RaDe-GS: Rasterizing Depth in Gaussian Splatting

BAOWEN ZHANG, Hong Kong University of Science and Technology, Hong Kong
CHUAN FANG, Hong Kong University of Science and Technology, Hong Kong
RAKESH SHRESTHA, Simon Fraser University, Canada
YIXUN LIANG, Hong Kong University of Science and Technology, Hong Kong
XIAOXIAO LONG, Hong Kong University of Science and Technology, Hong Kong
PING TAN, Hong Kong University of Science and Technology, Hong Kong and Simon Fraser University, Canada
https://baowenz.github.io/radegs

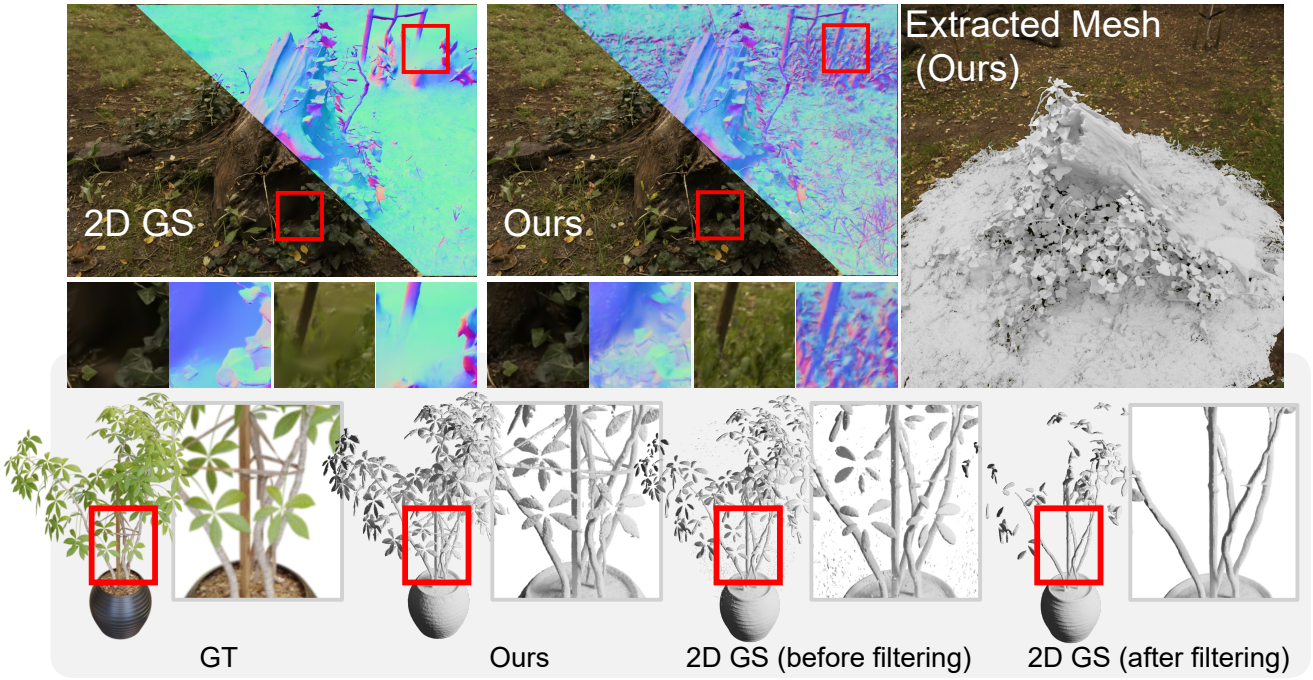

Fig. 1. We present a rasterized method to compute the depth and surface normal maps of general Gaussian splats. Our method achieves high-quality 3D shape reconstruction and maintains excellent training and rendering efficiency. In comparison, forcing Gaussian splats to be planar as in 2D GS [Huang et al. 2024] produces blurry novel view rendering and noisy 3D shape. 2D GS by default filters the noise after mesh extraction.

Gaussian Splatting (GS) has proven to be highly effective in novel view synthesis, achieving high-quality and real-time rendering. However, its potential for reconstructing detailed 3D shapes has not been fully explored. Existing methods often suffer from limited shape accuracy due to the discrete and unstructured nature of Gaussian splats, which complicates the shape extraction. While recent techniques like 2D GS have attempted to improve shape reconstruction, they often reformulate the Gaussian primitives in ways that reduce both rendering quality and computational efficiency. To address these problems, our work introduces a rasterized approach to render the depth maps and surface normal maps of general 3D Gaussian splats. Our method not only significantly enhances shape reconstruction accuracy but also maintains the computational efficiency intrinsic to Gaussian Splatting. It achieves a Chamfer distance error comparable to NeuraLangelo[Li et al. 2023] on the DTU dataset, taking only 5 minutes. Our method is a significant advancement in Gaussian Splatting and can be directly integrated into existing Gaussian Splatting-based methods.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; **Rendering**; *Machine learning approaches*.

Additional Key Words and Phrases: 3D gaussian, surface reconstruction, depth planarity

Authors' addresses: Baowen Zhang, Hong Kong University of Science and Technology, Hong Kong, Hong Kong, bzhangcm@connect.ust.hk; Chuan Fang, Hong Kong University of Science and Technology, Hong Kong, Hong Kong, cfangac@connect.ust.hk; Rakesh Shrestha, Simon Fraser University, Burnaby, Canada, rakeshs@sfu.ca; Yixun Liang, Hong Kong University of Science and Technology, Hong Kong, Hong Kong, lyxun2000@gmail.com; Xiaoxiao Long, Hong Kong University of Science and Technology, Hong Kong, Hong Kong, xxlong@connect.hku.hk; Ping Tan, Hong Kong University of Science and Technology, Hong Kong, Hong Kong and Simon Fraser University, Burnaby, Canada, pingtan@ust.hk.

1 INTRODUCTION

3D reconstruction from multi-view images is a classic problem with numerous applications in computer vision and graphics. This task typically involves generating depth maps through the multi-view stereo algorithms that utilize sophisticated optimization methods [Bleyer et al. 2011; Kolmogorov and Zabih 2001; Sun et al. 2003] or pretrained neural networks [Gu et al. 2019; Yao et al. 2018]. The

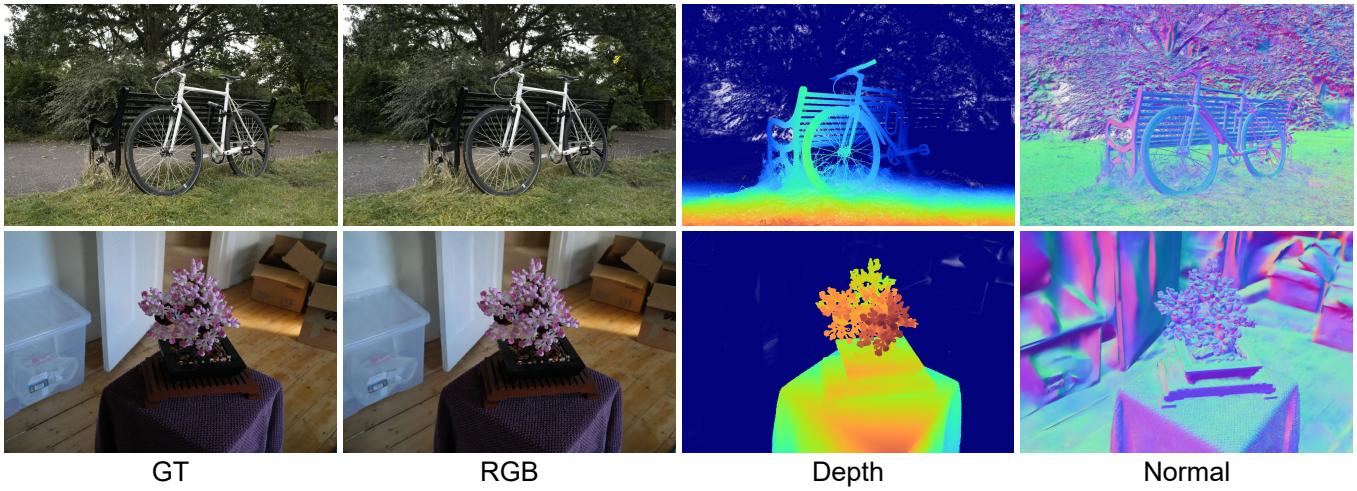

Fig. 2. Quality results on Mip-NeRF 360 dataset. From left to right, the images are ground truth color images, rendered color images, rendered depth maps, and rendered normal maps, respectively.

depth maps estimated from different viewpoints can then be integrated [Newcombe et al. 2011] to create a complete triangle mesh model. Although this traditional image-based modeling approach [Fuhrmann et al. 2014] delivers accurate results, it has limited robustness, particularly on reflective and shiny surfaces.

Neural Radiance Field (NeRF) [Mildenhall et al. 2021] employs an implicit representation of 3D scenes and achieves photorealist results for novel-view rendering through an analysis-by-synthesis approach. Despite its success, the original NeRF method tends to produce biased depth maps and noisy 3D reconstruction. NeuS [Wang et al. 2021] incorporates a Signed Distance Function (SDF) into the NeRF framework, significantly improving the accuracy of depth and shape reconstructions. These improvements in 3D reconstruction accuracy are further enhanced by hierarchical volumetric features in NeuraLangelo [Li et al. 2023]. However, the implicit representation of 3D scenes requires ray tracing to render images, which makes the training of NeRF computationally inefficient. Consequently, these methods [Li et al. 2023; Mildenhall et al. 2021; Wang et al. 2021] often take several hours to optimize a 3D model from a set of input images.

Gaussian Splatting (GS) [Kerbl et al. 2023] introduces an explicit representation for more efficient optimization and rendering. It represents a 3D scene using a set of translucent Gaussian spheres, which can be rendered efficiently by rasterization and can reconstruct a 3D scene in minutes. However, this representation complicates the computation of SDFs, as analyzed in [Guédon and Lepetit 2023]. Consequently, it is hard to extract 3D surfaces from GSs, which are necessary for applications like simulation and obstacle detection. Some recent works [Guédon and Lepetit 2023; Huang et al. 2024] attempt to make the Gaussian spheres planar to facilitate surface extraction. However, this lower dimensional representation leads to optimization challenges, especially for complicated shapes, as shown in the grasses and stump in Figure 1. In general, 2D GS methods [Guédon and Lepetit 2023; Huang et al. 2024] decrease the Peak Signal-to-Noise Ratio (PSNR), as evidenced in Table 3 and Table 4.

To address these challenges, GOF [Yu et al. 2024c] introduces a ray-tracing based method to compute the opacity along light rays to extract high-quality surfaces. While this approach generates excellent surface reconstruction, the ray-tracing process costs significant computational overhead. For example, GOF requires about one hour to optimize a scene from the DTU dataset [Jensen et al. 2014], while the standard GS method takes only about 5.2 minutes, as shown in Table 1.

We develop a rasterized method to compute depth maps for general Gaussian splats. Our method enjoys similar computation efficiency as the standard GS [Kerbl et al. 2023] thanks to the rasterized approach in computing depth and normal maps. On the DTU dataset, our method achieves a shape reconstruction accuracy of 0.69 mm Chamfer Distance after 5-minute training. This performance matches that of the implicit method NeuraLangelo, which has an accuracy of 0.61 mm, and exceeds that of recent GS-based methods like GOF (0.74 mm) and 2D GS (0.80 mm).

We have derived a closed-form solution for the intersections of light rays and Gaussian splats. Specifically, we evaluate the Gaussian values along each light ray from the camera center. The intersecting point on each ray is identified as the point where the Gaussian values are maximized. These intersection points between a Gaussian splat and a bundle of light rays lie on a general curved surface, which defines the projected depth of a Gaussian splat on the image plane. We discovered that these intersection points are co-planar under the approximate affine projection [Zwicker et al. 2002]. As a result, the projected depth of each Gaussian splat can be computed efficiently by rasterization according to our derived planar equation. The final depth map is computed as the median depth among the projected Gaussian splats, taking into account their translucency. Similarly, the surface normal map is derived through rasterized computation. This approach allows our method to produce accurate depth and normal maps, while maintaining the rendering and optimization efficiency of Gaussian splitting. As shown in Figure. 2, our method can render high quality color, depth, and normal maps from Gaussian splats.

In summary, the main contribution of this work is a novel rasterized method for computing depth and normal maps tailored to general Gaussian splats. Extensive experimental evaluation demonstrates that our method achieves high-quality 3D reconstruction, comparable to those of the recent implicit method NeuraLangelo[Li et al. 2023]. Additionaly, it maintains rendering and optimization efficiency on par with the original 3D Gaussian Splats [Kerbl et al. 2023].

2 RELATED WORK

2.1 Stereo and Multi-view Stereo

Stereo and multi-view stereo depth estimation are classical problems in computer vision. Traditional methods employ sophisticated optimization techniques, such as belief-propagation [Sun et al. 2003], graph-cut [Kolmogorov and Zabin 2004], and PatchMatch [Barnes et al. 2009; Bleyer et al. 2011] to establish pixel correspondences. While these methods have achieved accurate 3D reconstructions, they often struggle with textureless regions. More recent works [Kendall et al. 2017; Luo et al. 2016; Zbontar and LeCun 2015] utilize neural networks for end-to-end depth map estimation. MVSNet [Yao et al. 2018] integrates learning-based cost-volume filtering in the classical plane-sweeping stereo algorithm [Collins 1996]. Furthermore, CasMVSNet [Gu et al. 2020] designs a cascade cost-volume filtering to improve depth accuracy and reduce GPU memory usage. TransMVSNet [Ding et al. 2022] applies the attention mechanism in Transformers to improve feature correlation. Stereo and multi-view stereo methods recover a depth map for each input image, which can be fused into a complete 3D model with truncated signed distance functions (TSDF) [Curless and Levoy 1996a; Newcombe et al. 2011; Zach et al. 2011].

2.2 Nerual Radiance Field

Unlike multi-view stereo algorithms that rely on depth map representation, NeRF [Mildenhall et al. 2021] parameterizes a scene with an MLP network that maps spatial coordinates to color and translucency values. NeRF achieves photo-realistic novel viewpoint rendering by optimizing network parameters through an analysis-by-synthesis approach. Subsequent developments have led to significant improvements. For anti-aliasing rendering, MipNeRF [Barron et al. 2021] uses a conical frustum to represent the target scene at varying scales, and MipNeRF360 [Barron et al. 2022] further extends this method to unbounded scenes. Although these methods achieve faithful rendering, they require extensive training hours. Efforts to accelerate training include Plenoxels [Fridovich-Keil et al. 2022], which uses a 3D grid of spherical harmonics, and Instant-NGP [Müller et al. 2022], which combines a hashed feature volume and a shallow MLP. Additional methods focus on improving rendering efficiency at the inference stage through techniques like baking [Hedman et al. 2021] or distillation methods [Chen et al. 2023a; Reiser et al. 2023; Yu et al. 2021].

Another important direction in NeRF research aims to improve the accuracy of reconstructed 3D shapes. Notable works[Li et al. 2023; Wang et al. 2022, 2021; Yariv et al. 2021; Yu et al. 2022] have integrated a Signed Distance Function (SDF) with the radiance field to create high-fidelity 3D models. NeuS [Wang et al. 2021] establishes the connection between translucency and SDF values to ensure unbiased surface reconstruction. Moreover, Neuralangelo [Li et al. 2023] incorporates multi-resolution feature grids with SDF, achieving high-quality reconstruction on large-scale scenes. Despite these improvements, NeRF-based methods typically require expensive optimization with substantial GPU hours and memory. For instance, Neuralangelo [Li et al. 2023] requires about 128 GPU hours to reconstruct a single scene in the Tanks & Temples dataset [Knapitsch et al. 2017].

2.3 Gaussian Splatting

Gaussian Splatting [Kerbl et al. 2023] employs a set of 3D Gaussian primitives to represent a 3D scene. Combined with rasterized rendering, it and many variant works [Cheng et al. 2024; Fan et al. 2023; Lin et al. 2024; Wang et al. 2024] achieve real-time rendering and fast optimization. Building on this success, Mip-Splatting [Yu et al. 2024a] incorporates low-pass filters to address aliasing problems. LightGaussian [Fan et al. 2023] optimizes memory usages with a compact format for 3D Gaussians, while VastGaussian [Lin et al. 2024] scales Gaussian Splatting to larger scenes.

However, extracting 3D surfaces from Gaussian splats remains challenging due to their discrete and unstructured nature. To overcome this challenge, SuGaR [Guédon and Lepetit 2023] and NeuGS-[Chen et al. 2023b] favor flat Gaussians that better align with object surfaces. GSDF [Yu et al. 2024b] proposes a dual-branch network combining the standard 3D GS with NeuS [Wang et al. 2021] to improve rendering fidelity and reconstruction accuracy simultaneously. Moreover, post-processing and joint-optimization methods [Guédon and Lepetit 2023] can improve the results, but at the cost of increased training time. 2D GS [Huang et al. 2024] directly replaces 3D Gaussian primitives with flat 2D Gaussians for effective surface reconstruction. Yet, it sacrifices novel-view synthesis capability and training stability because the 2D Gaussians can lead to degenerate scene representation and fail to capture more complicated scenes.

Our work analyzes the depth evaluation in standard 3D Gaussian splats and proposes a novel rasterized method for depth map computation. Our method not only achieves precise surface reconstruction but also retains the rendering and optimization efficiency of 3D Gaussian splats.

3 METHOD

3.1 Gaussain Splatting Preliminary

The standard Gaussian Splatting [Kerbl et al. 2023] represents a 3D scene by a set of translucent 3D Gaussians. Each 3D Gaussian is defined as,

$$G(\mathbf{x}) = e^{-(\mathbf{x}-\mathbf{x}_c)^\top \Sigma^{-1}(\mathbf{x}-\mathbf{x}_c)}, \tag{1}$$

where $\mathbf{x}_c \in \mathbb{R}^3$ is the Gaussian center, and $\Sigma \in \mathbb{R}^{3\times3}$ is the covariance matrix. The covariance Σ is parameterized by a scaling matrix $\mathbf{S}$ and rotation matrix $\mathbf{R}$ as $\Sigma = \mathbf{RSS}^\top\mathbf{R}^\top$.

Approximate Local Affine Projection. Gaussian Splatting approximates the perspective camera projection locally by an affine transformation [Zwicker et al. 2002] for each 3D Gaussian to achieve

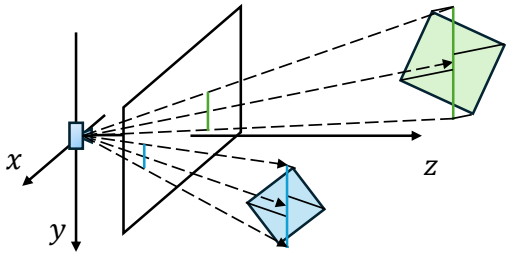

Fig. 3. Illustration of the local affine projection in Gaussian Splatting [Kerbl et al. 2023; Zwicker et al. 2002]. It approximates the perspective projection by a local parallel projection at each Gaussian splat.

efficient rasterized rendering, as illustrated in Figure 3. The projected 2D Gaussian can be computed as,

$$\Sigma' = \mathbf{JW}\Sigma\mathbf{W}^\top\mathbf{J}^\top, \tag{2}$$

where $\Sigma' \in \mathbb{R}^{3\times3}$ is the covariance matrix in camera coordinate frame, $\mathbf{W}$ is the rotation matrix from the world to camera coordinate system, and $\mathbf{J}$ is the Jacobian of the perspective transformation. The 2D Gaussian covariance is obtained by skipping the last row and column of Σ'.

Alpha Blending. Gaussian Splatting sorts the projected 2D Gaussians by their depth and computes the color at each pixel by α-blending,

$$c = \sum_{i\in N} c_i\alpha_i \prod_{j=1}^{i-1}(1-\alpha_j), \tag{3}$$

where c is the rendered pixel color, c_i is the color of the $i-$th Gaussian kernel computed from its spherical harmonics coefficients and viewing direction, α_i is the pixel translucency determined by the opacity of the $i-$th Gaussian kernel and the pixel's position.

3.2 Rasterizing Depth for Gaussian Splats

Standard Gaussian Splatting [Kerbl et al. 2023] evaluates the depth of each 2D Gaussian by its center to sort them for alpha blending. This constant depth per Gaussian splats cannot capture shape details. Therefore, we aim to compute a spatially varying depth within the projected 2D Gaussian and derive a rasterized method for its efficient evaluation.

Assume (u_c, v_c) is the center of the 2D Gaussian. For a pixel (u, v) covered by the projected Gaussian, we compute its depth as,

$$d = z_c + \mathbf{p}\begin{pmatrix}\Delta u\\ \Delta v\end{pmatrix}, \tag{4}$$

where z_c is the depth of the Gaussian center, $\Delta u = u_c - u$ and $\Delta v = v_c - v$ are the relative pixel positions. The 1×2 vector $\mathbf{p} \in \mathbb{R}^2$ is determined by the Gaussian parameters and camera extrinsic parameters. This formulation enables rasterized computation of spatially varying depths within a projected Gaussian. We derive it in detail in this subsection.

3.2.1 Depth Under Perspective Projection. To make the derivation easier to understand, we first introduce the concepts in the camera coordinates with perspective projection. As shown in Figure 4 (a),

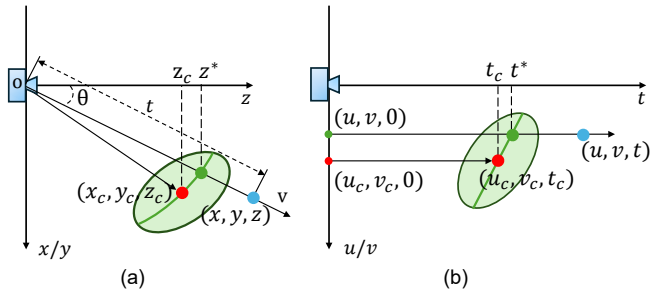

Fig. 4. Intersecting a light ray with a 3D Gaussian in the 'camera space' (a) and 'ray space' (b). The Gaussian center (x_c, y_c, z_c) is transformed to (u_c, v_c, t_c). A point (x, y, z) is transformed to (u, v, t). The green curve in (a) and the green line in (b) represent the set of intersections of the Gaussian with different light rays.

consider a light ray leaving the camera center $\mathbf{o}$ with unit direction $\mathbf{v}$. A point on this ray is parameterized by the distance t to $\mathbf{o}$ as,

$$\mathbf{x} = \mathbf{o} + t\mathbf{v}. \tag{5}$$

The Gaussian value on the ray can be computed as a function of t as follows,

$$G^1(t) = e^{-(\mathbf{o}+t\mathbf{v}-\mathbf{x}_c)^\top\Sigma^{-1}(\mathbf{o}+t\mathbf{v}-\mathbf{x}_c)}. \tag{6}$$

According to Equation 6, the Gaussian value along the ray is a 1D Gaussian function.

We define the 'intersection point' of the ray and the 3D Gaussian to be the point that maximizes the 1D Gaussian function $G^1(t)$. As shown in Figure 4 (a), the green point is the intersection of the ray with the Gaussian splat. The distance t^* between the intersection point and the camera center can be computed in closed-form by locating the maximum value of $G^1(t)$ as,

$$t^* = \frac{\mathbf{v}^\top\Sigma^{-1}(\mathbf{x}_c - \mathbf{o})}{\mathbf{v}^\top\Sigma^{-1}\mathbf{v}}. \tag{7}$$

Equation 7 implies that intersections of a 3D Gaussian and a bundle of light rays form a curved surface, where different pixels have different depth values of t^* and different viewing direction $\mathbf{v}$.

3.2.2 Depth Under Local Affine Projection. Now, we derive the pixel's depth under the local affine projection – the projection model adopted in[Kerbl et al. 2023; Zwicker et al. 2002], where each 3D Gaussian undergoes an affine projection. We will show this simplified projection model allows for a rasterized method to compute the spatially varying depth of a projected Gaussian splat.

Under the local affine projection, each 3D Gaussian is undergoing an affine projection locally, as shown in Figure 4 (b). Following [Zwicker et al. 2002], we refer to the coordinate system in Figure 4 (b) as the 'ray space', while the one in Figure 4 (a) is the 'camera space' for the clarity of discussion.

Transformation from Camera to Ray Space. Ray space is a non-Cartesian coordinate system that enables an easy formulation of our derivation. The blue point in Figure 4 (a), $\mathbf{x} = (x, y, z)^\top$ in the camera space, is transformed to the blue point in Figure 4 (b), $\mathbf{u} = (u, v, t)$ in the ray space. The first two coordinates (u, v) are the image plane coordinates, and t is the distance between the point

and the uv-plane. In other words, $t = \sqrt{x^2 + y^2 + z^2}$. Points on a light ray share the u and v coordinates but have varying distances t to the camera center. Note the light ray direction $\mathbf{v}$ is normalized to $(0, 0, 1)^\top$ in the ray space.

The Gaussian Splats are transformed into the ray space, too. The transformed Gaussian function is,

$$G'(\mathbf{u}) = e^{-(\mathbf{u}-\mathbf{u}_c)^\top \Sigma'^{-1}(\mathbf{u}-\mathbf{u}_c)}, \tag{8}$$

where $\mathbf{u}$ is a point in ray space, $\mathbf{u}_c$ and Σ' are the transformed center and covariance matrix respectively. We denote the transformed Gaussian center as $\mathbf{u}_c = (u_c, v_c, t_c)^\top$. The transformed covariance matrix can be computed according to Equation 2.

Intersection in Ray Space. We derive the intersection in ray space by locating the maximum value of Gaussian on the light ray. Similarly, in the ray space, a point is parameterized by its distance t to the uv-plane as,

$$\mathbf{u} = \mathbf{u}_o + t\mathbf{v}', \tag{9}$$

where $\mathbf{u}_o = (u, v, 0)^\top$ and $\mathbf{v}' = (0, 0, 1)^\top$. By substituting Equation 9 into Equation 8, we can get the 1D Gaussian function defined on the light ray as,

$$G'^1(t) = e^{-(\mathbf{u}_o+t\mathbf{v}'-\mathbf{u}_c)^\top \Sigma'^{-1}(\mathbf{u}_o+t\mathbf{v}'-\mathbf{u}_c)}. \tag{10}$$

Similarly, the maximum point can be located as,

$$t^* = \frac{\mathbf{v}'^\top \Sigma'^{-1}(\mathbf{u}_c - \mathbf{u}_o)}{\mathbf{v}'^\top \Sigma'^{-1}\mathbf{v}'}. \tag{11}$$

While Equation 11 is similar to Equation 7, the direction $\mathbf{v}'$ here is a constant vector $(0, 0, 1)^\top$. Consequently, we can pre-compute $\mathbf{v}'^\top \Sigma'^{-1}\mathbf{v}'$ and $\mathbf{v}'^\top \Sigma'^{-1}$ for each Gaussian. In this way, the intersection point can be computed simply as

$$t^* = \hat{\mathbf{q}}(\mathbf{u}_c - \mathbf{u}_o), \tag{12}$$

where the 1×3 vector $\hat{\mathbf{q}}$ is defined as,

$$\hat{\mathbf{q}} = \frac{\mathbf{v}'^\top \Sigma'^{-1}}{\mathbf{v}'^\top \Sigma'^{-1}\mathbf{v}'}. \tag{13}$$

Depth of Intersection. We now derive the depth value of the intersection point, i.e. the green point in Figure 4 (a) and (b). As shown in Figure 4 (a), t is the distance between the 3D point $\mathbf{x}$ and the camera center $\mathbf{o}$. So, the depth of $\mathbf{x}$ is simply,

$$d = cos\theta\ t^*, \tag{14}$$

where θ is the angle between the light ray and the camera's principal axis. To simplify computation, we approximate it by θ_c, the angle defined by the Gaussian center $\mathbf{x}_c$. Therefore, the depth of $\mathbf{x}$ becomes,

$$d = cos\theta_c t^* = \frac{z_c}{t_c} t^* = \frac{z_c}{t_c}\hat{\mathbf{q}}(\mathbf{u}_c - \mathbf{u}_o) = \hat{\mathbf{p}}(\mathbf{u}_c - \mathbf{u}_o), \tag{15}$$

where $\hat{\mathbf{p}} = \frac{z_c}{t_c}\hat{\mathbf{q}}$ is a constant 1×3 vector for a fixed Gaussian splat.

We can further reformulate Equation 15 as follows,

$$d \quad = \hat{\mathbf{p}}(\mathbf{u}_c - \mathbf{u}_o) = \hat{\mathbf{p}}\begin{pmatrix} u_c - u \\ v_c - v \\ t_c \end{pmatrix} = \hat{\mathbf{p}}\begin{pmatrix} 0 \\ 0 \\ t_c \end{pmatrix} + \hat{\mathbf{p}}\begin{pmatrix} \Delta u \\ \Delta v \\ 0 \end{pmatrix}. \tag{16}$$

As proved in the appendix,

$$\hat{\mathbf{p}}\begin{pmatrix} 0 \\ 0 \\ t_c \end{pmatrix} = z_c. \tag{17}$$

Therefore, we can skip the third element of $\hat{\mathbf{p}}$ to get $\mathbf{p}$ in Equation 4. Thus, we obtain a rasterized method to compute the spatial varying depth for each pixel covered by the Gaussian splat.

3.3 Rasterizing Normal for Gaussian Splats

This section explains the computation of surface normal directions projected by a Gaussian splat. We first show that the intersection points form a plane in ray space as indicated by the green line segment in the Gaussian splat in Figure. 4 (b). Therefore, we take the plane's normal direction as that of the projected Gaussian. We then transform the normal vector from the 'ray space' to the 'camera space' to compute the normal map.

Plane Normal in Ray Space. We derive the plane equation in ray space to get its normal direction. As we shown in the appendix, the intersection point in the ray space is,

$$\mathbf{u} = \begin{pmatrix} u \\ v \\ t^* \end{pmatrix} = \begin{pmatrix} u_c \\ v_c \\ t_c \end{pmatrix} + \begin{pmatrix} -\Delta u \\ -\Delta v \\ (\Delta u \quad \Delta v)\mathbf{q}^\top \end{pmatrix}, \tag{18}$$

where $\mathbf{q}$ is a 1×2 vector, which has the first two components of $\hat{\mathbf{q}}$ in Equation 13. Note that $(u_c, v_c, t_c)^\top$ is the Gaussian center $\mathbf{u}_c$ and is a constant vector. Therefore, all the intersection points between the Gaussian splat and a bundle of light rays should lie on a plane in the ray space. The plane equation can be derived from Equation 18 as,

$$(\mathbf{u} - \mathbf{u}_c) \quad = \quad \begin{pmatrix} -\Delta u \\ -\Delta v \\ (\Delta u \quad \Delta v)\mathbf{q}^\top \end{pmatrix}, \tag{19}$$

$$\begin{pmatrix} \mathbf{q} & 1 \end{pmatrix}(\mathbf{u} - \mathbf{u}_c) \quad = \quad \begin{pmatrix} \mathbf{q} & 1 \end{pmatrix}\begin{pmatrix} -\Delta u \\ -\Delta v \\ (\Delta u \quad \Delta v)\mathbf{q}^\top \end{pmatrix} = 0. \tag{20}$$

According to Equation 20, the vector $\begin{pmatrix} \mathbf{q} & 1 \end{pmatrix}$ is the normal of the plane formed by all the intersection points. We choose the normal pointing toward the image plane, which is

$$\mathbf{n}' = -\begin{pmatrix} \mathbf{q} & 1 \end{pmatrix}^\top. \tag{21}$$

Here, $\mathbf{n}'$ is a 3×1 vector since $\mathbf{q}$ is a 1×2 vector. The denotation $'$ represents parameters in the ray space.

Plane Normal in Camera Space. With the normal direction derived in the ray space, we transform it back to the camera space by the local affine transformation as,

$$\mathbf{n} = \mathbf{J}^\top \mathbf{n}', \tag{22}$$

where $\mathbf{J}$ is the local affine matrix. After transformation, the vector $\mathbf{n}$ is normalized to unit length.

Our rasterized depth and normal maps are derived from general 3D Gaussian Splats under the local affine transformation assumption, which is assumed in the standard Gaussian Splatting [Kerbl et al. 2023]. Consequently, our method can be directly integrated into existing methods utilizing 3D Gaussian Splatting.

3.4 Loss Functions

Even with our derived depth and normal maps, the Gaussian Splatting cannot recover shape details if it is only trained with the photometric supervision $\mathcal{L}_c$ that minimizes the difference between rendered and input images. To address this problem, we follow the 2D Gaussian Splatting [Huang et al. 2024] to apply additional depth distortion loss and normal consistency loss. We detail these two terms here to make this paper self-contained.

Depth distortion loss. The depth distortion loss encourages different Gaussian splats on a ray to be close to each other by minimizing the disparity of their depths as,

$$\mathcal{L}_d = \sum_{i,j} \omega_i \omega_j (d_i - d_j)^2, \tag{23}$$

where $\omega_i = \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j)$ is the blending weight of $i-$th Gaussian and d_i is its depth.

Normal consistency loss. The normal consistency loss ensures the Gaussian splats align with the surface by measuring the consistency between normal directions computed from the Gaussian and the depth map, respectively,

$$\mathcal{L}_n = \sum_i \omega_i (1 - \mathbf{n}_i^\top \tilde{\mathbf{n}}), \tag{24}$$

where $\tilde{\mathbf{n}}$ is the surface normal direction obtained by applying finite-difference on the depth map.

Our final training loss $\mathcal{L}$ is,

$$\mathcal{L} = \mathcal{L}_c + w_d \mathcal{L}_d + w_n \mathcal{L}_n. \tag{25}$$

4 EXPERIMENT

We evaluate the performance of our method on both novel view synthesis and 3D reconstruction with standard benchmark datasets and compare it with state-of-the-art (SOTA) implicit and explicit approaches.

4.1 Experimental Setup

4.1.1 Implementation Details. We build our method upon the public code of 3D GS [Kerbl et al. 2023] and implement customized CUDA kernels for our rasterized depth, normal map computation and regularization terms. We use the default parameters of 3D GS and incorporate the 3D filter proposed in Mip-Splatting [Yu et al. 2024a] and the densification approach proposed by GOF [Yu et al. 2024c]. We use a strategy similar to 3D GS to stop densification at 15k iterations and optimize all of our models for 30k iterations. Following GOF [Yu et al. 2024c], We set $w_d = 100$ and $w_n = 5$ in all our experiments and detach the gradient propagation of the blending weight ω when calculating depth distortion loss. All our experiments are conducted on a single NVIDIA H800 GPU.

4.1.2 Mesh Extraction. We render depth maps for all training views and then adopt TSDF [Curless and Levoy 1996b] for mesh extraction. Furthermore, we adopt marching tetrahedra algorithm [Yu et al. 2024c] to our method, which conducts marching cube on tetrahedra grids and uses binary search to refine the surface. Nevertheless, marching tetrahedra is designed for 3D Gaussian under perspective camera model. So it cannot improve our method by a large margin as GOF, and cannot be directly applied on 2D GS. Table. 1 shows the accuracy of our method with TSDF fusion, and Table. 2 reports the accuracy of our method with either TSDF fusion or marching tetrahedra.

4.1.3 Datasets. We conduct the surface reconstruction experiments on the DTU [Jensen et al. 2014] and Tanks & Temples dataset (TNT) [Knapitsch et al. 2017]. Following the prior works, 15 scenes from the DTU dataset and 6 scenes from the TNT dataset are selected for evaluation. With the camera poses provided by datasets, we apply COLMAP [Schönberger and Frahm 2016] to generate a sparse point cloud of each scene for initialization.

For novel view synthesis, we experiment on the Mip-NeRF360 dataset [Barron et al. 2022] and the Synthetic-NeRF [Mildenhall et al. 2021] dataset. The Mip-NeRF360 dataset contains large indoor and outdoor scenes, while the Synthetic-NeRF contains object-level scenes with challenging reflections and detailed shapes.

4.1.4 Evaluation protocols. To facilitate the comparison with previous methods, we report the **Chamfer Distance (CD)** on the DTU dataset and the **F1-score** on the TNT dataset. To evaluate the quality of novel view syntheis, we adopt **PSNR**, **SSIM** and **LIPIPS** as metrics.

4.1.5 Baselines. We compare our method with the SOTA Gaussian Splatting methods for surface reconstruction, including GOF [Huang et al. 2024], SuGaR [Guédon and Lepetit 2023], 2D GS [Huang et al. 2024], and 3D GS [Kerbl et al. 2023]. We also compare with NeRF-based implicit methods, including VolSDF [Yariv et al. 2021], NeuS [Wang et al. 2021], and Neuralangelo [Li et al. 2023]. These methods adopt Signed Distance Function (SDF) to represent the scene and transform the SDF to opacity for ray tracing based volume rendering.

4.2 Comparison

4.2.1 Surface Reconstruction. We compare our method with existing methods on the DTU and the TNT dataset. As shown in Table 1, our method outperforms all Gaussian splatting-based methods and achieves competitive results with Neuralangelo in terms of Chamfer Distance error. Figure 5 visualize some results generated by different Gaussian splatting based methods. Our method produces smooth and precise shapes. In contrast, 3D GS generates noisy meshes due to the biased depth rendering. At the same time, 2D GS and SuGaR tend to be unstable at the specular and highlight areas, thus producing inaccurate surface prediction, while our method is more robust to these problems. We include more comparison in Figure 7 and add visualizations of the reconstructed meshes in Figure 8 and Figure 9.

Table 2 compares our method with other methods on the TNT dataset. Again, our method outperforms all 3D GS-based methods. However, our F1-score is lower than Neuralangelo and close to NeuS. We believe this is mainly due to the voxel resolution during our TSDF fusion of the depth maps. The TNT dataset contains large-scale scenes such as the '*Courthouse*' and '*Meetingroom*'. Due to memory constraints, our current version is limited to a lower-resolution voxel grid in TSDF fusion. We could employ a multi-resolution TSDF fusion [Vespa et al. 2019] to obtain a better F1 score. We leave this task to future research.

Table 1. **Quantitative comparison on the DTU Dataset [Jensen et al. 2014]**. We report the Chamfer Distance error of different methods. Our method achieves the best performance among all explicit Gaussian Spaltting based methods, producing comparable accuracy as NeuraLangelo. We compare our method with other Gaussian Splatting methods under different resolutions. Methods with and without '(full resolution)' are trained under full resolution and half resolution respectively.

		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
	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
	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
	Neuralangelo [Li et al. 2023]	0.37	0.72	0.35	0.35	0.87	0.54	0.53	1.29	0.97	0.73	0.47	0.74	0.32	0.41	0.43	0.61	> 12h
explicit	3D GS [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	5.2m
	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	52m
	2D GS [Huang et al. 2024]	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	8.9m
	GOF [Yu et al. 2024c]	0.50	0.82	0.37	0.37	1.12	0.74	0.73	1.18	1.29	0.68	0.77	0.90	0.42	0.66	0.49	0.74	55m
	Our (20K iterations)	0.45	0.74	0.36	0.39	0.90	0.73	0.77	1.13	1.26	0.66	0.62	0.83	0.37	0.61	0.46	0.69	5.0m
	Our (30K iterations)	0.46	0.73	0.33	0.38	0.79	0.75	0.76	1.19	1.22	0.62	0.70	0.78	0.36	0.68	0.47	0.68	8.3m
	2D GS (full resolution) [Huang et al. 2024]	0.53	0.84	0.39	0.37	1.33	1.23	0.96	1.35	1.29	0.81	0.72	1.62	0.40	0.74	0.51	0.87	26.6m
	GOF (full resolution) [Yu et al. 2024c]	0.52	0.85	0.50	0.40	1.38	1.05	0.96	1.31	1.35	0.80	0.82	0.86	0.47	0.62	0.46	0.82	136m
	Our (full resolution)	0.40	0.71	0.33	0.37	0.87	0.79	0.77	1.22	1.26	0.70	0.65	0.85	0.33	0.66	0.44	0.69	20.2m

Table 2. **Quantitative comparison on the Tanks & Temples Dataset [Knapitsch et al. 2017]**. We report the F1-score and average optimization time. All results are evaluated with the official evaluation scripts. ♯ represents using marching tetrahedra algorithm[Yu et al. 2024c] to extract meshes. Our method achieves the best F1 score among Gaussian Splatting and TSDF fusion based methods while maintaining the capacity of fast training.

	Implicit			Explicit						
	NeuS	Geo-Neus	Neurlangelo	SuGaR	3D GS	2D GS	GOF	GOF♯	Our	Our♯
Barn	0.29	0.33	0.70	0.14	0.13	0.36	0.37	0.51	0.43	0.43
Caterpillar	0.29	0.26	0.36	0.16	0.08	0.23	0.21	0.41	0.26	0.32
Courthouse	0.17	0.12	0.28	0.08	0.09	0.13	0.11	0.28	0.11	0.21
Ignatius	0.83	0.72	0.89	0.33	0.04	0.44	0.63	0.68	0.73	0.69
Meetingroom	0.24	0.20	0.32	0.15	0.01	0.16	0.23	0.28	0.17	0.25
Truck	0.45	0.45	0.48	0.26	0.19	0.26	0.50	0.59	0.53	0.51
Mean	0.38	0.35	0.50	0.19	0.09	0.30	0.34	0.46	0.37	0.40
Time	>24h	>24h	>24h	73 m	7.9 m	12.3 m	69 m	69m	11.5 m	11.5 m

Table 3. **Quantitative results on Mip-NeRF 360 [Barron et al. 2022] dataset.** Our method achieved SOTA NVS results, especially in the outdoor scenes in terms of LPIPS.

	Outdoor Scene			Indoor scene		
	PSNR ↑	SSIM ↑	LPIPS ↓	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
MipNeRF360	24.47	0.691	0.283	31.72	0.917	0.180
Mobile-NeRF	21.95	0.470	0.470	-	-	-
BakedSDF	22.47	0.585	0.349	27.06	0.836	0.258
SuGaR	22.93	0.629	0.356	29.43	0.906	0.225
BOG	23.94	0.680	0.263	27.71	0873	0.227
3D GS	24.64	0.731	0.234	30.41	0.920	0.189
Mip-Splatting	24.65	0.729	0.245	30.90	0.921	0.194
2D GS	24.34	0.717	0.246	30.40	0.916	0.195
GOF	24.82	0.750	0.202	30.79	0.924	0.184
Ours	25.17	0.764	0.199	30.74	0.928	0.165

4.2.2 Novel view synthesis. We further compare our method against previous methods on the Mip-NeRF360 and Synthetic NeRF datasets to evaluate their novel view synthesis capability. Table 3 and Table 4 present quantitative results. As we can see from these tables, our method achieves the highest average PSNR on the Synthetic NeRF dataset and gets the highest score in most metrics on the Mip-NeRF360 dataset. In comparison, the 2D GS and SuGaR methods generate poorer novel view rendering than the standard 3D GS as shown in Table 3 and Table 4. This comparison suggests that the planar Gaussian constraints adopted in 2D GS and SuGaR hurt the model's performance in representing complicated scenes. Our method keeps the original 3D GS and achieves better data representation and novel view synthesis results.

Moreover, we show the novel view synthesis results on the Mip-NeRF360 dataset in Figure 6. As shown in the framed regions, our method can render high-quality images, while SuGaR generates

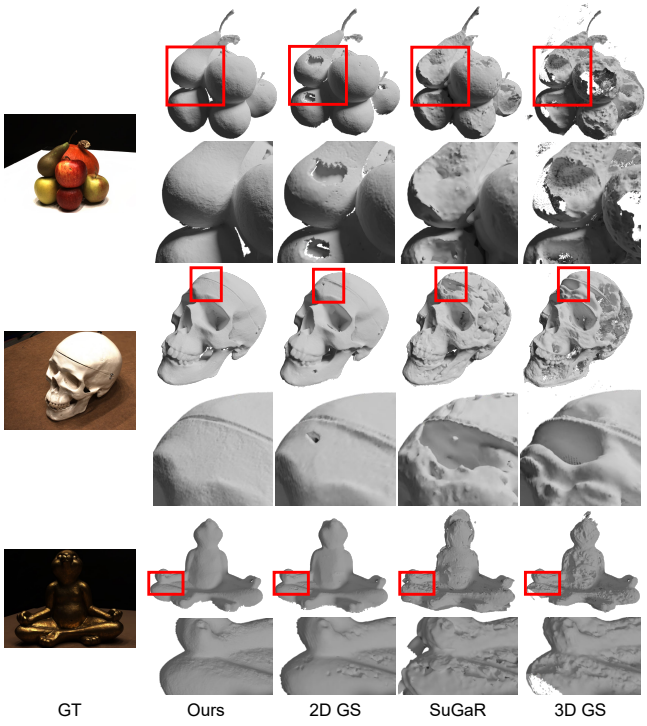

Fig. 5. Visual comparison of our method and the previous Gaussian-based methods on the DTU dataset.

Fig. 6. Comparisons of our method to previous GS-based methods. Our method achieves high-quality novel view synthesis, while others generate blurry results around the bench and artifacts at the distant wall, as shown in the framed regions.

Table 4. **PSNR scores for Synthetic NeRF [Mildenhall et al. 2021]**. Our method outperforms others in novel view synthesis. We use ⋆ to represent 2D GS trained with regularization. To ensure fair comparisons, we train our model, GOF, and 2D GS with regularization. While 2D GS reports geometry results with regularization, it does not do so for novel view synthesis.

	Mic	Chair	Ship	Materials	Lego	Drums	Ficus	Hotdog	Avg.
Plenoxels	33.26	33.98	29.62	29.14	34.10	25.35	31.83	36.81	31.76
INGP-Base	36.22	35.00	31.10	29.78	36.39	26.02	33.51	37.40	33.18
Mip-NeRF	36.51	35.14 ‘	30.41	30.71	35.70	25.48	33.29	37.48	33.09
Point-NeRF	35.95	35.40	30.97	29.61	35.04	26.06	36.13	37.30	33.30
3D GS	35.36	35.83	30.80	30.00	35.78	26.15	34.87	37.72	33.32
2D GS⋆	34.86	34.77	30.47	29.58	33.98	25.47	35.25	36.83	32.65
2D GS	35.09	35.05	30.60	29.74	35.10	26.05	35.57	37.36	33.07
GOF	35.81	36.34	30.71	30.19	35.64	26.21	35.25	37.54	33.46
Our	35.46	36.46	31.35	30.34	35.75	26.29	35.37	37.76	33.60

blurry results for places with complicated shapes, such as grass and leaves. Due to unstable optimization, it also generates artifacts at the distant wall. Please zoom in on the digital version for clearer visualization. More qualitative comparisons can be found in Figure 10.

4.2.3 Computational Efficiency. Table 1 and Table 2 provides the training time of different methods. In general, GS-based explicit methods are much more computationally efficient than implicit NeRF-based methods. Our method reconstructs a scene in about 8.3 minutes on DTU dataset and 11.5 minutes on TnT dataset, while all

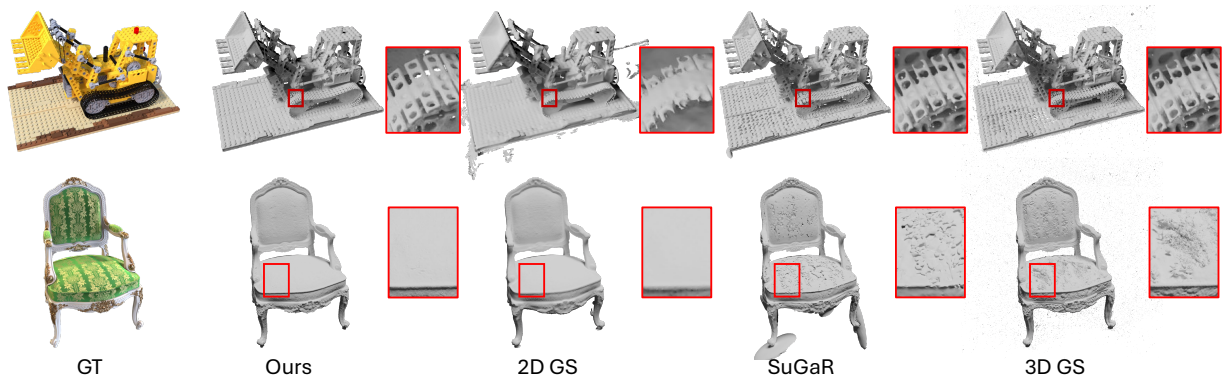

Fig. 7. Qualitative comparisons in surface reconstruction between our method and previous Gaussian-based methods on the NeRF Synthetic dataset.

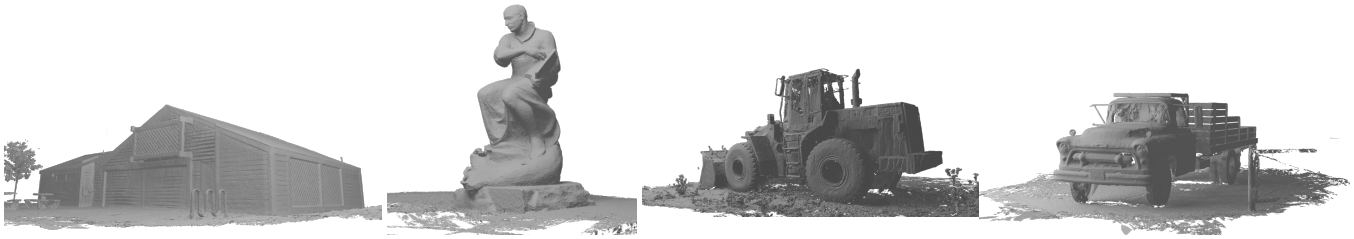

Fig. 8. Surface reconstruction results on the Tanks & Templs dataset [Knapitsch et al. 2017].

Fig. 9. Surface reconstruction results on the DTU dataset [Jensen et al. 2014].

implicit methods take hours. As shown in Table 1, 2D GS takes 8.9 minutes on DTU dataset and 12.3 minites on TnT dataset, which is slower than our method. GOF is even slower, taking one hour for training, which can be attributed to its time-consuming ray tracing technique. Compared with 3D GS, the extra time-consumption comes from the forward and backward process of Geometry regularization that requires per-pixel computation. And we believe it can be further improved by more efficient regularization in the future. We further train our model with 20K iterations on DTU dataset (shown in Table 1). Our method could maintain high accuracy with fewer optimization steps like 20k, which validates the effectiveness of our proposed depth resterization.

5 CONCLUSION

We introduce RaDe-GS, which includes a rasterized method to compute the depth and surface normal maps, to improve Gaussian Splatting's shape reconstruction capability while maintaining its training and rendering efficiency. Our method is derived from general Gaussian primitives, it can be easily integrated into any Gaussian-Splatting-based method. Our experiments on multiple public datasets demonstrate that our RaDe-GS produces superior results than previous implicit or explicit methods.

Limitation. Our current TSDF fusion is limited to low-resolution voxel grids for large-scale scenes, thus hindering accurate surface extraction from Gaussians. The adaptive TSDF fusion techniques can be used to extract mesh in a coarse-to-fine manner, which leads

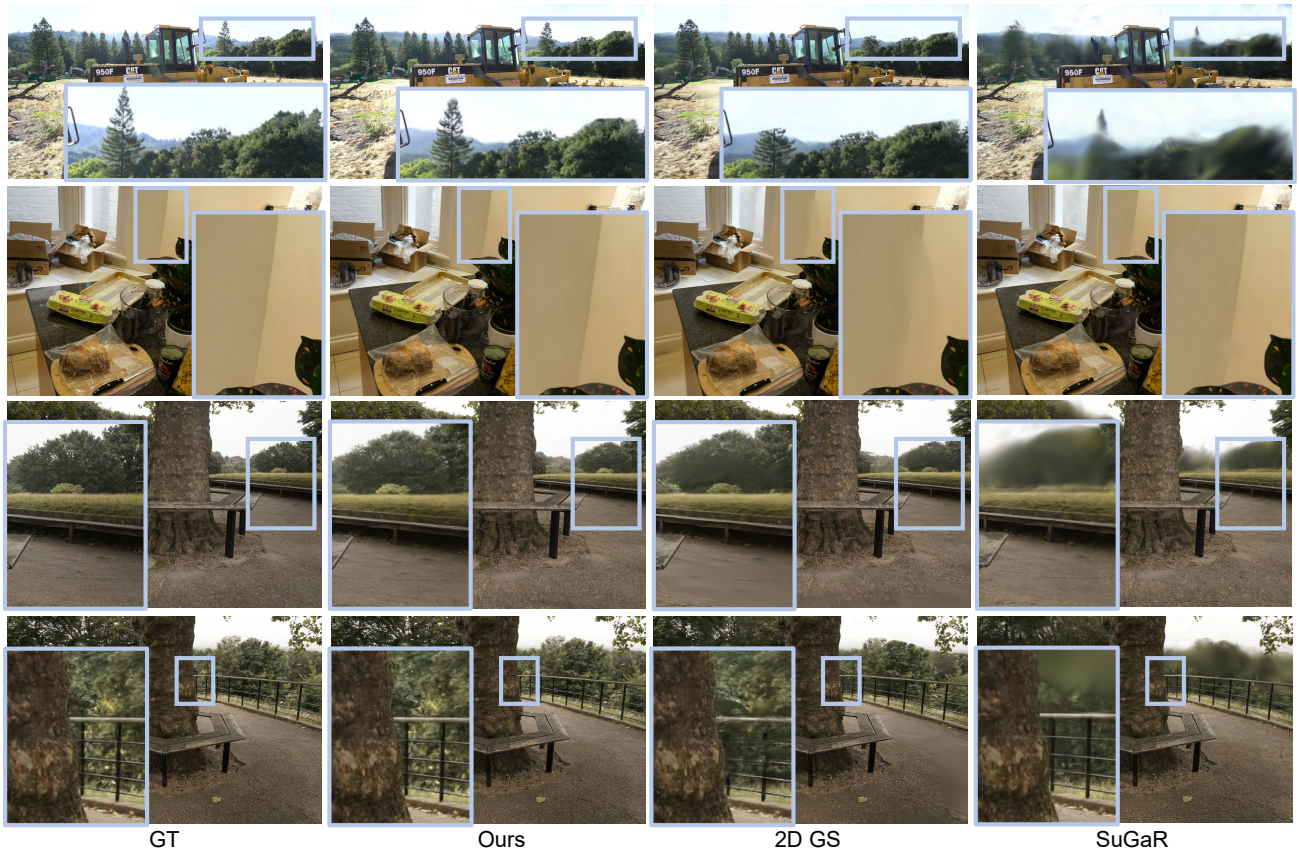

Fig. 10. We show more novel view rendering comparisons of ours to previous methods and the corresponding ground truth images from held-out test views. The scenes are from Tanks & Temples dataset and Mip-NeRF360 dataset. We highlight some regions with red rectangles.

to less memory consumption but better geometry quality. For the object containing reflective surfaces, e.g, the Metal Scissor of DTU dataset (shown in Figure 9), our method fails to accurately reconstruct the reflective surfaces limited by the simple color function used in 3D GS. This problem may be alleviated by combining with advanced color representation proposed in GaussianShader [Jiang et al. 2023].

REFERENCES

Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. 2009. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.* 28, 3 (2009), 24.

Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. 2021. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5855–5864.

Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. 2022. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5470–5479.

Michael Bleyer, Christoph Rhemann, and Carsten Rother. 2011. PatchMatch Stereo - Stereo Matching with Slanted Support Windows. In *BMVC*.

Hanlin Chen, Chen Li, and Gim Hee Lee. 2023b. Neusg: Neural implicit surface reconstruction with 3d gaussian splatting guidance. *arXiv preprint arXiv:2312.00846* (2023).

Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 2023a. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16569–16578.

Kai Cheng, Xiaoxiao Long, Kaizhi Yang, Yao Yao, Wei Yin, Yuexin Ma, Wenping Wang, and Xuejin Chen. 2024. GaussianPro: 3D Gaussian Splatting with Progressive Propagation. *arXiv preprint arXiv:2402.14650* (2024).

Robert Collins. 1996. A space-sweep approach to true multi-image matching. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Brian Curless and Marc Levoy. 1996a. A Volumetric Method for Building Complex Models from Range Images. In *SIGGRAPH*.

Brian Curless and Marc Levoy. 1996b. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 303–312.

Yikang Ding, Wentao Yuan, Qingtian Zhu, Haotian Zhang, Xiangyue Liu, Yuanjiang Wang, and Xiao Liu. 2022. TransMVSNet: Global Context-aware Multi-view Stereo Network with Transformers. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2023. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245* (2023).

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5501–5510.

Simon Fuhrmann, Fabian Langguth, and Michael Goesele. 2014. Mve-a multi-view reconstruction environment. *GCH* 3 (2014), 4.

Xiaodong Gu, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan, and Ping Tan. 2019. Cascade Cost Volume for High-Resolution Multi-View Stereo and Stereo Matching. *arxiv preprint arXiv:1912.06378*.

Xiaodong Gu, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan, and Ping Tan. 2020. Cascade cost volume for high-resolution multi-view stereo and stereo matching. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2495–2504.

Antoine Guédon and Vincent Lepetit. 2023. SuGaR: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *arXiv preprint arXiv:2311.12775* (2023).

Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. 2021. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5875–5884.

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2024. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. *arXiv preprint arXiv:2403.17888* (2024).

Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. 2014. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 406–413.

Yingwenqi Jiang, Jiadong Tu, Yuan Liu, Xifeng Gao, Xiaoxiao Long, Wenping Wang, and Yuexin Ma. 2023. GaussianShader: 3D Gaussian Splatting with Shading Functions for Reflective Surfaces. *arXiv preprint arXiv:2311.17977* (2023).

Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. 2017. End-to-End Learning of Geometry and Context for Deep Stereo Regression. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics* 42, 4 (2023), 1–14.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *ACM Transactions on Graphics* 36, 4 (2017).

Vladimir Kolmogorov and Ramin Zabih. 2001. Computing visual correspondence with occlusions using graph cuts. In *ICCV*. IEEE.

Vladimir Kolmogorov and Ramin Zabin. 2004. What energy functions can be minimized via graph cuts? *IEEE transactions on pattern analysis and machine intelligence* 26, 2 (2004), 147–159.

Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H. Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. 2023. Neuralangelo: High-Fidelity Neural Surface Reconstruction. arXiv:2306.03092 [cs.CV]

Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, et al. 2024. VastGaussian: Vast 3D Gaussians for Large Scene Reconstruction. *arXiv preprint arXiv:2402.17427* (2024).

Wenjie Luo, Alexander G. Schwing, and Raquel Urtasun. 2016. Computing the Stereo Matching Cost with a Convolutional Neural Network. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)* 41, 4 (2022), 1–15.

Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee, 127–136.

Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. 2023. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–12.

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. 2003. Stereo matching using belief propagation. *TPAMI* 25 (2003), 787 – 800. Issue 7.

Emanuele Vespa, Nils Funk, Paul HJ Kelly, and Stefan Leutenegger. 2019. Adaptive-resolution octree-based volumetric SLAM. In *2019 International Conference on 3D Vision (3DV)*. IEEE, 654–662.

Jiepeng Wang, Peng Wang, Xiaoxiao Long, Christian Theobalt, Taku Komura, Lingjie Liu, and Wenping Wang. 2022. Neuris: Neural reconstruction of indoor scenes using normal priors. In *European Conference on Computer Vision*. Springer, 139–155.

Linhan Wang, Kai Cheng, Shuo Lei, Shengkun Wang, Wei Yin, Chenyang Lei, Xiaoxiao Long, and Chang-Tien Lu. 2024. DC-Gaussian: Improving 3D Gaussian Splatting for Reflective Dash Cam Videos. *arXiv preprint arXiv:2405.17705* (2024).

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689* (2021).

Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. 2018. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European conference on computer vision (ECCV)*. 767–783.

Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems* 34 (2021), 4805–4815.

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5752–5761.

Mulin Yu, Tao Lu, Linning Xu, Lihan Jiang, Yuanbo Xiangli, and Bo Dai. 2024b. GSDF: 3DGS Meets SDF for Improved Rendering and Reconstruction. *arXiv preprint arXiv:2403.16964* (2024).

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2024a. Mip-Splatting: Alias-free 3D Gaussian Splatting. *Conference on Computer Vision and Pattern Recognition (CVPR)* (2024).

Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. 2022. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *Advances in neural information processing systems* 35 (2022), 25018–25032.

Zehao Yu, Torsten Sattler, and Andreas Geiger. 2024c. Gaussian Opacity Fields: Efficient and Compact Surface Reconstruction in Unbounded Scenes. *arXiv preprint arXiv:2404.10772* (2024).

Christopher Zach, Thomas Pock, and Horst Bischof. 2011. A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration. In *International Conference on Computer Vision (ICCV)*.

Jure Zbontar and Yann LeCun. 2015. Efficient Deep Learning for Stereo Matching. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2002. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics* 8, 3 (2002), 223–238.

A DETAILS ON RAY SPACE DEPTH

In this section, we will show more derivation of depth rasterization. In the main submission, we formulate Gaussian depth as:

$$d = z_c + \mathbf{p}\begin{pmatrix}\Delta u\\ \Delta v\end{pmatrix}, \tag{26}$$

where z_c is the depth of the Gaussian center, $\Delta u = u_c - u$ and $\Delta v = v_c - v$ are the relative pixel positions. We have derived the second term $\mathbf{p}\begin{pmatrix}\Delta u\\ \Delta v\end{pmatrix}$, and now we show how we reach the first term z_c.

In Equation 16 of the main submission, we have divided the depth of a Gaussian into two parts:

$$d = \hat{\mathbf{p}}\begin{pmatrix}0\\0\\t_c\end{pmatrix} + \hat{\mathbf{p}}\begin{pmatrix}\Delta u\\ \Delta v\\ 0\end{pmatrix}, \tag{27}$$

where $\hat{\mathbf{p}}$ is in form of:

$$\hat{\mathbf{p}} = \frac{z_c}{t_c}\frac{\mathbf{v'}^\top \Sigma'^{-1}}{\mathbf{v'}^\top \Sigma'^{-1}\mathbf{v'}}, \tag{28}$$

where t_c is the distance from camera center to Gaussian center, Σ' is the Gaussian covariance in ray space, and $\mathbf{v'} = (0,0,1)^\top$ is a constant vector in ray space.

We will simplify the $\hat{\mathbf{p}}(0,0,t_c)^T$ of Equation 27. By substituting Equation 28 into the first part of Equation 27, we get:

$$\begin{aligned}\hat{\mathbf{p}}\begin{pmatrix}0\\0\\t_c\end{pmatrix} &= \frac{z_c}{t_c}\frac{\mathbf{v'}^\top \Sigma'^{-1}}{\mathbf{v'}^\top \Sigma'^{-1}\mathbf{v'}}\begin{pmatrix}0\\0\\t_c\end{pmatrix}\\ &= \frac{z_c}{t_c}\frac{\mathbf{v'}^\top \Sigma'^{-1}}{\mathbf{v'}^\top \Sigma'^{-1}\mathbf{v'}}(t_c\mathbf{v'})\\ &= \frac{z_c}{t_c}\frac{\mathbf{v'}^\top \Sigma'^{-1}\mathbf{v'}}{\mathbf{v'}^\top \Sigma'^{-1}\mathbf{v'}}t_c\\ &= z_c.\end{aligned} \tag{29}$$

B DETAILS ON GAUSSIAN NORMAL

In this section, we show the detailed derivation of Equation 18 in the main submission. Given image plane coordinates (u, v), ray from

(u, v) intersects with Gaussian on the point $\mathbf{u}$.

$$\mathbf{u} = \begin{pmatrix} u \\ v \\ t^* \end{pmatrix}. \tag{30}$$

As shown in the main submission, $t^* = \frac{t_c}{z_c} d$, and d is shown in Equation 26. We plug in and get that:

$$\begin{aligned} \mathbf{u} = \begin{pmatrix} u \\ v \\ t^* \end{pmatrix} = \begin{pmatrix} u \\ v \\ \frac{t_c}{z_c} d \end{pmatrix} \\ = \begin{pmatrix} u_c - \Delta u \\ v_c - \Delta v \\ t_c + (\Delta u \quad \Delta v)\mathbf{q}^\top \end{pmatrix} = \begin{pmatrix} u_c \\ v_c \\ t_c \end{pmatrix} + \begin{pmatrix} -\Delta u \\ -\Delta v \\ (\Delta u \quad \Delta v)\mathbf{q}^\top \end{pmatrix}, \end{aligned} \tag{31}$$

where $\mathbf{q} = \frac{t_c}{z_c}\mathbf{p}$. Since $\hat{\mathbf{q}} = \frac{t_c}{z_c}\hat{\mathbf{p}}$ (Equation 15 in the main submission) and $\mathbf{p}$ contains the first two components of $\hat{\mathbf{p}}$, we know that $\mathbf{q}$ is the same as $\hat{\mathbf{q}}$ skipping the last component.