2DGS: 2D Gaussian Splatting for Geometrically Accurate Radiance Fields

SIGGRAPH 2024

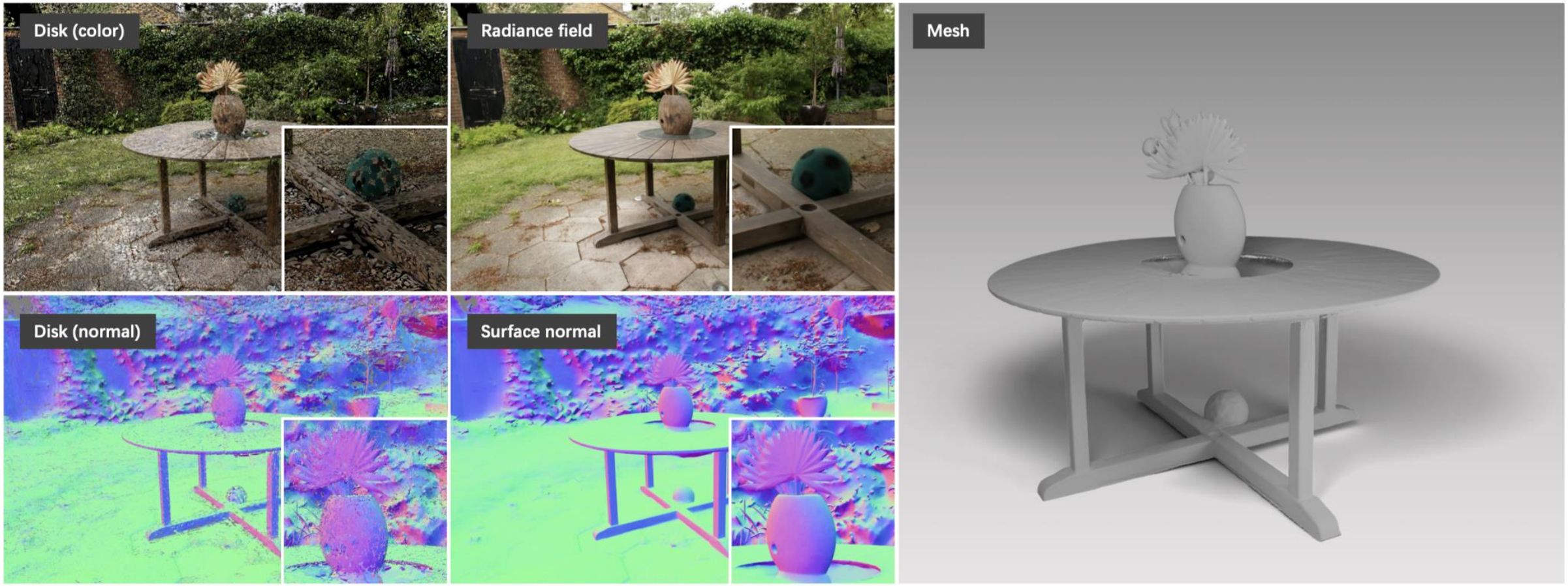
Binbin Huang[1] Zehao Yu[2,3] Anpei Chen[2,3] Andreas Geiger[2,3] Shenghua Gao[1]

[1]ShanghaiTech University [2]University of Tübingen [3]Tübingen AI Center

https://surfsplatting.github.io/

2DGS: a novel approach to model and reconstruct geometrically accurate radiance fields from multi-view images.

Key idea: collapse the 3D volume into a set of 2D oriented planar Gaussian disks.



(a) 2D disks as surface elements (b) 2D Gaussian splatting (c) Meshing

Fig. 1. Our method, *2DGS*, (a) optimizes a set of 2D oriented disks to represent and reconstruct a complex real-world scene from multi-view RGB images. These optimized 2D disks are tightly aligned to the surfaces. (b) With 2D Gaussian splatting, we allow real-time rendering of high quality novel view images with view consistent normals and depth maps. (c) Finally, our method provides detailed and noise-free triangle mesh reconstruction from the optimized 2D disks.

Contributions

In summary, we make the following contributions:

- We present a highly efficient differentiable 2D Gaussian renderer, enabling perspective-correct splatting by leveraging 2D surface modeling, ray-splat intersection, and volumetric integration.
- We introduce two regularization losses for improved and noise-free surface reconstruction.
- Our approach achieves state-of-the-art geometry reconstruction and NVS results compared to other explicit representations.

3DGS Quick Review

Mean Position

Opacity [0,1)

View-dependent color (spherical harmonics coefficients)

$$G_k = \{\mu_k, s_k, R_k, \alpha_k, c_k\}$$

Scale, rotation

Covariance matrix params $\Sigma = RSS^T R^T$

a 3D vector s for scaling
a quaternion q to represent rotation.
(normalized, a valid unit quaternion)

$$G(x) = e^{-\frac{1}{2}(x)^T \Sigma^{-1}(x)}$$

This Gaussian is multiplied by α in our blending process.

3 3D GAUSSIAN SPLATTING

Kerbl et al. [Kerbl et al. 2023] propose to represent 3D scenes with 3D Gaussian primitives and render images using differentiable volume splatting. Specifically, 3DGS explicitly parameterizes Gaussian primitives via 3D covariance matrix Σ and their location $\mathbf{p}_k$:

$$\mathcal{G}(\mathbf{p}) = \exp(-\frac{1}{2}(\mathbf{p}-\mathbf{p}_k)^\top \Sigma^{-1}(\mathbf{p}-\mathbf{p}_k)) \quad (1)$$

where the covariance matrix $\Sigma = \mathbf{RSS}^\top\mathbf{R}^\top$ is factorized into a scaling matrix $\mathbf{S}$ and a rotation matrix $\mathbf{R}$. To render an image, the 3D Gaussian is transformed into the camera coordinates with world-to-camera transform matrix $\mathbf{W}$ and projected to image plane via a local affine transformation $\mathbf{J}$ [Zwicker et al. 2001a]:

$$\Sigma' = \mathbf{JW}\Sigma\mathbf{W}^\top\mathbf{J}^\top \quad (2)$$

By skipping the third row and column of Σ', we obtain a 2D Gaussian $\mathcal{G}^{2D}$ with covariance matrix Σ^{2D}. Next, 3DGS [Kerbl et al. 2023] employs volumetric alpha blending to integrate alpha-weighted appearance from front to back:

$$\mathbf{c}(\mathbf{x}) = \sum_{k=1}^{K} \mathbf{c}_k \, \alpha_k \, \mathcal{G}_k^{2D}(\mathbf{x}) \prod_{j=1}^{k-1}(1-\alpha_j \, \mathcal{G}_j^{2D}(\mathbf{x})) \quad (3)$$

where k is the index of the Gaussian primitives, α_k denotes the alpha values and $\mathbf{c}_k$ is the view-dependent appearance. The attributes of 3D Gaussian primitives are optimized using a photometric loss.

3DGS Quick Review

Challenges in Surface Reconstruction. Reconstructing surfaces using 3D Gaussian modeling and splatting faces several challenges. First, the volumetric radiance representation of 3D Gaussians conflicts with the thin nature of surfaces. Second, 3DGS does not natively model surface normals, essential for high-quality surface reconstruction. Third, the rasterization process in 3DGS lacks multi-view consistency, leading to varied 2D intersection planes for different viewpoints [Keselman and Hebert 2023], as illustrated in Figure 2 (a). Additionally, using an affine matrix for transforming a 3D Gaussian into ray space only yields accurate projections near the center, compromising on perspective accuracy around surrounding regions [Zwicker et al. 2004]. Therefore, it often results in noisy reconstructions, as shown in Figure 5.

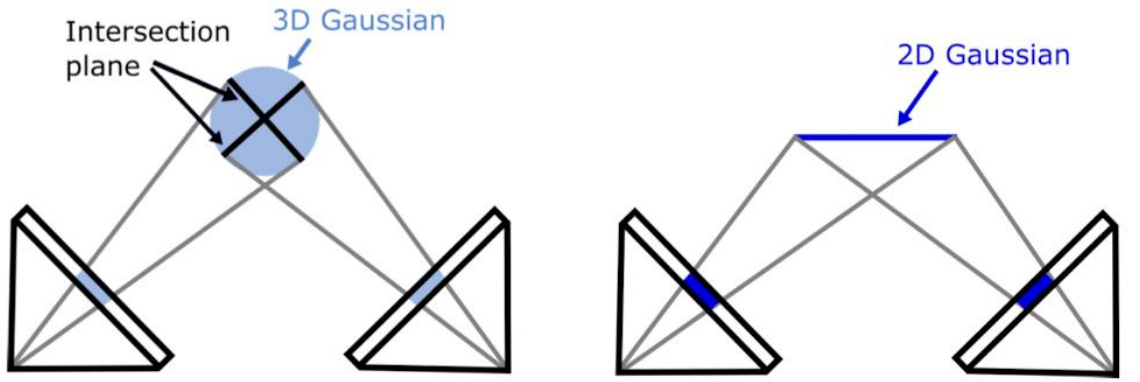


Fig. 2. Comparison of 3DGS and 2DGS. 3DGS utilizes different intersection planes for value evaluation when viewing from different viewpoints, resulting in inconsistency. Our 2DGS provides multi-view consistent value evaluations.

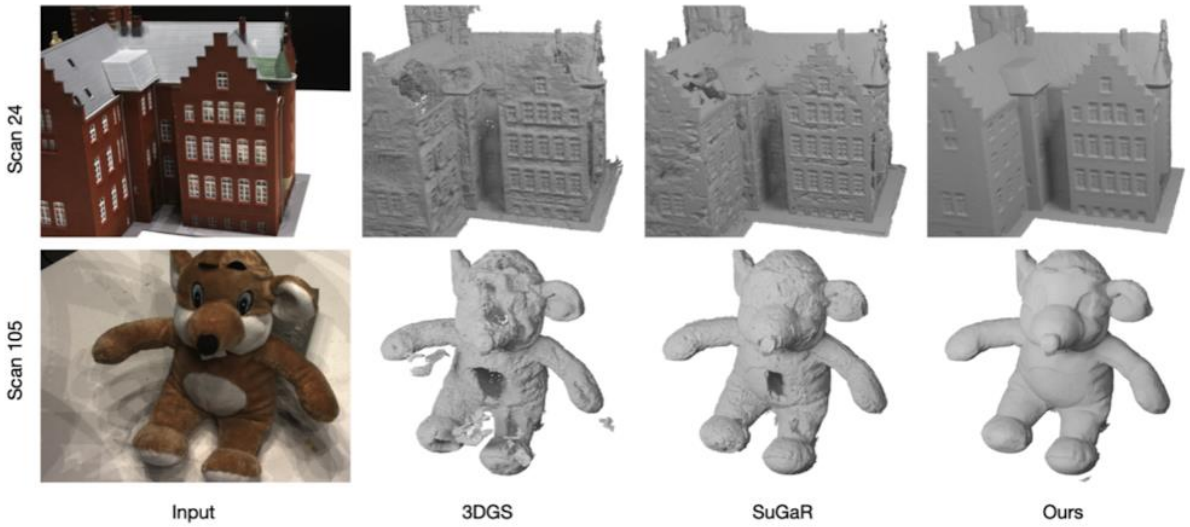


Fig. 5. Qualitative comparison on the DTU benchmark [Jensen et al. 2014]. Our 2DGS produces detailed and noise-free surfaces.

2DGS - Modeling

As illustrated in Figure 3, our 2D splat is characterized by its central point $\mathbf{p}_k$, two principal tangential vectors $\mathbf{t}_u$ and $\mathbf{t}_v$, and a scaling vector $\mathbf{S} = (s_u, s_v)$ that controls the variances of the 2D Gaussian. Notice that the primitive normal is defined by two orthogonal tangential vectors $\mathbf{t}_w = \mathbf{t}_u \times \mathbf{t}_v$. We can arrange the orientation into a 3×3 rotation matrix $\mathbf{R} = [\mathbf{t}_u, \mathbf{t}_v, \mathbf{t}_w]$ and the scaling factors into a 3×3 diagonal matrix $\mathbf{S}$ whose last entry is zero.

A 2D Gaussian is therefore defined in a local tangent plane in world space, which is parameterized:

$$P(u, v) = \mathbf{p}_k + s_u \mathbf{t}_u u + s_v \mathbf{t}_v v = \mathbf{H}(u, v, 1, 1)^{\mathrm{T}} \tag{4}$$

$$\text{where } \mathbf{H} = \begin{bmatrix} s_u \mathbf{t}_u & s_v \mathbf{t}_v & \mathbf{0} & \mathbf{p}_k \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{RS} & \mathbf{p}_k \\ \mathbf{0} & 1 \end{bmatrix} \tag{5}$$

where $\mathbf{H} \in 4 \times 4$ is a homogeneous transformation matrix representing the geometry of the 2D Gaussian. For the point $\mathbf{u} = (u, v)$ in uv space, its 2D Gaussian value can then be evaluated by standard Gaussian

$$\mathcal{G}(\mathbf{u}) = \exp\left(-\frac{u^2 + v^2}{2}\right) \tag{6}$$

The center $\mathbf{p}_k$, scaling (s_u, s_v), and the rotation $(\mathbf{t}_u, \mathbf{t}_v)$ are learnable parameters. Following 3DGS [Kerbl et al. 2023], each 2D Gaussian primitive has opacity α and view-dependent appearance c parameterized with spherical harmonics.

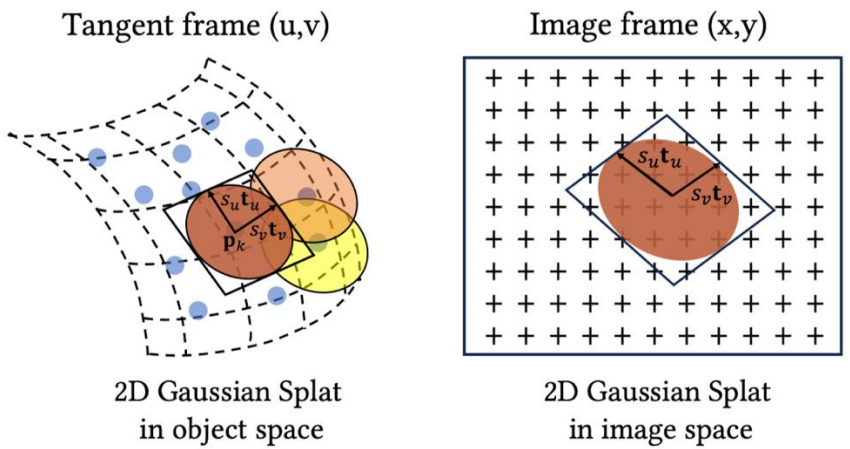
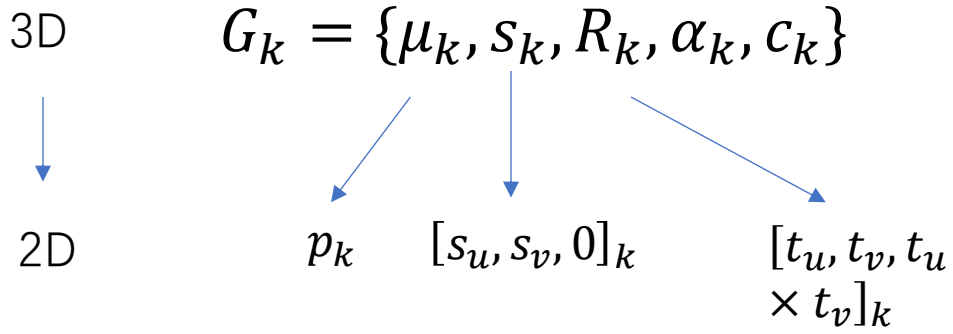


Fig. 3. Illustration of 2D Gaussian Splatting. 2D Gaussian Splats are elliptical disks characterized by a center point $\mathbf{p}_k$, tangential vectors $\mathbf{t}_u$ and $\mathbf{t}_v$, and two scaling factors (s_u and s_v) control the variance. Their elliptical projections are sampled through the ray-splat intersection (Section 4.2) and accumulated via alpha-blending in image space. 2DGS reconstructs surface attributes such as colors, depths, and normals through gradient descent.



ing "flat" 2D Gaussians embedded in 3D space. With 2D Gaussian modeling, the primitive distributes densities within a planar disk, defining the normal as the direction of the steepest change of density. This feature enables better alignment with thin surfaces. While

2DGS - Splatting

coordinates. Specifically, projecting the 2D splat onto an image plane can be described by a general 2D-to-2D mapping in homogeneous coordinates. Let $\mathbf{W} \in 4 \times 4$ be the transformation matrix from world space to screen space. The screen space points are hence obtained by

$$\mathbf{x} = (xz, yz, z, 1)^{\mathrm{T}} = \mathbf{W}P(u, v) = \mathbf{WH}(u, v, 1, 1)^{\mathrm{T}} \tag{7}$$

where $\mathbf{x}$ represents a homogeneous ray emitted from the camera and passing through pixel (x, y) and intersecting the splat at depth z. To rasterize a 2D Gaussian, Zwicker et al. proposed to project its conic into the screen space with an implicit method using $\mathbf{M} = (\mathbf{WH})^{-1}$. However, the inverse transformation introduces numerical instability especially when the splat degenerates into a line segment (i.e., if it is viewed from the side). To address this issue, previous surface splatting rendering methods discard such ill-conditioned transformations using a pre-defined threshold. However, such a scheme poses challenges within a differentiable rendering framework, as thresholding can lead to unstable optimization. To address this problem, we utilize an explicit ray-splat intersection inspired by [Sigg et al. 2006].

Ray-splat Intersection. We efficiently locate the ray-splat intersections by finding the intersection of three non-parallel planes, a method initially designed for specialized hardware [Weyrich et al. 2007]. Given an image coordinate $\mathbf{x} = (x, y)$, we parameterize the ray of a pixel as the intersection of two orthogonal planes: the x-plane and the y-plane. Specifically, the x-plane is defined by a normal vector $(-1, 0, 0)$ and an offset x. Therefore, the x-plane can be represented as a 4D homogeneous plane $\mathbf{h}_x = (-1, 0, 0, x)^{\mathrm{T}}$. Similarly, the y-plane is $\mathbf{h}_y = (0, -1, 0, y)^{\mathrm{T}}$. Thus, the ray $\mathbf{x} = (x, y)$ is determined by the intersection of the x-plane and the y-planes.

Next, we transform both planes into the local coordinates of the 2D Gaussian primitives, the uv-coordinate system. Note that transforming points on a plane using a transformation matrix $\mathbf{M}$ is equivalent to transforming the homogeneous plane parameters using the inverse transpose $\mathbf{M}^{-\mathrm{T}}$. Therefore, applying $\mathbf{M} = (\mathbf{WH})^{-1}$ is equivalent to $(\mathbf{WH})^{\mathrm{T}}$, eliminating explicit matrix inversion and yielding:

$$\mathbf{h}_u = (\mathbf{WH})^{\mathrm{T}}\mathbf{h}_x \quad \mathbf{h}_v = (\mathbf{WH})^{\mathrm{T}}\mathbf{h}_y \tag{8}$$

As introduced in Section 4.1, points on the 2D Gaussian plane are represented as $(u, v, 1, 1)$. At the same time, the intersection point should fall on the transformed x-plane and y-plane. Thus,

$$\mathbf{h}_u \cdot (u, v, 1, 1)^{\mathrm{T}} = \mathbf{h}_v \cdot (u, v, 1, 1)^{\mathrm{T}} = 0 \tag{9}$$

This leads to an efficient solution for the intersection point $\mathbf{u}(\mathbf{x})$:

$$u(\mathbf{x}) = \frac{\mathbf{h}_u^2\mathbf{h}_v^4 - \mathbf{h}_u^4\mathbf{h}_v^2}{\mathbf{h}_u^1\mathbf{h}_v^2 - \mathbf{h}_u^2\mathbf{h}_v^1} \qquad v(\mathbf{x}) = \frac{\mathbf{h}_u^4\mathbf{h}_v^1 - \mathbf{h}_u^1\mathbf{h}_v^4}{\mathbf{h}_u^1\mathbf{h}_v^2 - \mathbf{h}_u^2\mathbf{h}_v^1} \tag{10}$$

where $\mathbf{h}_u^i, \mathbf{h}_v^i$ are the i-th parameter of the 4D homogeneous plane parameters. We obtain the depth z of the intersected points via Eq. 7 and evaluate the Gaussian value with Eq 6.

2DGS - Splatting

Degenerate Solutions. When a 2D Gaussian is observed from a slanted viewpoint, it degenerates to a line in screen space. Therefore, it might be missed during rasterization. To deal with these cases and stabilize optimization, we employ the object-space low-pass filter introduced in [Botsch et al. 2005]:

$$\hat{\mathcal{G}}(\mathbf{x}) = \max\left\{\mathcal{G}(\mathbf{u}(\mathbf{x})), \mathcal{G}(\frac{\mathbf{x}-\mathbf{c}}{\sigma})\right\} \tag{11}$$

where $\mathbf{u}(\mathbf{x})$ is given by (10) and $\mathbf{c}$ is the projection of center $\mathbf{p}_k$. Intuitively, $\hat{\mathcal{G}}(\mathbf{x})$ is lower-bounded by a fixed screen-space Gaussian low-pass filter with center $\mathbf{c}$ and radius σ. In our experiments, we set $\sigma = \sqrt{2}/2$ to ensure sufficient pixels are used during rendering.

Rasterization. We follow a similar rasterization process as in 3DGS [Kerbl et al. 2023]. First, a screen space bounding box is computed for each Gaussian primitive. Then, 2D Gaussians are sorted based on the depth of their center and organized into tiles based on their bounding boxes. Finally, volumetric alpha blending is used to integrate alpha-weighted appearance from front to back:

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1} \mathbf{c}_i \, \alpha_i \, \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} (1 - \alpha_j \, \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x}))) \tag{12}$$

The iterative process is terminated when the accumulated opacity reaches saturation.

2DGS - Regularization

Depth Distortion. Different from NeRF, 3DGS's volume rendering doesn't consider the distance between intersected Gaussian primitives. Therefore, spreading out Gaussians might result in a similar color and depth rendering. This is different from surface rendering, where rays intersect the first visible surface exactly once. To mitigate this issue, we take inspiration from Mip-NeRF360 [Barron et al. 2022a] and propose a depth distortion loss to concentrate the weight distribution along the rays by minimizing the distance between the ray-splat intersections:

$$\mathcal{L}_d = \sum_{i,j} \omega_i \omega_j |z_i - z_j| \tag{13}$$

where $\omega_i = \alpha_i \hat{\mathcal{G}}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1}(1 - \alpha_j \hat{\mathcal{G}}_j(\mathbf{u}(\mathbf{x})))$ is the blending weight of the $i-$th intersection and z_i is the depth of the intersection points. Unlike the distortion loss in Mip-NeRF360, where z_i is the distance between sampled points and is not optimized, our approach directly encourages the concentration of the splats by adjusting the intersection depth z_i. Note that we implement this regularization term efficiently with CUDA in a manner similar to [Sun et al. 2022b].

Normal Consistency. As our representation is based on 2D Gaussian surface elements, we must ensure that all 2D splats are locally aligned with the actual surfaces. In the context of volume rendering where multiple semi-transparent surfels may exist along the ray, we consider the actual surface at the median point of intersection $\mathbf{p}_s$, where the accumulated opacity reaches 0.5. We then align the splats' normal with the gradients of the depth maps as follows:

$$\mathcal{L}_n = \sum_i \omega_i (1 - \mathbf{n}_i^{\mathrm{T}} \mathbf{N}) \tag{14}$$

where i indexes over intersected splats along the ray, ω denotes the blending weight of the intersection point, $\mathbf{n}_i$ represents the normal of the splat that is oriented towards the camera, and $\mathbf{N}$ is the normal estimated by the gradient of the depth map. Specifically, $\mathbf{N}$ is computed with finite differences from nearby depth points as follows:

$$\mathbf{N}(x, y) = \frac{\nabla_x \mathbf{p}_s \times \nabla_y \mathbf{p}_s}{|\nabla_x \mathbf{p}_s \times \nabla_y \mathbf{p}_s|} \tag{15}$$

By aligning the splat normal with the estimated surface normal, we ensure that 2D splats locally approximate the actual object surface.

2DGS - Regularization

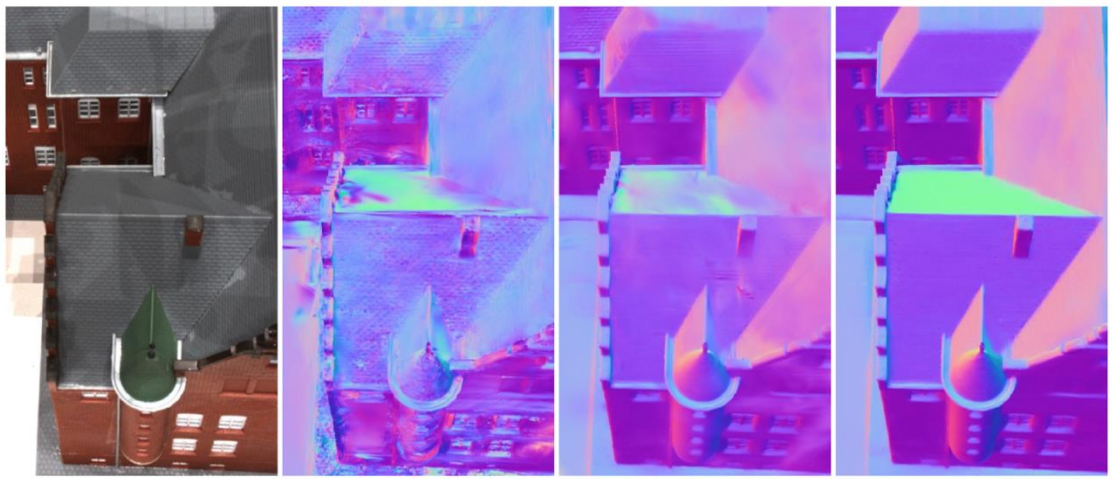
Final Loss. Finally, we optimize our model from an initial sparse point cloud using a set of posed images. We minimize the following loss function:

$$\mathcal{L} = \mathcal{L}_c + \alpha \mathcal{L}_d + \beta \mathcal{L}_n \tag{16}$$

where $\mathcal{L}c$ is an RGB reconstruction loss combining $\mathcal{L}_1$ with the D-SSIM term from [Kerbl et al. 2023], while $\mathcal{L}_d$ and $\mathcal{L}_n$ are regularization terms. We set $\alpha = 1000$ for bounded scenes, $\alpha = 100$ for unbounded scenes, and $\beta = 0.05$ for all scenes.

Table 5. Quantitative studies for the regularization terms and mesh extraction methods on the DTU dataset.

	Accuracy ↓	Completion ↓	Average ↓
A. w/o normal consistency	1.35	1.13	1.24
B. w/o depth distortion	0.89	0.87	0.88
C. w / expected depth	0.88	1.01	0.94
D. w / SPSR	1.25	0.89	1.07
E. Full Model	0.79	0.86	0.83



Input (A) w/o. NC (B) w/o. DD Full Model

Fig. 6. Qualitative studies for the regularization effects. From left to right – input image, surface normals without normal consistency, without depth distortion, and our full model. Disabling the normal consistency loss leads to noisy surface orientations; conversely, omitting depth distortion regularization results in blurred surface normals. The complete model, employing both regularizations, successfully captures sharp and flat features.

2DGS – Mesh Extraction

Mesh Extraction. To extract meshes from reconstructed 2D splats, we render depth maps of the training views using the depth value of the splats projected to the pixels and utilize truncated signed distance fusion (TSDF) to fuse the reconstruction depth maps, using Open3D [Zhou et al. 2018]. We set the voxel size to 0.004 and the truncated threshold to 0.02 during TSDF fusion. We also extend the original 3DGS to render depth and employ the same technique for surface reconstruction for a fair comparison.

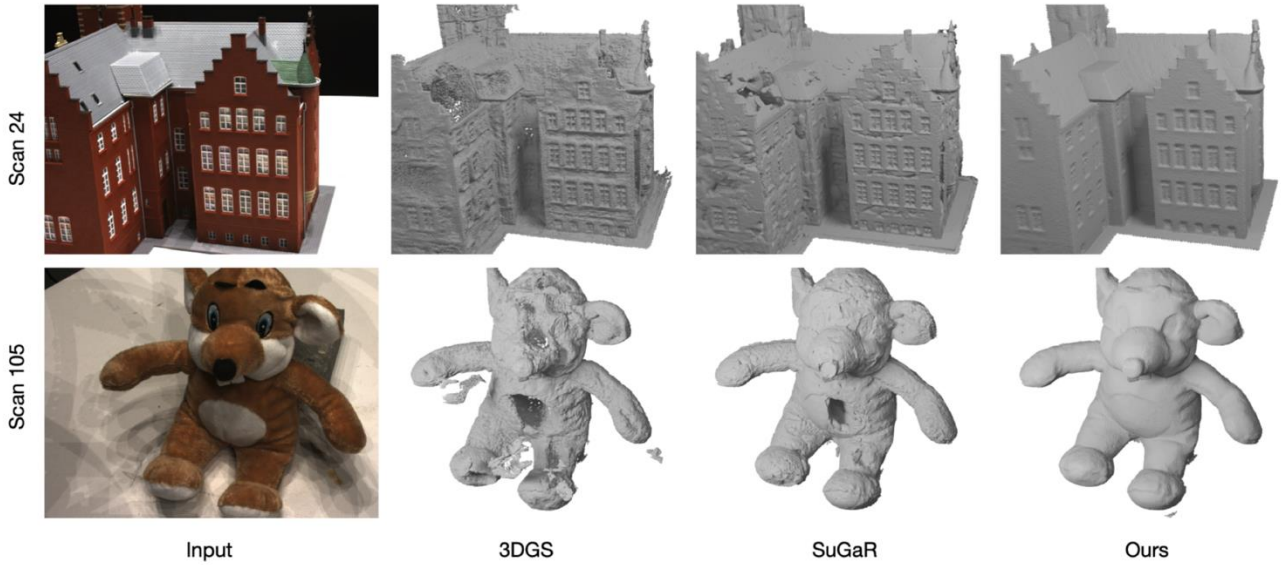


Fig. 5. Qualitative comparison on the DTU benchmark [Jensen et al. 2014]. Our 2DGS produces detailed and noise-free surfaces.

Experiments

Table 1. Quantitative comparison on the DTU Dataset [Jensen et al. 2014]. Our 2DGS achieves the highest reconstruction accuracy among other methods and provides 100× speed up compared to the SDF based baselines.

		24	37	40	55	63	65	69	83	97	105	106	110	114	118	122	Mean	Time
implicit	NeRF [Mildenhall et al. 2021]	1.90	1.60	1.85	0.58	2.28	1.27	1.47	1.67	2.05	1.07	0.88	2.53	1.06	1.15	0.96	1.49	> 12h
implicit	VolSDF [Yariv et al. 2021]	1.14	1.26	0.81	0.49	1.25	0.70	0.72	1.29	1.18	0.70	0.66	1.08	0.42	0.61	0.55	0.86	>12h
implicit	NeuS [Wang et al. 2021]	1.00	1.37	0.93	0.43	1.10	0.65	0.57	1.48	1.09	0.83	0.52	1.20	0.35	0.49	0.54	0.84	>12h
explicit	3DGS [Kerbl et al. 2023]	2.14	1.53	2.08	1.68	3.49	2.21	1.43	2.07	2.22	1.75	1.79	2.55	1.53	1.52	1.50	1.96	11.2 m
explicit	SuGaR [Guédon and Lepetit 2023]	1.47	1.33	1.13	0.61	2.25	1.71	1.15	1.63	1.62	1.07	0.79	2.45	0.98	0.88	0.79	1.33	~ 1h
explicit	2DGS-15k (Ours)	0.48	0.92	0.42	0.40	1.04	0.83	0.83	1.36	1.27	0.76	0.72	1.63	0.40	0.76	0.60	0.83	5.5 m
explicit	2DGS-30k (Ours)	0.48	0.91	0.39	0.39	1.01	0.83	0.81	1.36	1.27	0.76	0.70	1.40	0.40	0.76	0.52	0.80	10.9 m

Table 2. Quantitative results on the Tanks and Temples Dataset [Knapitsch et al. 2017]. We report the F1 score and training time.

	NeuS	Geo-Neus	Neurlangelo	SuGaR	3DGS	Ours
Barn	0.29	0.33	0.70	0.14	0.13	0.41
Caterpillar	0.29	0.26	0.36	0.16	0.08	0.23
Courthouse	0.17	0.12	0.28	0.08	0.09	0.16
Ignatius	0.83	0.72	0.89	0.33	0.04	0.51
Meetingroom	0.24	0.20	0.32	0.15	0.01	0.17
Truck	0.45	0.45	0.48	0.26	0.19	0.45
Mean	0.38	0.35	0.50	0.19	0.09	0.32
Time	>24h	>24h	>24h	>1h	14.3 m	15.5 m

Table 3. Performance comparison between 2DGS (ours), 3DGS and SuGaR on the DTU dataset [Jensen et al. 2014]. We report the averaged chamfer distance, PSNR (training-set view), reconstruction time, and model size.

	CD ↓	PSNR ↑	Time ↓	MB (Storage) ↓
3DGS [Kerbl et al. 2023]	1.96	**35.76**	11.2 m	113
SuGaR [Guédon and Lepetit 2023]	1.33	34.57	~1 h	1247
2DGS-15k (Ours)	0.83	33.42	**5.5 m**	**52**
2DGS-30k (Ours)	**0.80**	34.52	10.9 m	**52**

Experiments

Table 4. Quantitative results on Mip-NeRF 360 [Barron et al. 2022a] dataset. All scores of the baseline methods are directly taken from their papers whenever available. We report the performance of 3DGS, SuGaR and ours using 30k iterations.

	Outdoor Scene			Indoor scene		
	PSNR ↑	SSIM ↑	LIPPS ↓	PSNR ↑	SSIM ↑	LIPPS ↓
NeRF	21.46	0.458	0.515	26.84	0.790	0.370
Deep Blending	21.54	0.524	0.364	26.40	0.844	0.261
Instant NGP	22.90	0.566	0.371	29.15	0.880	0.216
MERF	23.19	0.616	0.343	27.80	0.855	0.271
BakedSDF	22.47	0.585	0.349	27.06	0.836	0.258
MipNeRF360	24.47	0.691	0.283	31.72	0.917	0.180
Mobile-NeRF	21.95	0.470	0.470	-	-	-
SuGaR	22.93	0.629	0.356	29.43	0.906	0.225
3DGS	24.64	0.731	0.234	30.41	0.920	0.189
2DGS (Ours)	24.34	0.717	0.246	30.40	0.916	0.195

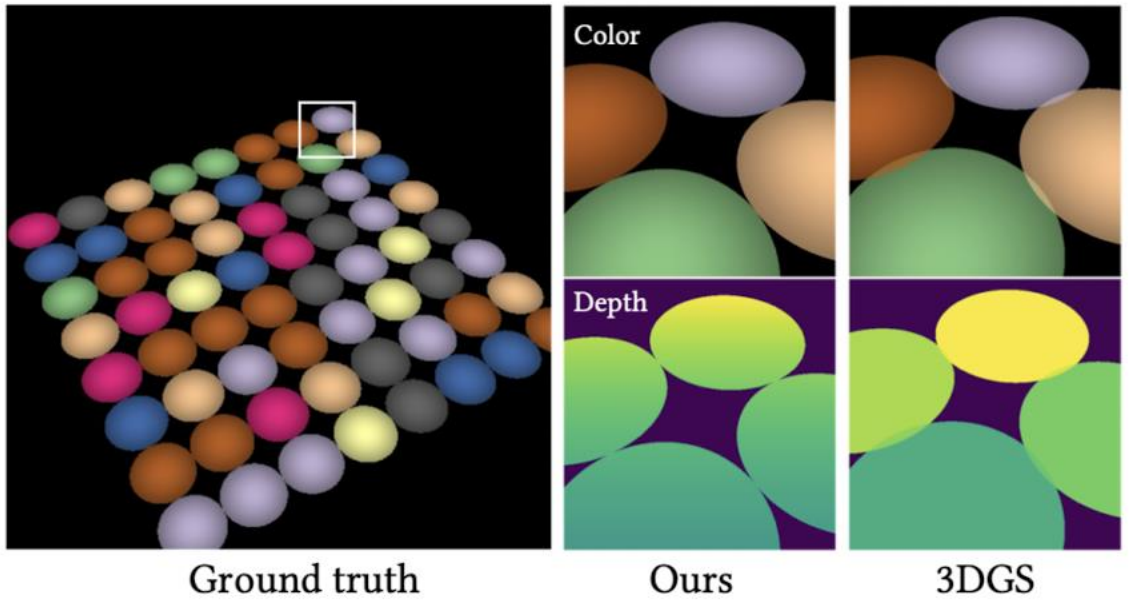


Fig. 7. Visualization of a plane tiled by 2D Gaussians. Affine approximation [Zwicker et al. 2001b] adopted in 3DGS [Kerbl et al. 2023] causes perspective distortion and inaccurate depth, violating normal consistency.

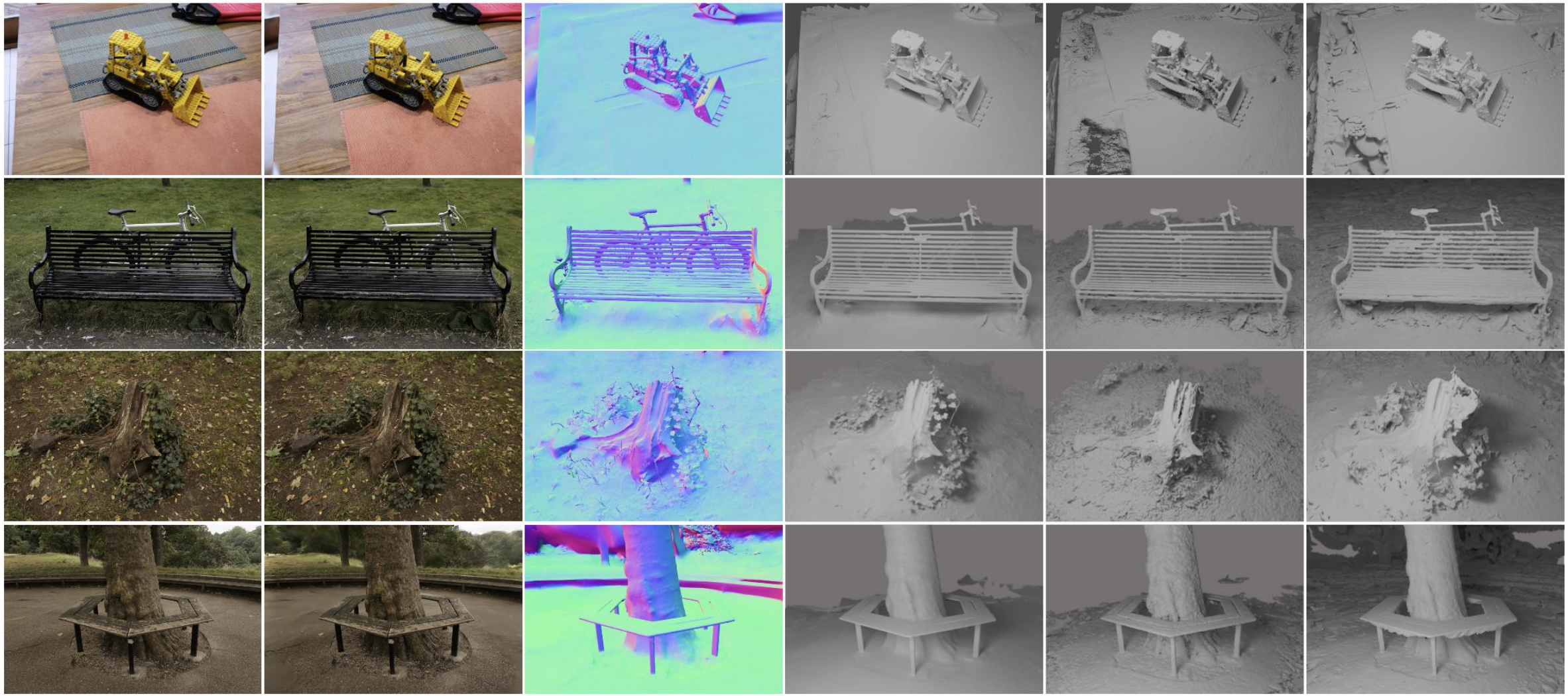
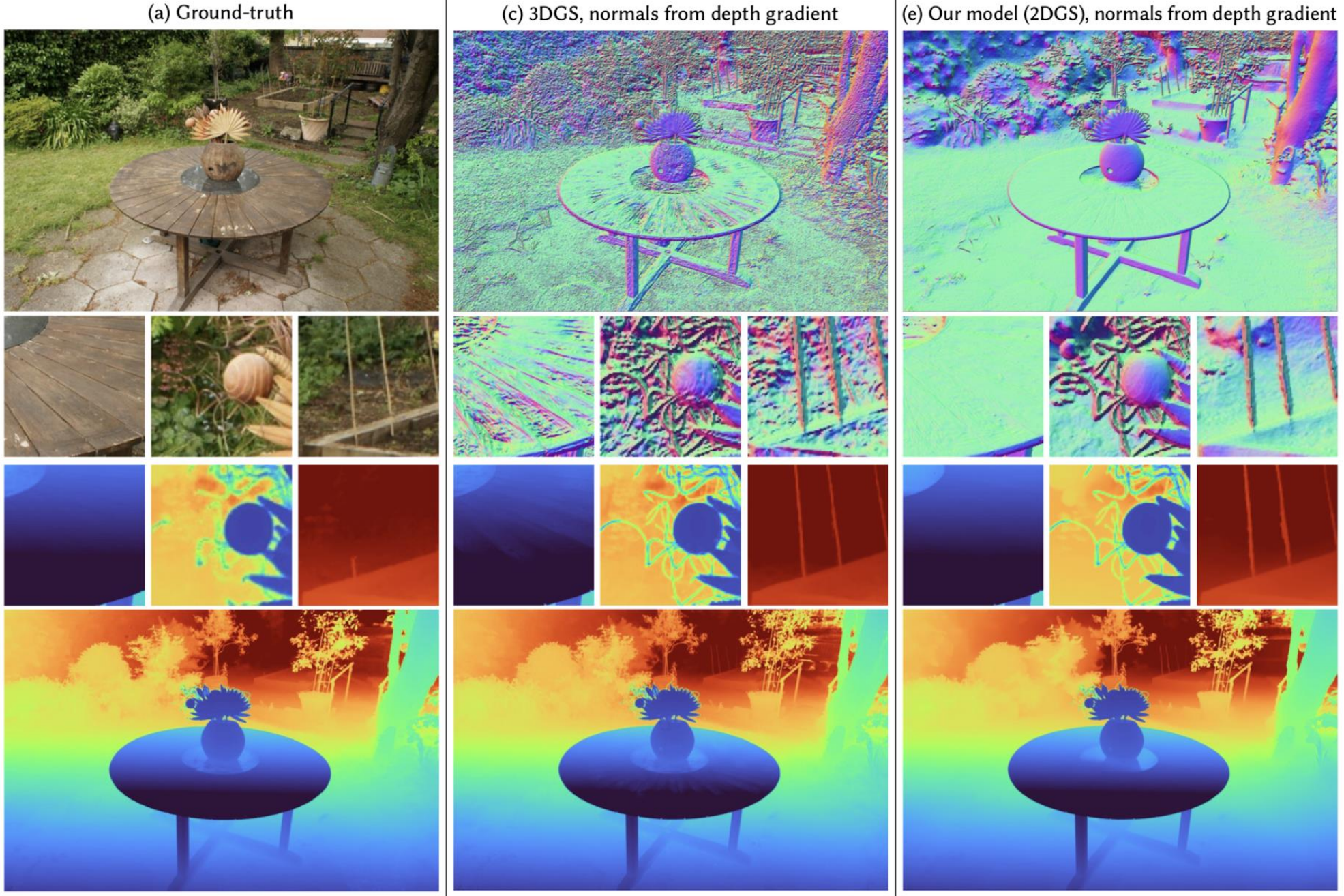


Fig. 4. Visual comparisons (test-set view) between our method, 3DGS [Kerbl et al. 2023], and SuGaR [Guédon and Lepetit 2023] using scenes from an real-world dataset [Barron et al. 2022b]. Our method excels at synthesizing geometrically accurate radiance fields and surface reconstruction, outperforming 3DGS and SuGaR in capturing sharp edges and intricate details.

Exp


(a) Ground-truth
(c) 3DGS, normals from depth gradient
(e) Our model (2DGS), normals from depth gradient
(b) MipNeRF360 [Barron et al. 2022b], SSIM=0.813
(d) 3DGS [Kerbl et al. 2023], SSIM=0.834
(f) Our model (2DGS), SSIM=0.845

Experim


scan65
scan69
scan83
scan97
scan105
scan106
scan110
scan114
scan118
scan122
2DGS
3DGS
scan24
scan37
scan40
scan55
scan63
scan65
scan69
scan83
scan97
scan105