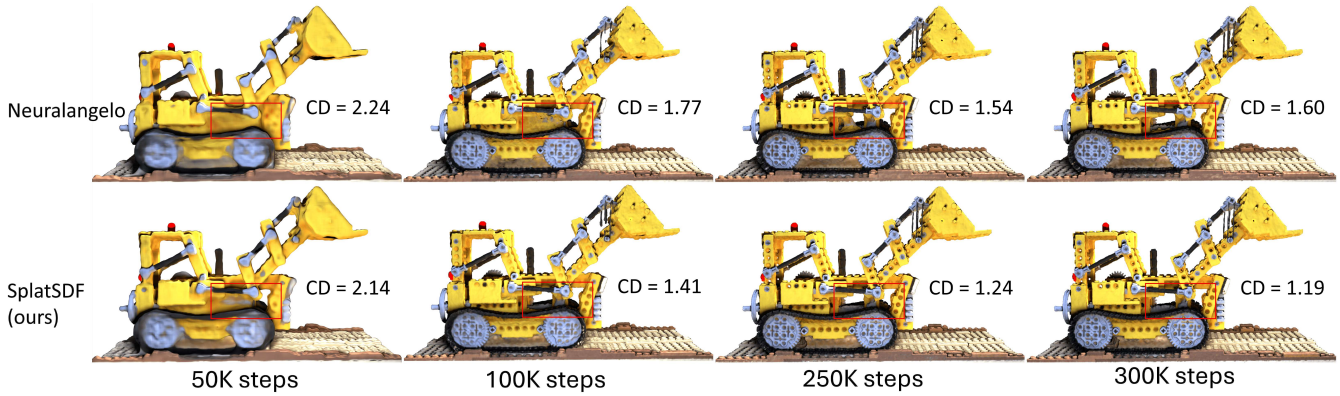

Fig. 1: SplatSDF accelerates SDF-NeRF training using a novel architecture that fuses 3D Gaussian Splat information. SplatSDF quickly caputres complex geometry, such as the holes in the red boxes, and converges $> 3\times$ faster compared to the best baseline, Neuralangelo. SplatSDF achieves better chamfer distance (CD) at 100k epochs/3.97 hours than Neuralangelo at 300K epochs/15.15 hours. Our code and more results are available on: https://blarklee.github.io/splatsdf/

***Abstract*— Signed distance-radiance field (SDF-NeRF) is a promising environment representation that offers both photo-realistic rendering and geometric reasoning such as proximity queries for collision avoidance. However, the slow training speed and convergence of SDF-NeRF hinder their use in practical robotic systems. We propose SplatSDF, a novel SDF-NeRF architecture that accelerates convergence using 3D Gaussian splats (3DGS), which can be quickly pre-trained. Unlike prior approaches that introduce a consistency loss between separate 3DGS and SDF-NeRF models, SplatSDF directly fuses 3DGS at an architectural level by consuming it as an input to SDF-NeRF during training. This is achieved using a novel *sparse 3DGS fusion* strategy that injects neural embeddings of 3DGS into SDF-NeRF around the object surface, while also permitting inference without 3DGS for minimal operation. Experimental results show SplatSDF achieves $3\times$ faster convergence to the same geometric accuracy than the best baseline, and outperforms state-of-the-art SDF-NeRF methods in terms of chamfer distance and peak signal to noise ratio, unlike consistency loss-based approaches that in fact provide limited gains. We also present computational techniques for accelerating gradient and Hessian steps by $3\times$. We expect these improvements will contribute to deploying SDF-NeRF on practical systems.**

I. INTRODUCTION

Signed distance fields (SDFs) are useful environment representation in robotics as they offer important geometric reasoning for environment reconstruction [1], planning and manipulation [2]–[6]. SDF-neural radiance field (SDF-NeRF) techniques further enable photorealistic rendering by ray-marching an opacity field derived from the SDF [7]. We are interested in developing efficient training techniques that allow using SDF-NeRF as a unified representation for both geometric and photometric reasoning.

A crucial challenge to this end is computational efficiency. Because the photometric accuracy of SDF-NeRF derives from volumetric rendering with ray marching, many epochs are required to distinguish object surfaces from free space. The ambiguity between object surface and free space can also lead to poor convergence with spurious 'ghost' artifacts.

We explore the use of 3D Gaussian splats (3DGS) [8]–[16] to accelerate the convergence of SDF-NeRFs. 3DGS can be trained expediently through rasterization, but falls short of providing proximity queries that are essential for robotics.

To combine the benefits of the two approaches, we propose *SplatSDF*, a novel approach that fuses 3DGS into SDF-NeRF at training time for enhanced convergence. Unlike concurrent work where 3DGS and SDF are separate models only linked through consistency losses [17], we fuse 3DGS into SDF-NeRF at an *architectural level* by consuming 3DGS as input to SDF-NeRF during training to guide convergence. Meanwhile, 3DGS is no longer required at inference time, thus yielding a minimal representation that provides both geometric and photometric accuracies. These benefits are realized through a novel *sparse 3DGS fusion strategy* that injects an embedding of 3DGS into SDF-NeRF only at the

This work was supported by the Ministry of Trade, Industry and Energy (MOTIE), Korea, under the Strategic Technology Development Program, supervised by the Korea Institute for Advancement of Technology (KIAT) [Grant No. P0026052].

Dept. of Electrical and Computer Engineering, University of California, San Diego (UCSD), La Jolla, CA, USA. {runfa, dlgeorge, k3suzuki, b7du, kmblee, natanasov, tqn001}@ucsd.edu

object surface.

Our experimental results show that SplatSDF achieves 3× faster convergence for the same geometric accuracy, as shown in Figure 1, and the converged results outperform state-of-the-art (SOTA) SDF-NeRF methods in terms of chamfer distance (CD) and peak signal-to-noise-ratio (PSNR). This is in contrast to other approaches that guide SDF using GS through consistency losses, which we found to actually achieve limited gains. We also present computational methods that speed up gradient and Hessian computation by 3×. Ablation studies demonstrate the effectiveness of using all 3DGS attributes, as opposed to treating 3DGS as a point cloud. In sum, our contributions are as follows:

- We propose *SplatSDF*, a novel SDF-NeRF architecture that uses GS to accelerate convergence.
- A novel sparse 3DGS fusion strategy effectively injects neural embedding of 3DGS into SDF-NeRF.
- Experimental results show that SplatSDF actually improves the training speed and photometric/geometric accuracy of SDF-NeRF unlike prior approaches to GS fusion.

II. RELATED WORK

To achieve photometric accuracy, volumetric rendering is required. Whereas NeRF [18]–[20] uses volumetric rendering to train MLPs that store the color and opacity of a scene, an SDF-NeRF replaces the opacity MLP with an MLP for the SDF of a scene [21]. For supervision with images, volumetric rendering is achieved by converting the SDF to opacity using a density function, such as the logistic distribution [21] or the Laplace distribution [7].

3DGS is a more recent photometric representation than NeRF that replaces volumetric rendering with rasterization for faster training. Therefore, a possible alternative direction to ours is to improve the geometric capabilities of 3DGS. In this regard, improvements of 3DGS, such as SUGAR [22], Gaussian surfels [23], GOF [24] and 2DGS [25] introduce consistency losses that improve geometric accuracy in terms of depth rendering. However, all of these methods require intermediate depth rendering for proximity queries or surface extraction, using methods such as Poisson surface reconstruction [22], [23], truncated SDF fusion on a voxel grid [25], or marching tetrahedra [24]. We focus on training an SDF-NeRF, which directly allows continuous proximity queries that are useful for robotics without rendering depth.

Some concurrent work also aims to leverage 3DGS for faster training of SDF-NeRF. These methods introduce a consistency loss between separate 3DGS and SDF-NeRF models, such as surface normal consistency in 3DGSR [17], depth consistency in GSDF [26]. In contrast to these methods, the proposed SplatSDF first pre-trains a 3DGS, and directly uses the pre-trained 3DGS as an input to SDF-NeRF at training time. This has two notable merits: we can exploit the fact that 3DGS can be pre-trained much quicker than SDF-NeRF, and the resulting performance significantly improves upon standalone SDF-NeRF, which, as our experiments show, is not the case with the consistency loss-based approaches.

III. PROBLEM STATEMENT

We assume that we are given RGB images with known camera poses and a sufficiently trained 3DGS [8] model $\mathcal{G}$ of the scene. Each Gaussian $G \in \mathcal{G}$ is parameterized by a mean $\mu(G)$, a covariance $\Sigma(G)$, opacity $\alpha(G)$, and spherical harmonics $SH(G)$. The RGB images may be given or rendered from the 3DGS model $\mathcal{G}$.

Our objective is to recover an SDF-NeRF, comprising an SDF $f_{\mathcal{S}} : \mathbb{R}^3 \to \mathbb{R}$ of the occupied space $\mathcal{S}$ in the scene, and a radiance field $\mathbf{c}(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}^3$ The SDF is defined as $f_{\mathcal{S}}(\mathbf{x}) = \pm \inf_{\mathbf{y} \in \partial\mathcal{S}} \|\mathbf{x} - \mathbf{y}\|$ with positive sign when $\mathbf{x} \notin \mathcal{S}$ and negative sign otherwise. This allows geometric reconstruction of the surface of $\mathcal{S}$ as the zero-level set of its SDF $\partial\mathcal{S} = \left\{\mathbf{x} \in \mathbb{R}^3 \mid f_{\mathcal{S}}(\mathbf{x}) = 0\right\}$. Our approach uses the 3DGS $\mathcal{G}$ during training but yields the SDF $f_{\mathcal{S}}$ as an MLP that can be queried independently of $\mathcal{G}$ at inference time.

Overview of approach: Figure 2 shows an overview of the proposed SplatSDF. Our contribution focuses on the SDF, whereas the radiance field is designed similarly to NeRF [18].

A 3DGS aggregator (Sec. IV-A) constructs a 3DGS embedding e_{gs}, which is used in conjunction with an SDF embedding e_{sdf} during training. Then, the per-point SDF is predicted as:

$$f_{\mathcal{S}}(\mathbf{x}) = f_{sdf}(Fuse(e_{sdf}(\mathbf{x}), e_{gs}(\mathbf{x}, \mathcal{G}))), \tag{1}$$

where f_{sdf} is an MLP, and $Fuse$ is the sparse 3DGS fusion module (Sec. IV-B).

At inference time, the 3DGS embedding e_{gs} is optional, and the SDF model can be queried as:

$$f_{\mathcal{S}}(\mathbf{x}) = f_{sdf}(e_{sdf}(\mathbf{x})). \tag{2}$$

In what follows, we concretely define our main contributions in the sparse 3DGS fusion and the 3DGS aggregator modules. The sparse 3DGS fusion module (Sec. IV-B) fuses the SDF embedding $e_{sdf}(\mathbf{x})$ and the 3DGS embedding $e_{gs}(\mathbf{x})$ near the geometry's surface. The 3DGS aggregator (Sec. IV-A) constructs per-Gaussian embeddings $e_g(G)$ that are combined into a 3DGS embedding e_{gs} by the sparse weighted 3DGS fusion module.

IV. SPLATSDF

A. 3DGS aggregator

The 3DGS aggregator constructs per-Gaussian embeddings $e_g(G)$ by combining the attributes of each Gaussian. The 3DGS attributes are important because they reflect shape beyond the center points, as illustrated in our results (Table 7). The 3DGS aggregator aligns the feature spaces to provide an explicit notion of local geometry beyond sparse center positions. To ensure consistency, we share the same hash-encoder h [27] between the SDF embedding e_{sdf} and the Gaussian embedding e_g, so that they are defined as:

$$\begin{aligned} e_{sdf}(\mathbf{x}) &= h(\mathbf{x}), \\ e_g(G) &= f_{agg}(h(\mu(G)), \Sigma(G), c(G), SH(G)), \end{aligned}$$

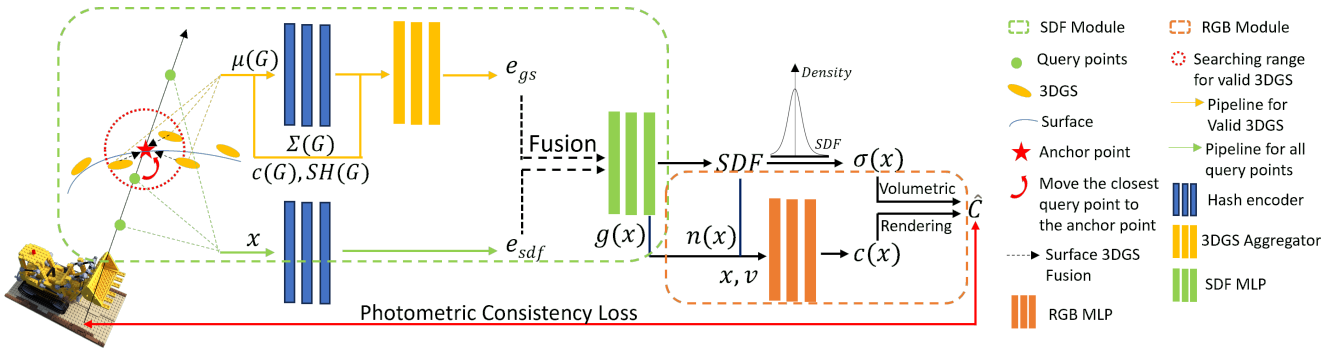

Fig. 2: **Overview**. SplatSDF takes posed RGB images and 3DGS to train an SDF-NeRF model. We use 3DGS-rendered depth to identify the anchor point and shift the closest query point to the anchor point. With a shared hash encoder, we construct query-point SDF embeddings e_{sdf} and 3DGS embeddings e_{gs}. Our method applies a *3DGS aggregator* to merge the 3DGS attributes: mean μ, covariance Σ, color c, and spherical harmonics SH. A novel *surface 3DGS fusion* strategy merges e_{gs} and e_{sdf} only around anchor points to regress the SDF value. The SDF value is converted to per-point density $\sigma(x)$. We take the geometric features $g(x)$, the surface normal from SDF $n(x)$, the query-point coordinates x, and the viewing angle v to estimate per-point color $c(x)$ and obtain the per-pixel color $\hat{C}$ by volumetric rendering, supervised with input images. Our contribution is the design of the *3DGS aggregator* and the *surface 3DGS fusion*.

where f_{agg} is an MLP, and μ, Σ, c, SH are the center coordinate, covariance, color and spherical harmonics parameters of G. Because the 3DGS aggregator shares the same hash encoder h between e_{sdf} and e_g, the two embeddings remain consistent, and hence the 3DGS accelerates training but is not required at inference time. We do not aggregate the GS opacity α in f_{agg} at this stage, because it plays an important role in the following fusion step.

B. Sparse 3DGS fusion

The sparse 3DGS fusion stage constructs 3DGS embedding $e_{gs}(\mathbf{x}, \mathcal{G})$ from the per-Gaussian embeddings $e_g(G)$, and fuses them with the SDF embeddings e_{sdf}. This implements the fusion function $Fuse(e_{sdf}(\mathbf{x}), e_{gs}(\mathbf{x}, \mathcal{G}))$ in (1).

Weighted Fusion of per-Gaussian Embeddings: Inspired by the alpha blending in 3DGS [8], we propose a novel weighted blending strategy to fuse the per-Gaussian embeddings $e_g(G)$ into a GS embedding $e_{gs}(\mathbf{x}, \mathcal{G})$ at each query point $\mathbf{x}$, as:

$$e_{gs}(\mathbf{x}, \mathcal{G}) = \frac{1}{K} \sum_{G \in \mathrm{KNN}(\mathbf{x}, \mathcal{G})} e_g(G) w(\mathbf{x}, G) \alpha(G). \quad (3)$$

Here, $\mathrm{KNN}(\mathbf{x}, \mathcal{G})$ denotes the K GS with means nearest to the query point $\mathbf{x}$, and $\alpha(G)$ is the opacity of Gaussian G. The weight $w(\mathbf{x}, G) = \exp\left(-\frac{1}{2}(\mathbf{x} - \mu(G))^T \Sigma(G)^{-1} (\mathbf{x} - \mu(G))\right)$ and opacity α balance the contribution of neighbor GS embeddings to the query-point GS embedding. Division by the number of selected Gaussians K normalizes the resulting embedding.

The weighted blending strategy is an extension of the 2D blending strategy in 3DGS [8] to 3D in the following sense. While 3DGS uses a projected 2D Gaussian weight function for blending colors in the image plane, our approach uses the 3D Gaussian weight function to blend the embeddings in the 3D space directly. Furthermore, the KNN algorithm can be seen as the 3D equivalent to frustum culling in the 2D projection step of 3DGS rasterization. Moreover, the original 3DGS uses 2D elliptical weighted average (EWA) volumetric splatting. Including opacity for 3D EWA is our novel design, which allows varying the contribution of each Gaussian based on its opacity in the scene. This is particularly useful for handling overlapping or less reliable Gaussians, as well as regions with varying densities.

In our implementation, we accelerated the KNN algorithm by hashing the GS $\mathcal{G}$ and the set of query points $\mathbf{X} = \{\mathbf{x}\}$ into L^3 voxels, inspired by PointNeRF [28]. Doing so decreases the computational complexity from $O(|\mathbf{X}||\mathcal{G}|)$ in the naive case to $O(\frac{|\mathbf{X}||\mathcal{G}|}{L^3})$, where $|\mathbf{X}|$ and $|\mathcal{G}|$ are the number of query points and the number of Gaussians. Since we only consider Gaussians belonging to the same voxel as the query point, we are implicitly imposing a radius constraint in addition to the K-nearest requirement.

Surface 3DGS Fusion: To fuse the GS embedding $e_{gs}(\mathbf{x}, \mathcal{G})$ with the SDF embedding $e_{sdf}(\mathbf{x})$, one possibility is to train an MLP that concatenates both GS and SDF embeddings and regresses another embedding of the same dimension. However, we present a fusion approach that is simpler but significantly more effective and efficient.

Our approach begins from the observation that all query points $\mathbf{x}$ lie along a ray $\mathbf{r}$ during training. We first compute an anchor point $\mathbf{x_r}$, which we define as the first intersection between the ray and the surface. This can be computed using the depth value rendered from the GS $\mathcal{G}$. Then, for each ray, we replace the closest query point to the anchor point $\mathbf{x_r}$, with the anchor point $\mathbf{x_r}$ itself, and take the GS embedding $e_{gs}(\mathbf{x_r}, \mathcal{G})$ as the output. For all other points, we use the SDF embedding $e_{sdf}(\mathbf{x})$. Thus, we merely *'replace'* the SDF embedding with the GS embedding at the anchor point, rather than *'concatenating'* embeddings at all query points.

This fusion strategy is effective because the GS embedding

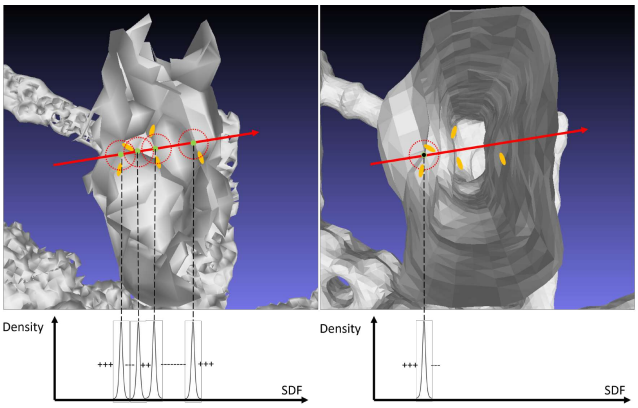

Fig. 3: **Dense 3D Fusion vs Surface 3D Fusion**. Left: Dense 3DGS fusion on all valid query points (green points). Right: Surface 3DGS fusion only on the anchor point (black point). Fusing query points inside of surfaces using spurious GS (orange ellipsoids) far from the true surface leads to bumpy surface artifacts.

is used only near the surface. This avoids spurious Gaussian blobs that are found further from the surface, which is a common problem in 3DGS. Moreover, the computational complexity is dramatically reduced compared to a dense fusion along the entire ray since we only compute the GS embedding at one anchor point per ray.

This simple replacement with GS embeddings near the surface at the anchor points leads to notable improvements in convergence speed and accuracy over SDF-NeRF without GS. Figure 3 shows a qualitative comparison of the proposed strategy against a 'dense' fusion via concatenation and MLP regression over both GS and SDF embeddings. The first row of Figure 3 shows that the dense fusion approach (left) leads to bumpy surface artifacts. Further analysis shows that the zero-crossings of the SDF (i.e. peaks of density) align with spurious Gaussians, as shown in the second row. This indicates that spurious Gaussians lead to errors in SDF. Thus, using the GS embedding at query points far from the surface incorporates spurious Gaussians to the embedding, contributing erroneous density. Although spurious Gaussians do not cause significant errors when projected to 2D images, they can cause errors in our SDF model as we fuse directly in 3D. Our sparse 3DGS fusion strategy mitigates this issue.

C. Training

Our SplatSDF model is trained for photometric accuracy by supervising volumetrically rendered images against the target images. To this end, we construct a radiance field that estimates the per query-point radiance $\mathbf{c}(\mathbf{x})$ using an RGB MLP f_{color}, whose inputs are the query position $\mathbf{x}$, viewing direction $\mathbf{v}$, geometric features $g(\mathbf{x})$, and the normals $\nabla_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x})$ (i.e., the SDF gradient):

$$\mathbf{c}(\mathbf{x}) = f_{color}(\mathbf{x}, \mathbf{v}, g(\mathbf{x}), \nabla_{\mathbf{x}} f_{\mathcal{S}}(x)). \tag{4}$$

For simplicity, we do not explicitly input the GS $\mathcal{G}$ into the RGB MLP f_{color} because the geometric features $g(\mathbf{x})$ and the normals $\nabla_{\mathbf{x}} f_{\mathcal{S}}(\mathbf{x})$ are already derived from the GS $\mathcal{G}$.

For volumetric rendering, we convert the per-point SDF $f_{\mathcal{S}}(\mathbf{x})$ to per-point opacity $\sigma(\mathbf{x})$ using the logistic distribution $\phi_s(d) = se^{-sd}/(1+e^{-sd})^2$, where s is the inverse of standard deviation, and d is the SDF value at the query point x. The parameter s is learnable, and is expected to approach zero as the SDF-MLP f_{sdf} converges. The opacity is used in conjunction with the color module to volumetrically render images for supervision:

$$\hat{\mathbf{C}}(\mathbf{r}) = \sum_{\mathbf{x}_i \in \mathbf{r}} T_i \left[1 - \exp\left(-\sigma\left(\mathbf{x}_i\right)\delta_i\right)\right] \mathbf{c}(\mathbf{x}_i), \tag{5}$$

where $\delta_i = \|\mathbf{x}_{i+1} - \mathbf{x}_i\|_2$ is the distance between adjacent query points along the ray $\mathbf{r}$, and $T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$ denotes the accumulated transmittance.

Our method is trained with the same losses as Neuralangelo [29]: L1 photometric loss, Eikonal loss, and curvature loss. To show the effectiveness of our architecture, we do not use any auxiliary losses with depth or normal priors.

V. EXPERIMENTS

A. Datasets & Implementation Details

We use the DTU [30] and NeRF Synthetic datasets [18] for training and evaluation. We use 12 scenes from the DTU dataset which contains either 49 or 64 posed images per-scene obtained by a robot-held monocular RGB camera and the point cloud ground truth obtained from a structured-light scanner. For the NeRF Synthetic Dataset, we use 5 objects with 100 posed images for each scene. We use the "Lego" scene for ablation study.

We train the 3DGS [8] with the GS centers fixed at input pointcloud to retain geometric fidelity. As a reminder, we fix the well-trained GS and do not jointly optimize with SDF-NeRF, unlike concurrent work that jointly optimizes SDF and GS [17], [26]. Separate training of GS and SDF is beneficial because we can exploit the fast training speed of GS independently of SDF, while the end result of SDF-NeRF is not only more geometrically accurate (i.e. low CD) than GS, but also as photometrically accurate (i.e. high PSNR) as GS. Similarly, PointNeRF [28] uses accurate pointcloud to boost NeRF, but not NeRF to improve the pointcloud.

For our 3DGS aggregator, we use a 3-layer MLP where the 2nd layer concatenates the upper-triangle of the GS covariance and the 3rd layer concatenates the GS color and SH. We do not use object segmentation masks as in some previous works, but instead randomly sample 512 pixels for each view, and sample 128 points per ray. We follow Neuralangelo [29] to sample foreground and background points separately in the two SDF-MLP. We use uniform sampling for the background with 32 points and a coarse-to-fine sampling [18] for the foreground with 96 points. We force the anchor points to be on the foreground by only implementing the surface GS fusion on the foreground NeRF.

We use CD in mm as the geometric evaluation metric. As per previous work, we sample 3D grid coordinates to estimate the SDF and use Marching Cubes to get the surface

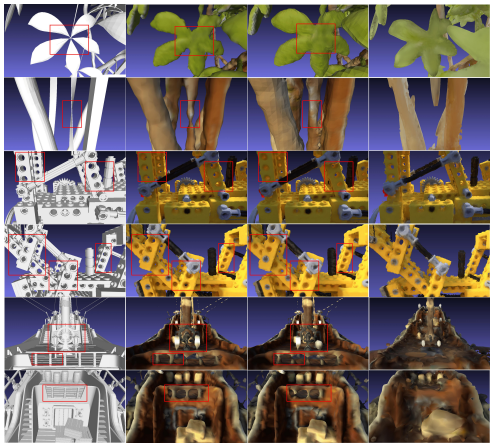

Fig. 4: **Surface Mesh Comparison** on the NeRF Synthetic Dataset. Left to right: Ground Truth mesh (color is not available), SplatSDF, Neuralangelo, SuGAR. Row 1-2: Ficus. Row 3-4: Lego. Row 5-6: Ship. Zoom in to check details in red boxes. No red boxes for SuGAR since it is overall worse than SDF-NeRFs. We only show zoom-in details in this figure.

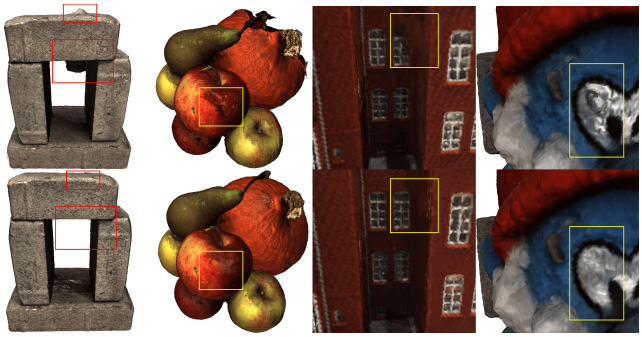

Fig. 5: **Qualitative comparisons on the DTU Dataset.** Top row: Neuralangelo [29]. Bottom row: Our SplatSDF. We use the same models as Quantitative results from Table I to generate the visual results. Zoom in to check the details in the red or yellow bounding boxes.

mesh. We then sample points from the mesh and compare with ground truth points to compute the CD. We use PSNR as the photometric evaluation metric. For the NeRF Dataset, we train on the standard training split with 100 images per scene and test on the standard testing split with 200 images per scene. For the DTU dataset, since there are no standard train/test splits, we train/test on the same images per-scene.

B. Qualitative Results

We present qualitative comparisons against Neuralangelo [29], which we found to be the strongest baseline. Figure 1 shows that our SplatSDF achieves faster and more accurate convergence. SplatSDF only takes 100K steps to get $CD = 1.41$, indicating > 3 times faster convergence compared to Neuralangelo which takes 300K steps to get $CD = 1.60$. SplatSDF takes 3.97 hours for 100K steps, which already

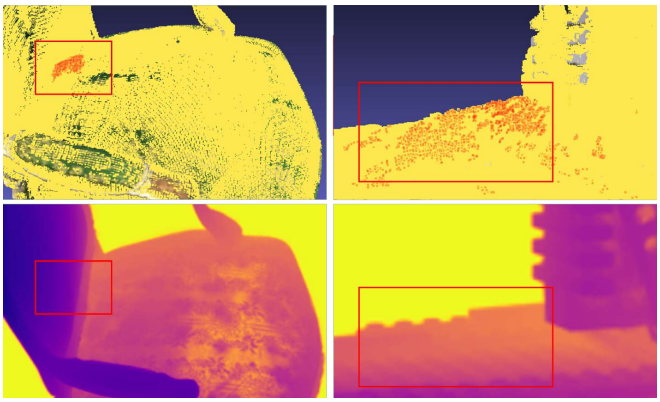

Fig. 6: **Tolerance to erroneous 3DGS initialized from noisy point cloud**. First row: GS center in yellow overlap with the estimated surface mesh. Noisy GS centers is in red and in red boxes. Second row: No noise in the red box from rendered depth.

achieves better CD than Neuralangelo trained to 300K steps taking 15.15 hours. SplatSDF also achieves better accuracy at convergence, capturing difficult shapes and details. While SDF-NeRF's iso-surface is always initialized as a unit sphere and "pulled" inside to fit to the convex surface, it is common to see it under-fitting to concave surfaces with small/thin details. This is because using the visual momentum alone from previous methods tends to smooth and blur the iso-surface and is insufficient to "pull" it to complex shapes. However, our SplatSDF amends this under-fitting with the novel architecture-level GS fusion. An example is shown in the red box in Figure 1 where our model quickly captures the holes, whereas previous methods do not. All results in Figure 4 are trained to converge and the details in the red boxes validate the improvements, such as the thin leaves and stems, the small holes in the Lego, the helm, the white lamp and the rail in the ship. We also include a comparison with the SOTA 3DGS-based surface reconstruction work SuGAR [22]. Although SuGAR achieves faster surface reconstruction, its quality is lower than SOTA SDF-NeRFs since they transform the shape of the Gaussian primitives to fit the surfaces causing noisy artifacts and missing details. Moreover, it cannot estimate the distance field for arbitrary 3D coordinates. Our superior results on DTU as shown in Figure 5 further validates the performance on real dataset, we show the colored mesh, not the rendered RGB view, the defects of Neuralangelo directly shows the geometric error instead of the photometric one.

Tolerance to noisy initialization Figure 6 shows that our SplatSDF can tolerate errors and noise in GS. We attribute this robustness to two reasons: 1. As shown in the second row of Figure 6, GS-rendered depth eliminates errors from erroneous GS centers because we use volumetric rendering rather than surface rendering and sstimate accurate anchor points to fuse correct surface 3DGS; 2. SDF-NeRF itself tends to under-fit to complex shapes by smoothing and blurring details, which alleviates the errors introduced by

TABLE I: **Quantitative Results on the DTU Dataset** (Chamfer Distance in mm ↓). Yellow is the best and pink is the second best. Our SplatSDF achieves the best geometric accuracy. All results are from latest papers and we validate the results of the baseline Neuralangelo. See Supplementary for the source of each result.

	Scan ID	24	37	40	55	63	65	69	83	105	106	110	114	Mean
Traditional	COLMAP [31]	0.81	2.05	0.73	1.22	1.79	1.58	1.02	3.05	2.05	1.00	1.32	0.49	1.43
	NeRF [18]	1.90	1.60	1.85	0.58	2.28	1.27	1.47	1.67	1.07	0.88	2.53	1.06	1.51
SDF-NeRF	UNISURF [32]	1.32	1.36	1.72	0.44	1.35	0.79	0.80	1.49	0.89	0.59	1.47	0.46	1.06
	MVSDF [33]	0.83	1.76	0.88	0.44	1.11	0.90	0.75	1.26	1.35	0.87	0.84	0.34	0.94
	VolSDF [7]	1.14	1.26	0.81	0.49	1.25	0.70	0.72	1.29	0.70	0.66	1.08	0.42	0.88
	NeuS [21]	1.00	1.37	0.93	0.43	1.10	0.65	0.57	1.48	0.83	0.52	1.20	0.35	0.87
	NeuS-12 [21]	0.93	1.07	0.81	0.38	1.02	0.60	0.58	1.43	0.78	0.57	1.16	0.35	0.81
	HF-NeuS [34]	0.76	1.32	0.70	0.39	1.06	0.63	0.63	1.15	0.80	0.52	1.22	0.33	0.79
	MonoSDF [35]	0.66	0.88	0.43	0.40	0.87	0.78	0.81	1.23	0.66	0.66	0.96	0.41	0.73
	RegSDF [36]	0.60	1.41	0.64	0.43	1.34	0.62	0.60	0.90	1.02	0.60	0.59	0.30	0.75
	PET-NeuS [37]	0.56	0.75	0.68	0.36	0.87	0.76	0.69	1.33	0.66	0.51	1.04	0.34	0.71
	OAV [38]	1.92	2.35	1.96	1.11	1.83	2.01	1.30	1.53	1.50	0.71	1.56	0.83	1.55
	TUVR [39]	0.83	1.06	0.57	0.40	1.00	0.62	0.62	1.41	0.94	0.57	1.07	0.35	0.79
	DebSDF [40]	0.71	0.94	0.46	0.39	1.05	0.61	0.59	1.49	0.88	0.61	1.05	0.34	0.76
	NeuralWarp [41]	0.49	0.71	0.38	0.38	0.79	0.81	0.82	1.20	0.68	0.66	0.74	0.41	0.67
	Geo-NeuS [42]	0.46	0.85	0.38	0.43	0.89	0.50	0.50	1.26	0.66	0.52	0.82	0.31	0.63
	Neuralangelo [29]	0.37	0.72	0.35	0.35	0.87	0.54	0.53	1.29	0.73	0.47	0.74	0.32	0.61
	SplatSDF (ours)	0.35	0.67	0.32	0.31	0.84	0.54	0.55	1.16	0.65	0.47	0.75	0.31	0.58
GS-based Surface Reconstruction	Scaffold-GS [43]	7.23	6.23	6.48	7.44	8.17	4.27	5.78	5.45	6.36	5.05	5.95	6.32	6.23
	3DGS [8]	2.14	1.53	2.08	1.68	3.49	2.21	1.43	2.07	1.75	1.79	2.55	1.53	2.02
	SuGAR [22]	1.47	1.33	1.13	0.61	2.25	1.71	1.15	1.63	1.07	0.79	2.45	0.98	1.38
	GaussianSurfels [23]	0.66	0.93	0.54	0.41	1.06	1.14	0.85	1.29	0.79	0.82	1.58	0.45	0.88
	2DGS [25]	0.48	0.91	0.39	0.39	1.01	0.83	0.81	1.36	0.76	0.70	1.40	0.40	0.79
	GausianField [24]	0.50	0.82	0.37	0.37	1.12	0.74	0.73	1.18	0.68	0.77	0.90	0.42	0.72
GS-guided SDF-NeRF	3DGSR [17]	0.68	0.84	0.70	0.39	1.16	0.87	0.77	1.48	0.87	0.69	0.80	0.42	0.81
	GSDF [26]	0.59	0.94	0.46	0.38	1.30	0.77	0.73	1.59	0.76	0.59	1.22	0.38	0.81

noisy 3DGS centers.

C. Quantitative Results

Table I shows a comparison on the DTU dataset over three categories. Our model outperforms the best baseline Neuralangelo [29] and achieves the lowest CD over all previous works. We analyze two reasons why our method performs worse than the baseline on a few scenes. The first reason is that the "anchor points" estimated from 3DGS rendered depth are inaccurate in some areas and the 3DGS quality is insufficient. The second reason is that the DTU dataset contains erroneous point clouds in some regions. The 3DGS-based surface reconstruction methods generally train faster but their accuracy is lower than SOTA SDF-NeRF methods, as can be seen in Table I and Figure 4. We also show the results from two concurrent methods, GSDF [26] and 3DGSR [17], which use 3DGS to guide SDF-NeRF. Their results show that loss-level connections alone do not provide improvements over SDF-NeRF, where our architecture-level fusion method does. We conduct more comparisons on the NeRF Synthetic Dataset (Table II), and the results show that our SplatSDF outperforms SOTA methods on both geometric and photometric accuracy, especially our baseline Neuralangelo [29]. The results in Table II further show that our model improves SDF-NeRF not only on geometric accuracy, but also on photometric accuracy.

D. Ablation Study

We conduct ablation studies (Figure 7) on the influence of three factors - sparse GS embedding fusion, the use of GS for depth images, and the use of GS over point clouds. All ablation variants outperform the strongest baseline, Neuralangelo, which shows the benefit of fusing pre-trained GS.

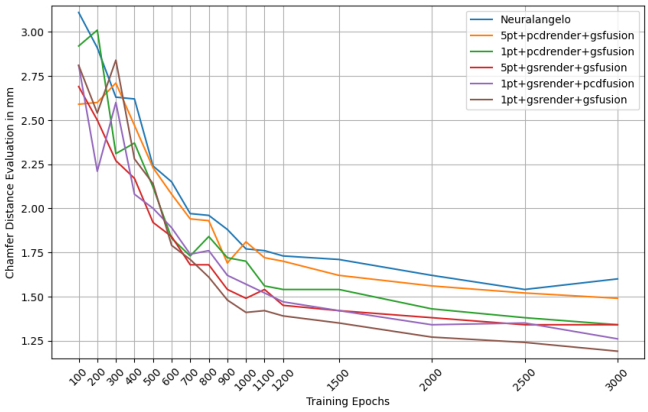

Fig. 7: **Ablation study of geometric accuracy on "Lego"**. All variations of our methods achieve faster convergence to better accuracy than the baseline "Neuralangelo".

Firstly, to examine the importance of Surface Fusion over Dense Fusion, we consider fusing five closest query points to the anchor point (5pt variants, shown in orange and red in Figure 7) as opposed to fusing only the anchor point (1pt variants, shown in green and brown in Figure 7). The results in Figure 7 show that fusing only the anchor point as proposed outperforms fusing more query points near the anchor point, consistent with the observation in Figure 3.

Secondly, we compare deriving anchor points based on

TABLE II: **Quantitative Results on NeRF Synthetic Dataset**. The top part shows the geometric accuracy in Chamfer Distance in mm ↓, and the bottom part shows the photometric accuracy in PSNR ↑. Yellow is the best and pink is the second best. Our SplatSDF achieves the best overall accuracy.

	Chair	Ficus	Lego	Mic	Ship	Mean
VolSDF [7]	1.18	3.01	2.26	1.13	6.42	2.80
NeuS [21]	3.99	0.94	2.56	1.00	5.38	2.77
NeRO [44]	1.27	1.22	1.90	0.87	4.95	3.72
BakedSDF [45]	1.83	10.9	1.13	0.84	3.88	3.32
NeRF2Mesh [46]	1.62	0.65	1.93	0.78	2.20	1.44
RelightableG [47]	3.65	1.26	1.63	1.76	3.35	2.33
HF-NeuS [34]	0.69	1.12	0.94	0.72	2.18	1.13
3DGSR [17]	1.01	0.69	1.35	1.15	3.35	1.51
Neuralangelo [29]	0.56	1.12	1.60	0.78	0.78	0.97
SplatSDF (ours)	0.71	0.91	1.19	0.75	0.72	0.86
NeRF [18]	33.00	30.15	32.54	32.91	28.34	31.39
VolSDF [7]	25.91	24.41	26.99	29.46	25.65	26.48
NeuS [21]	27.95	25.79	29.85	29.89	25.46	27.79
HF-NeuS [34]	28.69	26.46	30.72	30.35	25.87	28.42
PET-NeuS [37]	29.57	27.39	32.40	33.08	26.83	29.85
NeRO [44]	28.74	28.38	25.66	28.64	26.55	27.60
BakedSDF [45]	31.65	26.33	32.69	31.52	27.55	29.95
NeRF2Mesh [46]	34.25	30.08	34.90	32.63	29.47	32.27
Mip-NeRF [48]	35.14	33.29	35.70	36.51	30.41	34.21
3DGS [8]	35.36	34.87	35.78	35.36	30.80	34.43
Ins-NGP [27]	35.00	33.51	36.39	36.22	31.10	34.44
Neuralangelo [29]	34.72	35.91	33.20	36.79	31.45	34.41
SplatSDF (ours)	34.73	36.15	33.24	37.06	31.47	34.53

GS-rendered depth ("gsrender", shown in orange and green in Figure 7) and pointcloud-rendered depth ("pcdrender", shown in red and brown in Figure 7). It can be seen from Figure 7 that using GS-rendered depth outperforms the pointcloud-rendered depth. This is because the depth point cloud from MVSNet is noisy, which leads to inaccurate surface anchor points. In contrast, the depth image from a well-trained GS accurately estimates the anchor points. This is consistent with the observation that SplatSDF can tolerate noisy point cloud in Figure 6.

Lastly, we compare the proposed GS fusion ("gsfusion", shown in purple in Figure 7) against point cloud fusion ("pcdfusion", shown in brown in Figure 7). The point-cloud fusion baseline is implemented by removing the GS covariance and SH features from the GS aggregator, and removing the GS covariance and opacity terms in the distance weight for 3DGS fusion. The results in Figure 7 shows that the proposed GS fusion approach outperforms pointcloud fusion, which illustrates the importance of using all GS features, including covariance, opacity and SH coefficients.

E. Training Acceleration

Fast training is important for robotics applications. Our proposed SplatSDF reduces the steps for convergence. Here, we investigate how to accelerate individual steps.

We found the main bottleneck to be the computation of first- and second-order derivatives (Gradient & Hessian (GH), Table III). To accelerate the derivative computation, we first followed NeuS2 [49] - the fastest implementation available. NeuS2 attributes the speedup to the use of a closed-form second-order approximation using TinyCUD-

TABLE III: **Average computation time of each module (ms)** on a single 3090Ti GPU over 1k iterations. GH(P)/GH(N)/GH(FD): Gradient/Hessian with **P**ytorch vs. **N**eus2 TCNN VS **F**inite difference (proposed). SDF/SDF(N): Pytorch vs. **N**eus2 TCNN. RGB/RGB(N): **P**ytorch VS **N**eus2 TCNN. RS: ray sampling. GS: GS aggregator, KNN, & sparse fusion. AP: anchor points sampling.

GH(P)	GH(N)	GH(FD)	SDF	SDF(N)	RGB	RGB(N)	RS	GS	AP
19.95	18.64	6.02	3.19	3.12	1.13	0.98	11.16	6.44	0.03

ANN (TCNN) for the backpropagation of SDF and RGB MLPs. We integrated TCNN from NeuS2 into our PyTorch for the SDF and RGB MLPs in both forward & backward with NeuS2's purely CUDA C++. However, when testing over 1000 iterations on a 3090 Ti GPU (Table III), we found the speeds are nearly identical under the same architecture (19.95 ms to 18.64 ms), in contrary to the two orders of magnitude acceleration claimed in [49]. Moreover, using TCNN as MLP implementation also yields minor gains in forward steps (SDF: 3.19→3.12 ms; RGB: 1.13→0.98 ms). Thus, we argue that NeuS2's fast speed stems from a carefully integrated stack of CUDA C++ kernel implementations of ray marching, occupancy skipping, and memory layout optimization, rather than the use of TCNN. We defer translating our method to CUDA C++ to future work.

Instead, we found that combining TCNN with a batched central finite difference (FD) approximation provides a significant computation speed improvement with minimal modifications to our PyTorch implementation. Inspired by [50], we compute surface normals (gradient) and the diagonal Hessian via batched central finite differences (FD): For each point $x \in \mathbb{R}^3$, we evaluate the SDF at six offsets $\{x \pm \varepsilon e_i\}_{i=1}^3$ in a single parallelized TCNN forward step by packing the offsets along the channel dimension, where e_i are unit vectors along each axes, and ε is a small offset. This ensures that only first-order (no second-order) derivatives are taken during backpropagation. As Table III shows, using the batch FD approximation leads to a significant speed-up, from 19.95 ms to 6.02 ms, which is 3.31× faster. We will open-source all of our implementation.

VI. CONCLUSIONS

We proposed SplatSDF, a novel SDf-NeRF model with an architecture-level sparse 3DGS fusion to take advantage of the 3DGS attributes and accurately manipulate SDF embeddings on surface query points. Our SplatSDF significantly improves both the accuracy and efficiency of the SDF-NeRF on geometric and photometric accuracy, and the convergence speed. Future work lies in real-time training of SDF-NeRF with frozen 3DGS aggregator and online updates to 3DGS.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Trade, Industry and Energy (MOTIE), Korea, under the Strategic Technology Development Program, supervised by the Korea Institute for Advancement of Technology (KIAT) [Grant No. P0026052].

REFERENCES

[1] R. Li, U. Mahbub, V. Bhaskaran, and T. Nguyen, "Monoselfrecon: Purely self-supervised explicit generalizable 3d reconstruction of indoor scenes from monocular rgb views," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2024, pp. 656–666.

[2] K. M. B. Lee, Z. Dai, C. L. Gentil, L. Wu, N. Atanasov, and T. Vidal-Calleja, "Safe bubble cover for motion planning on distance fields," 2024. [Online]. Available: https://arxiv.org/abs/2408.13377

[3] D. M. Kim, K. M. B. Lee, Y. H. Seo, N. Raicevic, R. B. Li, K. Long, C. S. Yoon, D. M. Kang, B. J. Lim, Y. P. Kim, N. Atanasov, T. Nguyen, S. W. Jun, and Y. W. Kim, "A shared-autonomy construction robotic system for overhead works," in *ICRA workshop*, 2025.

[4] R. Li and T. Nguyen, "Monoplflownet: Permutohedral lattice flownet for real-scale 3d scene flow estimation with monocular images," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer Nature Switzerland, 2022, pp. 322–339.

[5] R. B. Li and T. Nguyen, "Sm3d: Simultaneous monocular mapping and 3d detection," in *2021 IEEE International Conference on Image Processing (ICIP)*, 2021.

[6] R. B. Li, D. Kim, X. Liu, K. Suzuki, D. Bhatt, N. Raicevic, X. Lin, K. M. B. Lee, N. Atanasov, and T. Nguyen, "Physgraph: Physically-grounded graph-transformer policies for bimanual dexterous hand-tool-object manipulation," 2026. [Online]. Available: https://arxiv.org/abs/2603.01436

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

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

[9] H. Nguyen, R. Li, A. Le, and T. Nguyen, "Waveletgaussian: Wavelet-domain diffusion for sparse-view 3d gaussian object reconstruction," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2025.

[10] H. Nguyen, R. B. Li, A. Le, and T. Nguyen, "Dwtgs: Rethinking frequency regularization for sparse-view 3d gaussian splatting," in *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, 2025.

[11] X. Liu, R. B. Li, K. Suzuki, and T. Nguyen, "Image-conditioned 3d gaussian splat quantization," *IEEE Access*, 2025.

[12] H. Nguyen, A. Le, B. R. Li, and T. Nguyen, "From coarse to fine: Learnable discrete wavelet transforms for efficient 3d gaussian splatting," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, October 2025, pp. 3139–3148.

[13] R. B. Li, M. Shaghaghi, K. Suzuki, X. Liu, V. Moparthi, B. Du, W. Curtis, M. Renschler, K. M. B. Lee, N. Atanasov, and T. Nguyen, "Dynagslam: Real-time gaussian-splatting slam for online rendering, tracking, motion predictions of moving objects in dynamic scenes," in *WACV*, 2026.

[14] K. Suzuki, B. Du, R. B. Li, K. Chen, L. Wang, P. Liu, N. Bi, and T. Nguyen, "Openhuman4d: Open-vocabulary 4d human parsing," in *British Machine Vision Conference (BMVC)*, 2025.

[15] K. Suzuki, B. Du, G. Krishnan, K. Chen, R. B. Li, and T. Nguyen, "Open-vocabulary semantic part segmentation of 3d human," in *International Conference on 3D Vision (3DV)*, 2025.

[16] K. Suzuki, B. Du, K. Chen, R. Li, and T. Nguyen, "Thp3d: Text-driven multi-granularity 3d human parsing," in *ECCV 2024 Workshops*, 2024.

[17] X. Lyu, Y.-T. Sun, Y.-H. Huang, X. Wu, Z. Yang, Y. Chen, J. Pang, and X. Qi, "3dgsr: Implicit surface reconstruction with 3d gaussian splatting," *ACM Trans. Graph.*, 2024.

[18] 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," in *ECCV*, 2020.

[19] H. Nguyen, B. R. Li, and T. Nguyen, "Dwtnerf: Boosting few-shot neural radiance fields via discrete wavelet transform," 2025. [Online]. Available: https://arxiv.org/abs/2501.12637

[20] B. Du, R. B. Li, C. Du, and T. Nguyen, "Glossgau: Efficient inverse rendering for glossy surface with anisotropic spherical gaussian," 2025. [Online]. Available: https://arxiv.org/abs/2502.14129

[21] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *NeurIPS*, 2021.

[22] A. Guédon and V. Lepetit, "Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering," *CVPR*, 2024.

[23] P. Dai, J. Xu, W. Xie, X. Liu, H. Wang, and W. Xu, "High-quality surface reconstruction using gaussian surfels," in *SIGGRAPH*, 2024.

[24] Z. Yu, T. Sattler, and A. Geiger, "Gaussian opacity fields: Efficient adaptive surface reconstruction in unbounded scenes," *ACM Transactions on Graphics*, 2024.

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

[26] M. Yu, T. Lu, L. Xu, L. Jiang, Y. Xiangli, and B. Dai, "Gsdf: 3dgs meets sdf for improved rendering and reconstruction," in *NeurIPS*, 2025.

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

[28] Q. Xu, Z. Xu, J. Philip, S. Bi, Z. Shu, K. Sunkavalli, and U. Neumann, "Point-NeRF: Point-based neural radiance fields," in *CVPR*, 2022.

[29] Z. Li, T. Müller, A. Evans, R. H. Taylor, M. Unberath, M.-Y. Liu, and C.-H. Lin, "Neuralangelo: High-fidelity neural surface reconstruction," in *CVPR*, 2023.

[30] R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanaes, "Large scale multi-view stereopsis evaluation," in *CVPR*, 2014.

[31] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *CVPR*, 2016.

[32] M. Oechsle, S. Peng, and A. Geiger, "Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction," in *ICCV*, 2021.

[33] J. Zhang, Y. Yao, and L. Quan, "Learning signed distance field for multi-view surface reconstruction," *International Conference on Computer Vision (ICCV)*, 2021.

[34] Y. Wang, I. Skorokhodov, and P. Wonka, "Hf-neus: Improved surface reconstruction using high-frequency details," 2022.

[35] Z. Yu, S. Peng, M. Niemeyer, T. Sattler, and A. Geiger, "Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction," *NeurIPS*, 2022.

[36] J. Zhang, Y. Yao, S. Li, T. Fang, D. McKinnon, Y. Tsin, and L. Quan, "Critical regularizations for neural surface reconstruction in the wild," in *CVPR*, 2022.

[37] Y. Wang, I. Skorokhodov, and P. Wonka, "Pet-neus: Positional encoding triplanes for neural surfaces," 2023.

[38] B. Miller, H. Chen, A. Lai, and I. Gkioulekas, "Objects as volumes: A stochastic geometry view of opaque solids," in *CVPR*, 2024.

[39] Y. Zhang, Z. Hu, H. Wu, M. Zhao, L. Li, Z. Zou, and C. Fan, "Towards unbiased volume rendering of neural implicit surfaces with geometry priors," in *CVPR*, 2023.

[40] Y. Xiao, J. Xu, Z. Yu, and S. Gao, "Debsdf: Delving into the details and bias of neural indoor scene reconstruction," *TPAMI*, 2024.

[41] F. Darmon, B. Bascle, J.-C. Devaux, P. Monasse, and M. Aubry, "Improving neural implicit surfaces geometry with patch warping," in *CVPR*, 2022.

[42] Q. Fu, Q. Xu, Y.-S. Ong, and W. Tao, "Geo-neus: Geometry-consistent neural implicit surfaces learning for multi-view reconstruction," *NeurIPS*, 2022.

[43] T. Lu, M. Yu, L. Xu, Y. Xiangli, L. Wang, D. Lin, and B. Dai, "Scaffold-gs: Structured 3d gaussians for view-adaptive rendering," in *CVPR*, 2024.

[44] Y. Liu, P. Wang, C. Lin, X. Long, J. Wang, L. Liu, T. Komura, and W. Wang, "Nero: Neural geometry and brdf reconstruction of reflective objects from multiview images," in *ACM Trans. Graph.*, 2023.

[45] 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," in *ACM SIGGRAPH*, 2023.

[46] J. Tang, H. Zhou, X. Chen, T. Hu, E. Ding, J. Wang, and G. Zeng, "Delicate textured mesh recovery from nerf via adaptive surface refinement," in *ICCV*, 2023.

[47] J. Gao, C. Gu, Y. Lin, H. Zhu, X. Cao, L. Zhang, and Y. Yao, "Relightable 3d gaussian: Real-time point cloud relighting with brdf decomposition and ray tracing," in *ECCV*, 2024.

[48] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," *ICCV*, 2021.

[49] Y. Wang, Q. Han, M. Habermann, K. Daniilidis, C. Theobalt, and L. Liu, "Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction," in *ICCV*, 2023.

[50] C. Xu and T. Takafumi, "Supernormal: Neural surface reconstruction via multi-view normal integration," in *CVPR*, 2024.