

UniGS: Unified Geometry-Aware Gaussian Splatting for Multimodal Rendering

Yusen Xie¹ Zhenmin Huang² Jianhao Jiao³ Dimitrios Kanoulas³ Jun Ma^{1,2}
¹HKUST (GZ) ²HKUST ³UCL

{yxie827@connect.hkust-gz.edu.cn; zhuangdf@connect.ust.hk; ucacjji@ucl.ac.uk;
d.kanoulas@ucl.ac.uk; jun.ma@ust.hk}

Project Page: <https://github.com/xieyuser/UniGS>

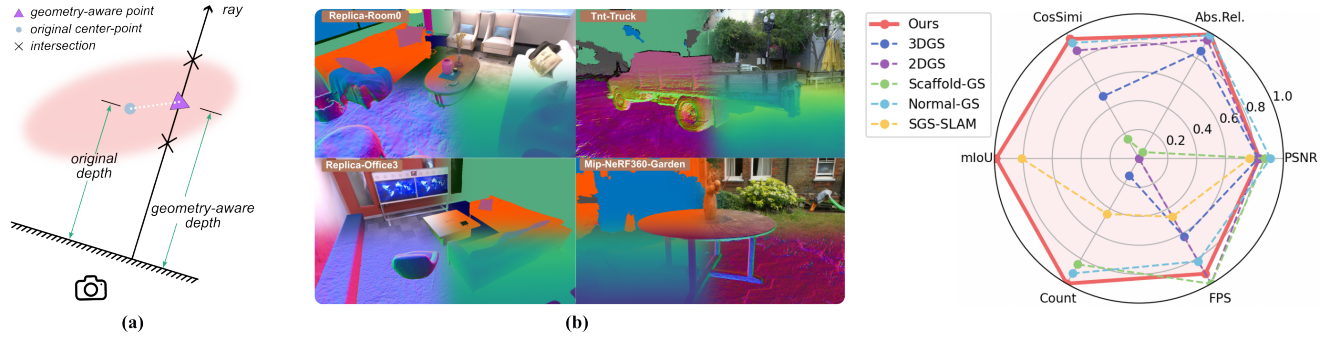


Figure 1. We propose a unified 3D Gaussian Splatting framework that jointly predicts RGB, depth, normal, and semantic map through a single forward rasterization process. (a) Our geometry-aware rendering method explicitly incorporates rotation and scaling attribute into both the rasterization and gradient propagation processes, where we derive an analytical solution for gradient propagation to significantly accelerate optimization. (b) We present multimodal rendering results across diverse scenes, showcasing RGB images, depth maps, surface normals, and semantic segmentation map in a four-panel visualization layout. (c) A comprehensive comparison with widely-used baselines in *Replica-Room0* dataset, underscoring the full modality coverage and strong performance of our framework. All scores are normalized to the $[0, 1]$ range by defining the worst and best reference values for each metric individually, ensuring 1 represents optimal performance. Incomplete dashed lines indicate that the corresponding baseline does not support specific modalities. Evaluation covers PSNR for RGB quality, absolute relative error (*Abs.Rel.*) for depth estimation, cosine similarity (*CosSimi*) for normal estimation, mean intersection-over-union (*mIoU*) for semantic segmentation, as well as system performance metrics including the number of Gaussian primitives (*Count*) and rendering speed in frames per second (*FPS*).

Abstract

In this paper, we propose UniGS, a unified map representation and differentiable framework for high-fidelity multimodal 3D reconstruction based on 3D Gaussian Splatting. Our framework integrates a CUDA-accelerated rasterization pipeline capable of rendering photo-realistic RGB images, geometrically accurate depth maps, consistent surface normals, and semantic logits simultaneously. We redesign the rasterization to render depth via differentiable ray-ellipsoid intersection rather than using Gaussian centers, enabling effective optimization of rotation and scale attribute through analytic depth gradients. Furthermore, we derive the analytic gradient formulation for surface normal rendering, ensuring geometric consistency among reconstructed 3D scenes. To improve computational and storage efficiency, we introduce a learnable attribute that enables differentiable pruning of Gaussians with minimal contribution during training. Quantitative and qualitative ex-

periments demonstrate state-of-the-art reconstruction accuracy across all modalities, validating the efficacy of our geometry-aware paradigm. Source code and multimodal viewer will be available on GitHub.

1. Introduction

High-quality novel view synthesis (NVS) and fast multimodal 3D reconstruction remains a challenging problem in computer graphics and attracts widespread attention in fields such as autonomous driving [19, 20, 67, 79] and robotics [40, 83]. In recent years, neural radiance fields (NeRF) [1, 2, 43, 56] and 3D Gaussian Splatting (3DGS) [26, 27] have made big breakthroughs in this field. NeRF uses coordinate-based networks to store 3D scene in multi-layer perceptrons (MLPs) via ray sampling, bringing in photo-realistic reconstruction. 3DGS, another approach, uses explicit 3D Gaussian primitives and modern GPU-accelerated splatting, and this design has been shown

to enable high-quality real-time NVS by many 3DGS-based frameworks [12, 18, 30, 38, 39, 41, 46, 50, 66, 67]. However, the two types of existing methods still have many issues, which mainly focus on the following aspects. NeRF requires long training times, lacks easy scene editability, and its implicit 3D information storage method renders geometric reconstruction less intuitive [43, 80]. Additionally, it performs poorly when applied to large-scale scenes [51, 54]. 3DGS and its related frameworks have more challenges: 1) Poor geometric structure and consistency: conventional image-centric 3DGS often ignores objects’ geometric structure and semantic info. It only uses Gaussian centers to represent depth, leaving out Gaussian rotation and scale attributes. This stops gradient propagation for geometric constraints (e.g. depth and normal), leading to inconsistent surface normals [26, 34, 41, 62] and misalignment between Gaussians primitives and reconstructed surfaces. 2) Inefficient densification and pruning: some 3DGS methods create too many redundant Gaussians primitives to fit RGB images well [26, 27, 34, 67]. Though this improves metrics, it slows down rendering and uses more GPU memory, losing 3DGS’s real-time edge. 3) Poor unified multimodal framework: some 3DGS-related methods use separate frameworks for multimodal rendering [15, 33], which causes repeated calculations and inconsistent 3D scene representations, failing to unify multimodal modeling. Existing studies have tried single or multiple scene modalities [10, 34, 41, 62], but there is still no unified framework that combines all these modalities while keeping storage efficient and rendering fast. This gap serves as the key motivation for our approach.

Against this backdrop, we propose **UniGS**, a *unified* geometry-aware 3D Gaussian Splatting map representation and framework that establishes a new paradigm for consistent 3D scene reconstruction. During rendering, each pixel is shaded via backward ray tracing to determine its intersection with individual Gaussian ellipsoids (Fig. 1(a)). This formulation explicitly integrates Gaussian rotation and scale attributes into the rendering pipeline. By leveraging analytical gradients derived from depth and normal supervision, it enables efficient gradient-based optimization of these geometric attributes. Furthermore, we introduce a learnable attribute to continuously prune Gaussians. Experimental results demonstrate that this attribute significantly improves computational and storage efficiency without sacrificing rendering or geometric accuracy. Crucially, our framework achieves high-fidelity rendering of *RGB*, *depth*, *normals*, and *semantic logits* through a single differentiable pipeline simultaneously (Fig. 1(b)). The loss from each modality can be back-propagated via analytical gradients to the shared 3D Gaussian representation, thereby maintaining consistency across all modalities in the reconstructed environment (Fig. 1(c)). Particularly, our method achieves

a 66.4% improvement in depth estimation accuracy while reducing the Gaussian primitive count by 17.2%. In this paper, our contributions are summarized as follows:

- We develop a novel differentiable depth rasterization method based on ray-ellipsoid intersection, which enables optimization of Gaussian rotation and scale attributes via analytic gradients propagation. This approach ensures that Gaussian primitives conform closely to underlying surface geometries, significantly improving geometric consistency in reconstructed scenes.
- We propose a trainable gradient factor in map representation for dynamically pruning insignificant Gaussians in a differentiable manner, which enhances rendering speed and storage efficiency.
- We propose a unified map representation and differentiable rasterization framework that integrates multimodal data. This integration not only enables mutual enhancement across modalities but also improves rendering efficiency and strengthens the consistency of the reconstructed environment.
- Through comprehensive experimental evaluation, we demonstrate that our approach achieves state-of-the-art reconstruction quality across all modalities while consistently maintaining real-time performance.

2. Related Works

Photo-Realistic Rendering and Densification. In the domain of photo-realistic reconstruction and rendering, NeRF and 3DGS represent implicit and explicit techniques, respectively. NeRF [1, 43] and its variants pioneer the use of implicit neural representations for NVS, achieving high-fidelity rendering by optimizing a continuous volumetric scene network using multi-view images. However, this approach typically requires extremely long training times. Subsequent works incorporate geometric prior structures [2, 55, 74] or efficient encoding [45] to accelerate the process. Despite these improvements, NeRF still faces inherent limitations, such as the difficulty in editing implicitly represented scenes. In contrast, 3DGS [26, 68] employs explicit Gaussian primitives as scene representations. Through compression [12, 18, 30, 41, 46] and efficient geometric structures [4, 27], several studies extend 3DGS to large-scale outdoor [35, 50, 66, 67] and urban [38, 39] environments while maintaining high-quality reconstruction. But the aforementioned research primarily focuses on RGB-based rendering and seldom incorporates explicit geometric information of the environment. Moreover, impressive efforts have been made to enhance optimization and storage efficiency. SteepestGS [59] uncovers fundamental principles of density control. DashGaussian [5] significantly accelerates 3DGS optimization. GeoTexDensifier [21] utilizes geometric information for Gaussian densification, and Pixel-GS [78] introduces pixel-level regularization for density control. These methods are regarded

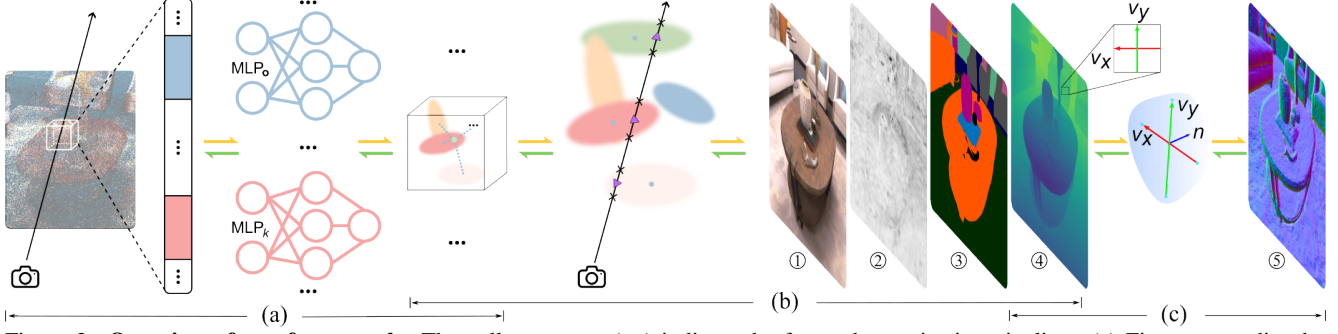


Figure 2. **Overview of our framework.** The yellow arrows ($\rightarrow$) indicate the forward rasterization pipeline. (a) First, we predict the semantic logits $\mathbf{o}$, contribution k , and other 3DGS attributes (i.e., position μ , rotation $\mathbf{q}$, scale $\mathbf{s}$, opacity α , SHs $\mathbf{h}$) in the anchor set through MLPs. For each anchor, we predict M raw 3DGS data available for rendering. (b) Then, during the rasterization stage, for each pixel, we use ray tracing to calculate its intersection ($\times$) with the Gaussian ellipsoid, which is closely related to the rotation and scale attributes. We replace the depth of the Gaussian center point ($\bullet$) with the depth of the midpoint ($\blacktriangle$) between the two intersection points. After splatting rasterization, we obtain RGB (①), gradient factor map (②), semantic (③), and depth (④). (c) To obtain the surface normal (⑤), we back-project the depth into three-dimensional space and then use a differential method to obtain the normals in the world coordinate system, ultimately producing the normal rendering map. The final rendered results are compared with their respective ground truths to calculate the loss. The green arrows ($\leftarrow$) indicate the direction of gradient propagation through a CUDA-accelerated analytical solution.

as post-processing techniques and lack integration within a unified framework.

Geometry-Aware Rendering. Geometry-aware approaches enhance reconstruction accuracy by incorporating surfel [9, 14, 17, 23, 62], normals [51, 62], and relighting [13, 65, 72] capabilities, enabling more physically realistic renderings and interactions. Some studies [47, 51, 58, 69] attempt to achieve accurate geometry within the NeRF framework, and while significant progress has been made, these methods still suffer from inherent limitations that are difficult to overcome with implicit representations. 3DGS [26, 41] does not inherently excel in geometric reconstruction either, as its Gaussian primitives are independent of each other, making precise geometry recovery an open challenge. Several works unitize 3DGS or 2DGS to recover depth [26, 41, 57, 62], surfel [17, 23, 62] and meshes [9, 14, 17], including normal estimation [62] and relighting [13, 65, 72] techniques, which can effectively reconstruct smooth object surfaces. Nevertheless, these approaches remain largely limited to small-scale objects. When extended to large environments such as outdoor or urban scenes [44, 64], the required computational time remains substantial. Some methods attempt to incorporate SDF-based surfel constraints [36, 60, 69, 70, 75], inverse rendering [52, 71], or restrictions [25, 34] on Gaussian properties. However, these approaches lack rigorous analytical derivation for geometric modeling, employ relatively limited quantitative metrics, and demonstrate constrained generalizability across diverse scenarios. Consequently, the improvements achieve thus far remain marginal in terms of geometric fidelity.

Semantic Rendering Frameworks. Several works attempt to integrate semantic information into both NeRF [6, 80] and 3DGS [15, 48], including techniques like Open-

Vocabulary Segmentation [10, 29, 76] and Language-Driven Semantic Segmentation [32, 37, 42, 49], with promising results. Embedding semantic information facilitates downstream tasks such as scene understanding [22, 81], navigation [24, 31], and scene editing [11, 73]. For instance, SGS-SLAM [33] directly uses semantic ground truth as supervision signal and rendering output in 3DGS, though it fails to formulate a unified 3DGS map representation. Feature-GS [82] extracts semantic features from the environment using 3DGS, and SEGGS-SLAM [63] enhances structural awareness with appearance embeddings. Nevertheless, such approaches exhibit limitations in reconstructing accurate environmental geometry and fail to achieve consistent reconstruction of scene geometry and semantics.

3. Methodology

The methodology section is organized as follows. Firstly, we define our map representation and objective of this work (Sec. 3.1). Then, the subsequent sections detail the key components of our framework: geometry-aware depth rasterization and normal estimation (Sec. 3.2); the novel gradient factor and differentiable pruning strategy (Sec. 3.3); and finally, the overall multimodal rendering pipeline, loss functions, and implementation details (Sec. 3.4). The brief overview of our framework is illustrated in Fig. 2.

3.1. Unified Map Representation and Objectives

Unified Map Representation. In our framework, each Gaussian $\mathcal{G}$ is defined by position $\mu \in \mathbb{R}^3$, quaternion $\mathbf{q} \in \mathbb{R}^4$, scale $\mathbf{s} \in \mathbb{R}^3$, opacity $\alpha \in \mathbb{R}$, and Spherical Harmonics (SHs) $\mathbf{h} \in \mathbb{R}^{C_h}$ (C_h is the pre-defined SHs number per color channel), semantic logits (encode the probability distribution across C_o semantic categories) $\mathbf{o} \in \mathbb{R}^{C_o}$ and

gradient factor $k \in \mathbb{R}$:

$$\mathcal{G} = \{\boldsymbol{\mu}, \mathbf{q}, \mathbf{s}, \alpha, \mathbf{h}, \mathbf{o}, k\}. \quad (1)$$

Objective. The objective of this framework is to render RGB color image, geometric depth, surface normals, semantic logits, and a gradient factor map (used in differentiable pruning) *simultaneously* within a *unified* rasterization pipeline. These rendered outputs are then compared against their corresponding 2D ground truth to compute respective loss, which guide the optimization of the 3D Gaussian scene through a differentiable backward process. To accelerate convergence and enhance computational efficiency, we derive the relevant *analytical solutions* and implement them using *CUDA-accelerated* gradient propagation. Finally, we can obtain a geometry consistent photo-realistic 3DGS map representation.

3.2. Geometry-Aware Rasterization

Ray-Gaussian Primitive Intersection Formulation. We define an ellipsoid $\mathcal{E}$ associated with Gaussian $\mathcal{G}$ by its center $\boldsymbol{\mu}$, rotation matrix $\mathbf{R}$ (converted from quaternion $\mathbf{q}$) and diagonal scaling matrix $\mathbf{S} = \text{diag}(s_x, s_y, s_z)$, and this ellipsoid consists of all 3D points $\mathbf{x}$ on its surface that satisfy the following equation:

$$(\mathbf{S}^{-1}\mathbf{R}^{-1}(\mathbf{x} - \boldsymbol{\mu}))^\top (\mathbf{S}^{-1}\mathbf{R}^{-1}(\mathbf{x} - \boldsymbol{\mu})) = 1. \quad (2)$$

For each pixel $p = (u, v)$, the corresponding camera ray $\mathbf{r}(t) = \mathbf{t}_{\text{cam}}^{\text{world}} + t\mathbf{d}$ is calculated through a unprojection process:

$$\mathbf{r}(t) = \mathbf{t}_{\text{cam}}^{\text{world}} + \text{normalize} \left(\mathbf{R}_{\text{cam}}^{\text{world}} \begin{bmatrix} \frac{u - c_x}{f_x} \\ \frac{v - c_y}{f_y} \\ 1 \end{bmatrix} \right), \quad (3)$$

where $\mathbf{t}_{\text{cam}}^{\text{world}}$ and $\mathbf{R}_{\text{cam}}^{\text{world}}$ are the translation vector and rotation matrix that transform a point in the camera frame to the world frame, respectively. The focal lengths f_x, f_y and principal point c_x, c_y are camera parameters.

To find the intersection between the ray $\mathbf{r}(t)$ and the ellipsoid $\mathcal{E}$, we transform the ray $\mathbf{r}(t)$ into the *local* (l) *scaled* (s) coordinate system of $\mathcal{E}$:

$$\begin{aligned} \mathbf{v}_s &= \mathbf{S}^{-1}\mathbf{v}_l, & \mathbf{v}_l &= \mathbf{R}^{-1}(\mathbf{t}_{\text{cam}}^{\text{world}} - \boldsymbol{\mu}), \\ \mathbf{d}_s &= \mathbf{S}^{-1}\mathbf{d}_l, & \mathbf{d}_l &= \mathbf{R}^{-1}\mathbf{d}. \end{aligned} \quad (4)$$

The intersection problem is then reduced to solving for t in the unit sphere equation:

$$\|\mathbf{v}_s + t\mathbf{d}_s\|^2 = 1. \quad (5)$$

The solution to this equation can be treated as that of a quadratic equation: for a light ray intersecting an ellipsoid, there are two valid solutions (t_1, t_2) for two intersection points, $t_1 = t_2$ for one point, and no valid solutions for no intersection. Details are in the supplementary materials.

Depth Rasterization. We take the *midpoint* $t_{\text{mid}} = (t_1 + t_2)/2$ of the two valid solutions (t_1, t_2) as the intersection

results. The replaced depth d of corresponding 3D primitive in camera coordinate can be computed by

$$\mathbf{P}_{\text{cam}} = \mathbf{R}_{\text{world}}^{\text{cam}}(\mathbf{t}_{\text{cam}}^{\text{world}} + t_{\text{mid}}\mathbf{d}), \quad d = \mathbf{P}_{\text{cam}, z}. \quad (6)$$

The final depth value $\tilde{D}(u, v)$ for pixel $p = (u, v)$ is obtained by alpha-rendering the depths d of all overlapping Gaussians $\mathcal{N}$:

$$\tilde{D}(u, v) = \sum_{i \in \mathcal{N}} \alpha_i d_i \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (7)$$

The above method allows to directly incorporate the geometric information (rotation $\mathbf{q}$ and scale $\mathbf{s}$ of the Gaussian $\mathcal{G}$) into the rasterization pipeline. This enables us to compute the gradients of the loss function with respect to the rotation $\mathbf{q}$ and scale $\mathbf{s}$, thereby effectively optimizing the Gaussian $\mathcal{G}$. Related gradient backpropagation derivations are provided in the supplementary materials.

Algorithm 1: Surface Normal Estimation by Depth

```

1 Input: Depth  $\tilde{D}(u, v)$ , STEP_1, STEP_2,  $\lambda$ 
2 Output: Normal  $\tilde{\mathbf{N}}(u, v)$ 
3 for each pixel  $p = (u, v)$  in parallel do
4   1. Backproject to World Coordinate
5   2. Compute Multi-Scale Normals
6     for  $g_i \in \{\text{STEP\_1}, \text{STEP\_2}\}$  do
7       Finite differences of neighboring points:
8          $\mathbf{v}_{x,i} = \mathbf{P}(u + g_i, v) - \mathbf{P}(u - g_i, v)$ ,
9          $\mathbf{v}_{y,i} = \mathbf{P}(u, v + g_i) - \mathbf{P}(u, v - g_i)$ ,
10        Estimate normal:  $\mathbf{n}_i = \mathbf{v}_{x,i} \times \mathbf{v}_{y,i}$ 
11   3. Fuse and Normalize
12     Ensure consistency:  $\mathbf{n}_2 \leftarrow \mathbf{n}_2 \cdot \text{sign}(\mathbf{n}_1 \cdot \mathbf{n}_2)$ 
13     Fuse estimates:  $\mathbf{n}_{\text{fused}} = \lambda \mathbf{n}_1 + (1 - \lambda) \mathbf{n}_2$ 
14     Normalize:  $\tilde{\mathbf{N}}(u, v) = \mathbf{n}_{\text{fused}} / \|\mathbf{n}_{\text{fused}}\|$ 
15   4. Orient Towards View Direction
```

Normal Rasterization from Depth. As mentioned previously, it is challenging to obtain sufficiently smooth and accurate surface normals through rasterization-based rendering due to the strong directional information inherent in normals. Therefore, in this paper, we compute normals indirectly from depth rasterization result. The focus lies on deriving the gradient propagation process from the normal loss back to the depth.

Our approach begins by unprojecting the depth map into world coordinate using the camera parameters. We then estimate the normals in world coordinate through a multi-step finite difference scheme, with step sizes denoted as STEP_1, STEP_2. The overall procedure is summarized in Alg. 1. Further details regarding the backprojection of depth pixels and orientation toward the view direction will be provided in the supplementary material. Additionally, the derivation of backpropagation from the normal loss to depth will also be included in the supplementary material.

3.3. Differentiable Pruning

Gradient Factor Rasterization and Backpropagation. We define a new attribute k in (1) for each Gaussian $\mathcal{G}$ to measure its gradient level during the rendering process. Alongside the rendering of RGB image, we can render a 2D gradient factor map $\tilde{\mathbf{K}}$ by alpha-blending:

$$\tilde{\mathbf{K}}(u, v) = \sum_{i \in \mathcal{N}} \alpha_i k_i \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (8)$$

Our objective is to drive each Gaussian’s k toward 1. Experimental results show this state corresponds to the optimal optimization effect of Gaussians. To enforce this, we minimize the L1-loss between $\tilde{\mathbf{K}}$ and an all-ones matrix $\mathbf{1}$:

$$\mathcal{L}_K = \|\tilde{\mathbf{K}} - \mathbf{1}\|_1. \quad (9)$$

Detailed analysis on why $k = 1$ is optimal and gradient backpropagation are provided in the supplementary materials.

Pruning. After a fixed interval of iterations K_p ($K_p = 3000$ in our experiments), our framework directly eliminates Gaussians with abnormal gradient factor $\|k - 1\|_1 > \mathcal{T}_k$, effectively pruning the set and reducing the number of Gaussians to enhance overall efficiency. After pruning operation, the k value of all Gaussians will be reset to 0.9 to maintain the sustainability of updates. Please refer to our code for details.

3.4. Multimodal Rendering Framework

Architecture Overview. Our system is built upon a modified Scaffold-GS [41] architecture. For each anchor point, we maintain a foundational 3D Gaussian representation (1). During every training iteration, multiple MLPs are used to predict attribute offsets for each anchor.

Attribute Prediction. For semantic logits $\mathbf{o} \in \mathbb{R}^{C_o}$ and gradient factor $k \in \mathbb{R}$ of a Gaussian $\mathcal{G}$, we replicate (denoted as **Repeat**) M times and modulate by the output of an MLP branch to obtain Gaussian features for rasterization:

$$\begin{bmatrix} \mathbf{o}_{\text{vis}} \\ k_{\text{vis}} \\ \dots \end{bmatrix} = \text{Repeat} \left(\begin{bmatrix} \mathbf{o}_{\text{ah}} \\ k_{\text{ah}} \\ \dots \end{bmatrix}, M \right) \otimes \sigma \begin{bmatrix} \text{MLP}_{\mathbf{o}}(\mathbf{f}_{\text{ah}}) \\ \text{MLP}_k(\mathbf{f}_{\text{ah}}) \\ \dots \end{bmatrix}, \quad (10)$$

where $\mathbf{o}_{\text{ah}} \in \mathbb{R}^{C_o}$ is the base semantic logits of the anchor, $k_{\text{ah}} \in \mathbb{R}$ is the base gradient factor of the anchor. M is the replication factor. $\mathbf{f}_{\text{ah}}$ represents the input feature for the anchor. $\text{MLP}_{\mathbf{o}} : \mathbb{R}^{Fdim} \rightarrow \mathbb{R}^{C_o}$ denotes the MLP branch for semantics. $\text{MLP}_k : \mathbb{R}^{Fdim} \rightarrow \mathbb{R}$ denotes the MLP branch for gradient factor. σ is the sigmoid activation function. $\otimes$ denotes element-wise multiplication. $Fdim$ is the feature dimension. The resulting modulated attributes are then combined with other predicted properties (i.e., position μ , rotation $\mathbf{q}$, scale $\mathbf{s}$, opacity α , SHs $\mathbf{h}$, denoted as ...) to form the final 3D Gaussians ready for projection and rendering.

Semantic Rendering and Backpropagation. We render a 2D semantic logits $\tilde{\mathbf{O}} \in \mathbb{R}^{H \times W \times C_o}$ along with rasteriza-

Algorithm 2: Multimodal Forward Rasterization Pipeline

Input: $\mathcal{G} = \{\mu, \mathbf{q}, \mathbf{s}, \alpha, \mathbf{h}, \mathbf{o}, \mathbf{k}\}$, Image dimensions W, H
Output: $\tilde{\mathbf{C}} \in \mathbb{R}^{W \times H \times 3}$, $\tilde{\mathbf{O}} \in \mathbb{R}^{W \times H \times C}$, $\tilde{\mathbf{D}} \in \mathbb{R}^{W \times H}$, $\tilde{\mathbf{K}} \in \mathbb{R}^{W \times H}$

```

1 for each pixel = (u, v) do in parallel
  // Compute Ray of pixel
2 computeRay (3);
3 Initialize accumulators:  $\mathbf{C} \leftarrow \mathbf{0}$ ,  $\mathbf{O} \leftarrow \mathbf{0}$ ,
   $D \leftarrow 0$ ,  $K \leftarrow 0$ ,  $T \leftarrow 0$ ;
4 for each Gaussian  $i$  intersecting pixel (u, v) do
5   ...
6    $\mathbf{h}_i, \mathbf{o}_i, \alpha_i \leftarrow$  Gaussian  $i$  parameters;
  // Ray-Ellipsoid Intersection
7 rayEllipsoidIntersection (5);
  // Use Midpoint for Depth
8  $d_i \leftarrow$  obtainDepth (6);
  // Alpha-Blending for All
  Attributes
9   for each  $ch$  in  $C_h$  do
10     $\mathbf{C}_{[ch]} \leftarrow \mathbf{C} + \mathbf{h}_i \cdot \alpha_i \cdot T$ ;
11   for each  $ch$  in  $C_o$  do
12     $\mathbf{O}_{[ch]} \leftarrow \mathbf{O} + \mathbf{o}_i \cdot \alpha_i \cdot T$ ;
13    $D \leftarrow D + d_i \cdot \alpha_i \cdot T$ ;
14    $K \leftarrow K + t_i \cdot \alpha_i \cdot T$ ;
15   ...
  // Pixel-Wise Assignment
16  $\tilde{\mathbf{C}}[x, y] \leftarrow \mathbf{C}$ ;  $\tilde{\mathbf{O}}[x, y] \leftarrow \mathbf{O}$ ;
17  $\tilde{\mathbf{D}}[x, y] \leftarrow D$ ;  $\tilde{\mathbf{K}}[x, y] \leftarrow K$ ;
18 Save  $T_{\text{final}} = T$ ;
```

tion pipeline. The forward process for the semantic logits in pixel $p = (u, v)$ and channel ch is defined as

$$\tilde{\mathbf{O}}_{[ch]}(u, v) = \sum_{i \in \mathcal{N}} \alpha_i \mathbf{o}_{i[ch]} \prod_{j=1}^{i-1} (1 - \alpha_j). \quad (11)$$

Then the rendered semantic logits $\tilde{\mathbf{O}}$ will be compared to its ground truth $\mathbf{O}^{\text{gt}}$ using Cross-Entropy Loss, resulting in semantic loss $\mathcal{L}_{\text{seg}}$. Let $\frac{\partial \mathcal{L}_{\text{seg}}}{\partial \tilde{\mathbf{O}}_{[ch]}}$ be the gradient of the loss with respect to the rendered ch -th channel of a specific pixel. The chain rule for the gradient of $\mathcal{L}_{\text{seg}}$ with respect to a specific semantic channel $\mathbf{o}_{i[ch]}$ of a Gaussian $\mathcal{G}$ is given by

$$\frac{\partial \mathcal{L}_{\text{seg}}}{\partial \mathbf{o}_{i[ch]}} = \frac{\partial \mathcal{L}_{\text{seg}}}{\partial \tilde{\mathbf{O}}_{[ch]}} \cdot \frac{\partial \tilde{\mathbf{O}}_{[ch]}}{\partial \mathbf{o}_{i[ch]}}, \quad (12)$$

where $\frac{\partial \tilde{\mathbf{O}}_{[ch]}}{\partial \mathbf{o}_{i[ch]}}$ can be derived from (11).

Tile-based Forward Rasterization. We extend the original 3DGS’s tile-based rendering pipeline [26]. During rasterization, for each pixel $p = (u, v)$, we calculate the direction

Table 1. Quantitative comparison experiments on the Replica dataset show that our method significantly outperforms other baselines.

Dataset	OOS1			OIS2			RIS2			R2S2		
Method Metrics	Abs.Rel ↓	RMSE ↓	CS ↑	Abs.Rel ↓	RMSE ↓	CS ↑	Abs.Rel ↓	RMSE ↓	CS ↑	Abs.Rel ↓	RMSE ↓	CS ↑
GeoGaussian [34]	0.5828	0.8006	0.3046	2.8959	1.1092	0.1861	0.1336	0.3236	0.5700	0.1082	0.2592	0.5998
3DGS [26]	1.2231	0.9183	0.2304	0.8711	0.6238	0.3201	0.2134	0.4092	0.4963	1.4422	1.0972	0.2905
Scaffold-GS [41]	2.2857	1.2102	0.2195	1.4609	0.9413	0.2854	4.0575	1.4786	0.1555	-	-	0.1286
2DGS [18]	0.0089	0.0679	0.8775	0.0172	0.0399	0.9299	0.0451	0.1400	0.9112	0.0735	0.1759	0.8480
Normal-GS [62]	0.0074	0.0657	0.8755	0.0154	0.0344	0.9234	0.0243	0.0766	0.9234	0.0192	0.0589	0.8987
Ours	0.0062	0.0310	0.8880	0.0015	0.0073	0.9686	0.0136	0.0467	0.9545	0.0144	0.0465	0.9076

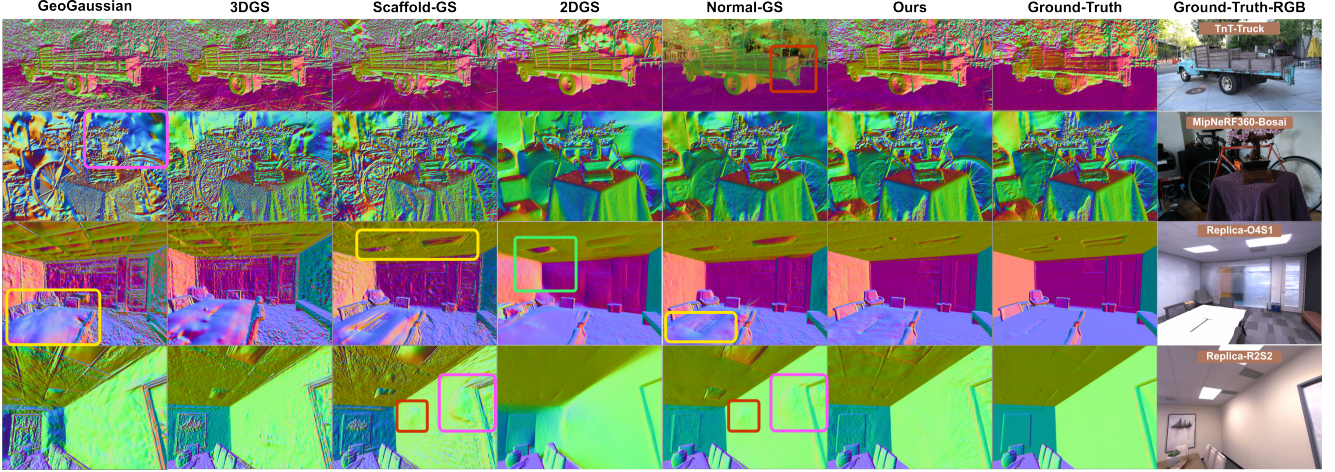


Figure 3. Across various dataset tests, our method demonstrates impressive results in geometric reconstruction. Even in textureless areas and edge regions, it remains capable of recovering fine geometric details.

$\mathbf{d}$ of the camera ray $r(t)$ and then compute the intersection t_1/t_2 with the Gaussian projected onto this pixel. Finally, we obtain the corresponding attributes using alpha blending rendering. The multimodal forward rasterization pipeline is outlined in Alg. 2. The corresponding backpropagation pipeline is described in the supplementary materials.

Loss Function. For the RGB image loss, we employ a combination of L1 loss $\mathcal{L}_{l1}$ and SSIM [61] loss $\mathcal{L}_{ssim}$. For the normal loss $\mathcal{L}_{normal}$, we use cosine similarity, which is calculated as

$$\mathcal{L}_{normal} = 1 - \text{Cosine Similarity}(\tilde{\mathbf{N}}, \tilde{\mathbf{N}}_{gt}). \quad (13)$$

Incorporating the gradient factor loss $\mathcal{L}_K$ and semantic loss $\mathcal{L}_{seg}$, we construct the comprehensive loss function as

$$\mathcal{L}_{all} = \lambda_1 \mathcal{L}_{l1} + \lambda_2 \frac{|\mathcal{L}_{l1}| \mathcal{L}_{ssim}}{|\mathcal{L}_{ssim}|} + \lambda_3 \frac{|\mathcal{L}_{l1}| \mathcal{L}_{normal}}{|\mathcal{L}_{normal}|} + \lambda_4 \frac{|\mathcal{L}_{l1}| \mathcal{L}_{depth}}{|\mathcal{L}_{depth}|} + \lambda_5 \frac{|\mathcal{L}_{l1}| \mathcal{L}_{seg}}{|\mathcal{L}_{seg}|} + \lambda_6 \frac{|\mathcal{L}_{l1}| \mathcal{L}_K}{|\mathcal{L}_K|}. \quad (14)$$

where weights are applied as $\lambda_1 = 1, \lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = \lambda_6 = 0.1$.

Implementation Details. Our framework is implemented by CUDA using the LibTorch framework [7], incorporating CUDA code for Gaussian Splatting and trained on a server with a 2.60 GHz Intel(R) Xeon(R) Platinum 8358P CPU, 1T GB RAM, and a NVIDIA RTX 4090 24 GB GPU. In all

sequences, the hyperparameters used in our experiments are listed in the supplementary materials.

4. Experiments

In this section, we first introduce the experimental setup in Sec. 4.1, including datasets, baselines, and parameter settings, etc. Given that RGB rendering is not the central focus of our study, the corresponding comparison will be provided in the supplementary material accordingly. Then in Sec. 4.2, we evaluate the geometry rendering performance (e.g. depth and normal). In Sec. 4.3, we evaluate the semantic rendering performance. In Sec. 4.4, we provide the analysis of running time and GPU memory consumption of the framework. Sec. 4.5 shows ablation experiments of the proposed framework.

4.1. Experiment Setup

Datasets. We conduct comparative experiments on widely used datasets, including 7 scenes from Mip-NeRF [1], 2 scenes from Tanks & Temples [28] and 2 scenes from Deep Blending [16]. All of these datasets do not provide precise geometric and semantic ground truth. To address this, we use the results obtained from the depth rendering pipeline in Normal-GS [62] as geometric priors for supervision. At the same time, we utilize open vocabulary queries to pre-process semantic labels with OpenSeed [76]. For accurate quantitative comparison experiments, we use

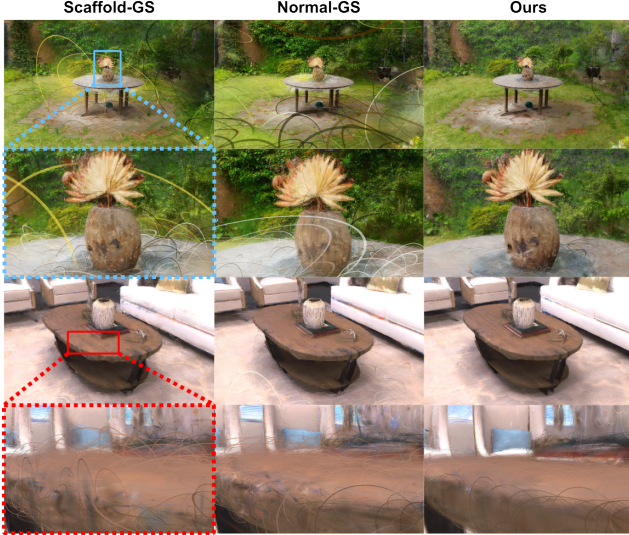


Figure 4. A qualitative demonstration showing that the Gaussian primitives reconstructed by our method are geometrically closer to the real surface.

the Replica [53] simulation dataset, which provides complete multimodal ground truth. We employ the pre-rendered Replica dataset from HiSplat [55], which includes 14 motion sequences across 7 scenes. The initial points in the Replica [53] dataset are obtained through mesh sampling, with a sampling interval of 5 in all sequences.

Baselines and Metrics. We compare our method with the existing SOTA NeRF-based dense framework MipNeRF [1], iNGP [45] and 3DGS framework [26], Scaffold-GS [41], GeoGaussian [34], 2DGS [18], Normal-GS [62], SemanticGaussian [15]. We evaluate the RGB rendering performance using PSNR, SSIM [61], and LPIPS [77]. The evaluation metrics for depth estimation we used are *Abs.Rel.* and RMSE. The evaluation metric we used for normal estimation is cosine similarity (CosSim), for semantic segmentation, we employed mean mIoU.

4.2. Geometry Rendering Evaluation

We conduct detailed qualitative and quantitative comparisons of depth and normal estimation against methods [26] [41] [23] [34] [62] [18] on the Replica [53] dataset. The comparison results are shown in Fig. 3 and Tab. 1. Normal-GS [62] and our method optimize the surface normals, and both qualitatively and quantitatively outperform other 3DGS frameworks. Our approach also exhibits superior geometric accuracy in some extreme scenarios, particularly in texture-less areas, surpassing even Normal-GS [62] and 2DGS [18]. More comparisons about geometry rendering can be found in supplementary materials.

We further compare the geometric impact after modifying our depth rendering method. Firstly, we complete the reconstruction process of a 3D Gaussian scene and then identify a detailed area in the reconstructed scene (as shown

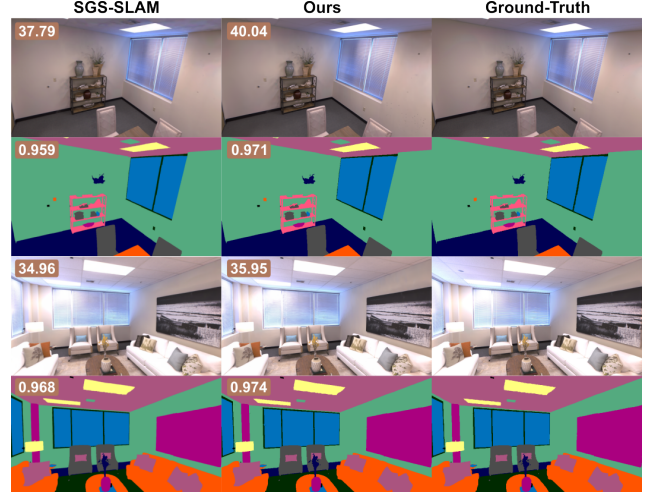


Figure 5. Quantitative comparison between our method and SGS-SLAM [33] in terms of RGB rendering and semantic segmentation clearly shows that our approach outperforms SGS-SLAM. The top-left corner of the image displays the PSNR value computed against the ground truth image.

Table 2. Quantitative geometric comparison between our method and other methods after sampling the reconstructed Gaussian model, compared to the initial points or mesh points.

Dataset	Method Metrics	MipNeRF-Garden				Replica-Room0			
		Radio ↓	Mean ↓	Std ↓	Hau-Dis ↓	Radio ↓	Mean ↓	Std ↓	Hau-Dis ↓
Scaffold-GS [41]		0.0390	0.0299	0.0383	0.592	0.0476	0.0120	0.0083	0.141
2DGS [18]		0.0607	0.0984	0.1272	1.091	0.0628	0.0519	0.0244	0.462
Normal-GS [62]		0.0388	0.0303	0.0377	0.936	0.0477	0.0120	0.0082	0.131
Ours		0.0341	0.0272	0.0298	0.405	0.0339	0.0137	0.0081	0.078

in Fig. 4). It is evident that our method, compared to others, results in optimized Gaussian primitives that tend to be closer to the actual surface of the object. This is because our method integrates rotation and scale attribute into the optimization process, thereby achieving geometric consistency. Additionally, we perform point sampling [34] of the Gaussian primitives within specific small regions, followed by a Z-score statistical comparison with the initial ground truth and a Hausdorff distance comparison, as shown in Tab. 2. It is evident that our method achieves consistent geometric surface reconstruction. The calculation method using the Z-score statistic is explained in the supplementary materials.

Table 3. Quantitative comparison with the SGS-SLAM [33] method in terms of RGB rendering, semantic rendering and FPS. TT represents training time (minutes). Best results are underlined.

Dataset	Method Metrics	All	Room0		Room2		Office2	
		TT ↓	PSNR ↑	FPS ↑	PSNR ↑	FPS ↑	PSNR ↑	FPS ↑
SGS-SLAM [33]		212.6	36.28	96.4	39.14	72.2	36.94	112.3
Ours		<u>32.2</u>	<u>36.66</u>	<u>184.2</u>	<u>40.06</u>	<u>234.4</u>	<u>38.16</u>	<u>229.3</u>

4.3. Semantic Rendering Evaluation

We compare our method with the SGS-SLAM [33] framework, which employs a task-specific Gaussian map representation and is supervised using semantic result images for semantic tasks, differing from our unified framework that

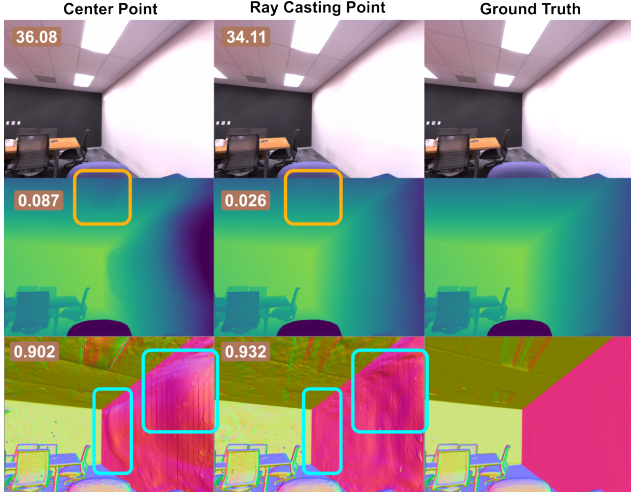


Figure 6. The use of a ray casting-based depth rendering approach significantly enhances the algorithm’s robustness in textureless regions. Furthermore, the Gaussian splats provide a tighter fit to the actual object surfaces.

predicts semantic logits. The comparison across multiple scenes is shown in Fig. 5 and Tab. 3. From the quantitative results, it can be observed that our method achieves superior performance in both image rendering quality (PSNR) and semantic segmentation quality (mIoU) within a very short training time. Thanks to our unified rendering architecture, the image rendering FPS is also fast. Additional results can be found in our supplementary video.

Table 4. Quantitative comparison results in terms of Gaussian splat count and rendering FPS.

Dataset Method Metrics	Mip-NeRF360 Count ↓ FPS ↑	Tanks & Temples Count ↓ FPS ↑	Deep Blending Count ↓ FPS ↑	Replica Count ↓ FPS ↑
GeoGaussian [34]	126021 440.9	94755 441.1	53457 436.2	100877 423.6
3DGS [26]	504167 128.7	421167 136.9	285394 162.4	543581 137.5
Scaffold-GS [41]	301839 240.1	247081 249.1	182304 324.0	235446 346.2
Normal-GS [62]	318064 180.3	226283 178.7	185243 250.7	204358 180.0
Ours	161081 169.8	119947 163.1	48970 296.7	169273 197.8

4.4. Runtime and Memory Cost Analysis

In Tab. 4, we present a comparison of the Count of Gaussians and rendering FPS against other methods. Our approach employs an effective differentiable pruning module, which removes a significant number of redundant Gaussians. As a result, the number of Gaussian splats in our reconstructed scenes is the smallest. However, since our method requires rendering multiple modalities simultaneously, the FPS experiences a certain degree of decrease. Nevertheless, this does not severely hinder our rendering speed. In some scenarios, our rendering speed remains superior even compared to Normal-GS [62].

4.5. Ablation Study

Geometry-Aware Rasterization. We compare the performance of depth rasterization that considers geometric information with that of depth rasterization using only the cen-

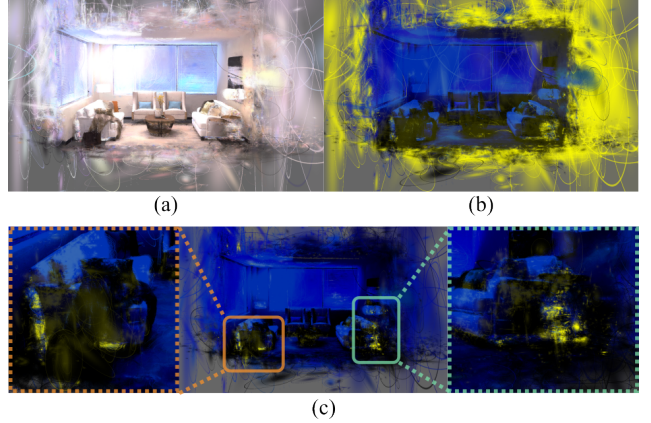


Figure 7. **Visualization of the distribution of the gradient factor.** To enhance the visualization of the gradient factor, the brightness of all Gaussians is uniformly minimized, resulting in blue regions as rendered by SuperSplat [8]. (a) Scene with original brightness. (b) Areas highlighted in yellow indicate regions where the gradient factor $k > 1.5$. (c) Areas highlighted in yellow indicate regions where $k < 0.5$.

ter points in terms of image quality. From the qualitative results presented in Fig. 6, it can be observed that incorporating set information forces the Gaussian spheres to align more closely with the actual surface of the objects. More quantitative ablation experiments on each modules can be found in our supplementary materials.

Differentiable Pruning. We conduct continuous optimization of the gradient factor without pruning any Gaussians to examine its distribution, as shown in Fig. 7. It can be observed that regions with anomalous gradient factor, which deviate significantly from the ideal value of 1, are typically areas lacking observational data or corresponding to occluded edges. In these regions, due to insufficient supervisory information from images, the gradients fail to converge to the desired value. Such areas represent environmental redundancy that can be removed to improve computational efficiency.

5. Conclusion

In this paper, we present **UniGS**, a *unified* and differentiable framework for high-fidelity multimodal 3D reconstruction using 3DGS. Our rasterization pipeline effectively renders photo-realistic RGB images, accurate depth maps, consistent surface normals, and semantic labels simultaneously. By implementing differentiable ray-ellipsoid intersection, we optimize rotation and scaling parameters with CUDA-accelerated analytic gradient backpropagation, ensuring geometric consistency in surface. We also introduce a learnable attribute for efficient pruning of minimally contributing Gaussians. Extensive experiments demonstrate state-of-the-art performance in terms of both reconstruction accuracy across all modalities and storage efficiency.

UniGS: Unified Geometry-Aware Gaussian Splatting for Multimodal Rendering

Supplementary Material

6. Methodology

In Sec. 6.1, we provide additional details on solving the intersection points between pixel rays and Gaussians. In Sec. 6.2, we focus on deriving the analytical solution for propagating depth gradients to the Gaussian parameters, specifically addressing the transition from depth to rotation, scale and positions. In Sec. 6.3, we include the implementation details that are omitted in Alg. 1. In Sec. 6.4, we develop the derivation of the gradient propagation from the cosine similarity loss of the normal estimation to the rendered depth map, enabling the normal gradients to flow back to depth gradient. This connection allows the normal gradients to be backpropagated further, thus linking the normal consistency loss directly to the Gaussian parameters. In Sec. 6.5, we expand the discussion on differentiable pruning, with a particular focus on elucidating why $k = 1$ serves as the optimization objective. Finally, in Sec. 6.6, we provide the implementation process for the backward rasterization gradient propagation corresponding to Alg. 2.

6.1. Ray-Eclipse Intersection Problem

The ray-eclipse intersection problem in (5) aim to solve for t in the equation of a unit sphere

$$\|\mathbf{v}_s + t\mathbf{d}_s\|^2 = 1, \quad (15)$$

which simplifies to the quadratic equation:

$$at^2 + bt + c = 0, \quad (16)$$

where the coefficients are defined as:

$$\begin{aligned} a &= \mathbf{d}_s \cdot \mathbf{d}_s, \\ b &= 2\mathbf{v}_s \cdot \mathbf{d}_s, \\ c &= \mathbf{v}_s \cdot \mathbf{v}_s - 1. \end{aligned} \quad (17)$$

The solutions are given by $t = \frac{-b \pm \sqrt{\Delta}}{2a}$, where $\Delta = b^2 - 4ac$. This is a simple quadratic equation solving problem.

6.2. Backward Propagation for Depth Rendering

The gradient of the loss $\mathcal{L}_{depth}$ with respect to the a single Gaussian parameters $\theta \in \{\mu, s, \mathbf{R}\}$ is propagated by

$$\frac{\partial \mathcal{L}_{depth}}{\partial \theta} = \frac{\partial \mathcal{L}_{depth}}{\partial \tilde{\mathbf{D}}} \cdot \frac{\partial \tilde{\mathbf{D}}}{\partial d_i} \cdot \frac{\partial d_i}{\partial \theta}, \quad (18)$$

where $\frac{\partial \tilde{\mathbf{D}}}{\partial d_i}$ can be calculated from (7). The primary challenge is to compute the gradients of d_i with respect to the Gaussian parameters μ , s , and $\mathbf{R}$, which requires backpropagating through the ray-ellipsoid intersection. The gradient

flow is structured as

$$\begin{aligned} \frac{\partial d_i}{\partial \theta} &= \frac{\partial d_i}{\partial t_{mid}} \cdot \frac{\partial t_{mid}}{\partial (a, b, c)} \cdot \frac{\partial (a, b, c)}{\partial (\mathbf{v}_s, \mathbf{d}_s)} \\ &\quad \cdot \frac{\partial (\mathbf{v}_s, \mathbf{d}_s)}{\partial (\mathbf{v}_l, \mathbf{d}_l)} \cdot \frac{\partial (\mathbf{v}_l, \mathbf{d}_l)}{\partial \theta}. \end{aligned} \quad (19)$$

The key steps in this chain will be derived below.

Gradient of Depth w.r.t. t_{mid} . The depth d_i of the intersection point is a function of the midpoint ray parameter t_{mid} . The gradient is computed as the dot product of the ray direction $\mathbf{d}$ with the z -axis of the camera view matrix $\mathbf{R}_{world}^{cam}$:

$$\begin{aligned} \frac{\partial d_i}{\partial t_{mid}} &= \mathbf{R}_{world}^{cam}[2, 0] \cdot \mathbf{d} \rightarrow x + \mathbf{R}_{world}^{cam}[2, 1] \cdot \mathbf{d} \rightarrow y \\ &\quad + \mathbf{R}_{world}^{cam}[2, 2] \cdot \mathbf{d} \rightarrow z. \end{aligned} \quad (20)$$

Gradient of t_{mid} w.r.t. Quadratic Coefficients. The midpoint parameter $t_{mid} = (t_1 + t_2)/2 = -b/(2a)$ is a function of the quadratic coefficients a and b . Its derivatives are

$$\frac{\partial t_{mid}}{\partial a} = \frac{b}{2a^2}, \quad \frac{\partial t_{mid}}{\partial b} = -\frac{1}{2a}, \quad \frac{\partial t_{mid}}{\partial c} = 0. \quad (21)$$

Gradients of Coefficients w.r.t. Local Coordinates. The coefficients a , b , and c are functions of the scaled local ray origin $\mathbf{v}_s$ and direction $\mathbf{d}_s$ from (17). Their gradients are

$$\begin{aligned} \frac{\partial a}{\partial \mathbf{d}_s} &= 2\mathbf{d}_s, \quad \frac{\partial b}{\partial \mathbf{v}_s} = 2\mathbf{d}_s, \\ \frac{\partial b}{\partial \mathbf{d}_s} &= 2\mathbf{v}_s, \quad \frac{\partial c}{\partial \mathbf{v}_s} = 2\mathbf{v}_s. \end{aligned} \quad (22)$$

Applying the chain rule through t_{mid} and the coefficients, the gradients w.r.t. the *local* (l) *scaled* (s) coordinates are:

$$\begin{aligned} \frac{\partial t_{mid}}{\partial \mathbf{v}_s} &= \frac{\partial t_{mid}}{\partial b} \cdot \frac{\partial b}{\partial \mathbf{v}_s} + \frac{\partial t_{mid}}{\partial c} \cdot \frac{\partial c}{\partial \mathbf{v}_s} \\ &= -\frac{1}{a} \cdot \mathbf{d}_s, \\ \frac{\partial t_{mid}}{\partial \mathbf{d}_s} &= \frac{\partial t_{mid}}{\partial a} \cdot \frac{\partial a}{\partial \mathbf{d}_s} + \frac{\partial t_{mid}}{\partial b} \cdot \frac{\partial b}{\partial \mathbf{d}_s} \\ &= \frac{b}{a^2} \cdot \mathbf{d}_s - \frac{1}{a} \cdot \mathbf{v}_s. \end{aligned} \quad (23)$$

Gradients w.r.t. Scale s . The *local* (l) *scaled* (s) coordinates are defined as $\mathbf{v}_s = \mathbf{v}_l \oslash \mathbf{s}$ and $\mathbf{d}_s = \mathbf{d}_l \oslash \mathbf{s}$, where $\oslash$ denotes element-wise division. The gradients with respect to the scale s are computed as:

$$\frac{\partial (\mathbf{v}_s, \mathbf{d}_s)}{\partial \mathbf{s}} = -\mathbf{v}_{l,k} \oslash \mathbf{s} - \mathbf{d}_{l,k} \oslash \mathbf{s}. \quad (24)$$

Gradients w.r.t. Position μ and Rotation $\mathbf{R}$. The local coordinates before scaling are obtained by rotating the world-space vectors into the Gaussian's local frame: $\mathbf{v}_l = \mathbf{R}^\top \mathbf{v}$ and $\mathbf{d}_l = \mathbf{R}^\top \mathbf{d}$, where $\mathbf{v} = \mathbf{o} - \mu$. The gradient with

respect to the position μ is

$$\frac{\partial d_i}{\partial \mu} = -\mathbf{R} \left(\frac{\partial d_i}{\partial \mathbf{v}_l} \odot \mathbf{s} \right). \quad (25)$$

The gradient with respect to the rotation matrix $\mathbf{R}$ is accumulated from both $\mathbf{v}$ and $\mathbf{d}$:

$$\frac{\partial d_i}{\partial \mathbf{R}} = \left(\frac{\partial d_i}{\partial \mathbf{v}_l} \odot \mathbf{s} \right) \mathbf{v}^\top + \left(\frac{\partial d_i}{\partial \mathbf{d}_l} \odot \mathbf{s} \right) \mathbf{d}^\top. \quad (26)$$

Finally, the gradient $\frac{\partial d_i}{\partial \mathbf{R}}$ is converted to a gradient $\frac{\partial \mathbf{R}}{\partial \mathbf{q}}$ with respect to the quaternion $\mathbf{q}$ using the standard adjoint-sensitive conversion, ensuring the constraint of staying on the S^3 manifold through tangent space projection.

6.3. Normal Estimation from Depth

Following the rendering of the depth map $\tilde{\mathbf{D}}$, we estimate a corresponding surface normal map $\tilde{\mathbf{N}}$ in a post-processing step. In this supplementary section, we will elaborate on two key operations of Alg. 1: backprojecting depth pixels and orienting towards the view direction.

Depth Backprojection to 3D Points. Each pixel $p = (u, v)$ with depth value $\tilde{\mathbf{D}}(u, v)$ is backprojected to a 3D point $\mathbf{P}_{\text{cam}}$ in the camera coordinate system using the intrinsic camera parameters:

$$\mathbf{P}_{\text{world}} = \begin{bmatrix} \mathbf{R}_{\text{cam}}^{\text{world}} & \mathbf{t}_{\text{cam}}^{\text{world}} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \frac{(u-c_x)}{f_x} \cdot \tilde{\mathbf{D}}(u, v) \\ \frac{(v-c_y)}{f_y} \cdot \tilde{\mathbf{D}}(u, v) \\ \tilde{\mathbf{D}}(u, v) \\ 1 \end{bmatrix}, \quad (27)$$

where $c_x = W/2$, $c_y = H/2$ are the coordinates of the principal point, and f_x, f_y are the focal lengths.

View Direction Correction. Finally, the normal's orientation is corrected to face the camera. The normal is flipped if it points away from the camera, determined by the sign of the dot product between the normalized view direction $\mathbf{d}_{\text{view}}$ and the normal:

$$\tilde{\mathbf{N}}(u, v) \leftarrow \begin{cases} \tilde{\mathbf{N}}(u, v), & \text{if } \tilde{\mathbf{N}}(u, v) \cdot \mathbf{d}_{\text{view}} \leq 0 \\ -\tilde{\mathbf{N}}(u, v), & \text{otherwise} \end{cases}, \quad (28)$$

where $\mathbf{d}_{\text{view}} = \frac{\mathbf{C} - \mathbf{P}_{\text{world}}}{\|\mathbf{C} - \mathbf{P}_{\text{world}}\|}$ is the normalized vector from the 3D point $\mathbf{P}_{\text{world}}$ to the camera center $\mathbf{C}$. This ensures all normals adhere to a consistent front-facing convention.

6.4. Backward Propagation from Normals to Depth

To enable optimization via losses defined on the rendered normal map $\tilde{\mathbf{N}}$, the gradients must be backpropagated through the normal estimation pipeline to the depth map $\tilde{\mathbf{D}}$ and subsequently to the underlying 3D Gaussian parameters. Overall gradient backpropagation pipeline is shown in Alg. 3. The gradient of the loss $\mathcal{L}_{\text{normal}}$ with respect to

Algorithm 3: Backward Propagation from Normals to Depth

```

1 Input:  $\frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{N}}}$ , STEP_1, STEP_2,  $\lambda$ 
2 Output:  $\frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{D}}}$ 
3 for each pixel  $p = (u, v)$  in parallel do
4   1. View-Facing Orientation
     Flip (32)  $\leftarrow \frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{N}}(u, v)}$ 
5   2. Grad Normalization (33)
6   3. Grad Fusion (34)  $\leftarrow \lambda$ 
7   4. Compute Multi-Scale Gradient
     for STEP_X  $\in \{\text{STEP\_1}, \text{STEP\_2}\}$  do
9     Gradient Atom Addition (35) (36)
10  5. Backprojection Gradient (37)
11   $\frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{D}}(u, v)} \leftarrow$  (29)

```

the depth map $\tilde{\mathbf{D}}$ is propagated by

$$\frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{D}}(u, v)} = \frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{N}}(u, v)} \cdot \frac{\partial \tilde{\mathbf{N}}(u, v)}{\partial \mathbf{n}_{\text{fused}}^{(\text{orient})}} \cdot \frac{\partial \mathbf{n}_{\text{fused}}^{(\text{orient})}}{\partial \mathbf{n}_{\text{fused}}} \cdot \frac{\partial \mathbf{n}_{\text{fused}}}{\partial (\mathbf{n}_1, \mathbf{n}_2)} \cdot \left(\frac{\partial \mathbf{n}_1}{\partial \mathbf{P}_{\text{world}}^{n_1}} + \frac{\partial \mathbf{n}_2}{\partial \mathbf{P}_{\text{world}}^{n_2}} \right) \cdot \frac{\partial \mathbf{P}_{\text{world}}}{\partial \tilde{\mathbf{D}}(u, v)}, \quad (29)$$

where $\mathbf{P}_{\text{world}}^{n_1}$ and $\mathbf{P}_{\text{world}}^{n_2}$ represent the sets of 3D points in the neighborhoods used to compute $\mathbf{n}_1$ and $\mathbf{n}_2$, respectively. **Cosine Similarity w.r.t. Normal.** For encouraging geometric consistency (13) is the negative cosine similarity between the predicted normal $\tilde{\mathbf{N}}$ and a target normal $\tilde{\mathbf{N}}_{\text{target}}$:

$$\mathcal{L}_{\text{normal}} = 1 - \frac{1}{|\Omega|} \sum_{(u, v) \in \Omega} \tilde{\mathbf{N}}(u, v) \cdot \tilde{\mathbf{N}}_{\text{target}}(u, v), \quad (30)$$

where Ω denotes the set of pixels. The gradient of this loss with respect to the predicted normal at a specific pixel is straightforward:

$$\frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{N}}(u, v)} = -\frac{1}{|\Omega|} \tilde{\mathbf{N}}_{\text{target}}(u, v). \quad (31)$$

This gradient $\frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{N}}}$ is the seed that initiates the backward pass through the normal estimation pipeline.

View-facing Orientation Flip. If the normal is flipped in the forward pass to face the viewer (defined by (28)), the incoming gradient is correspondingly flipped:

$$\frac{\partial \mathcal{L}_{\text{normal}}}{\partial \mathbf{n}_{\text{fused}}^{(\text{orient})}} = \begin{cases} -\frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{N}}(u, v)}, & \text{if flipped} \\ \frac{\partial \mathcal{L}_{\text{normal}}}{\partial \tilde{\mathbf{N}}(u, v)}, & \text{otherwise} \end{cases}, \quad (32)$$

Normalization. The gradient is then backpropagated through the normalization operation $\tilde{\mathbf{N}} = \mathbf{n}_{\text{fused}} / \|\mathbf{n}_{\text{fused}}\|$. The gradient w.r.t. the unnormalized vector $\mathbf{n}_{\text{fused}}$ is given by:

$$\frac{\partial \mathbf{n}_{\text{fused}}^{(\text{orient})}}{\partial \mathbf{n}_{\text{fused}}} = \frac{(\mathbf{I} - \tilde{\mathbf{N}}(u, v) \tilde{\mathbf{N}}(u, v)^\top)}{\|\mathbf{n}_{\text{fused}}\|}. \quad (33)$$

Normal Fusion. The gradient is distributed to the two con-

stituent normal estimates $\mathbf{n}_1$ and $\mathbf{n}_2$ from which $\mathbf{n}_{\text{fused}} = \lambda \mathbf{n}_1 + (1 - \lambda) \mathbf{n}_2$ is computed. If their directions were inconsistent in the forward pass ($\mathbf{n}_1 \cdot \mathbf{n}_2 < 0$), the sign of the gradient for $\mathbf{n}_2$ is flipped:

$$\frac{\partial \mathbf{n}_{\text{fused}}}{\partial \mathbf{n}_1} = \lambda, \quad \frac{\partial \mathbf{n}_{\text{fused}}}{\partial \mathbf{n}_2} = (1 - \lambda) \cdot \text{sign}(\mathbf{n}_1 \cdot \mathbf{n}_2). \quad (34)$$

Finite Differences to 3D Points. The core of the backward pass involves computing how the estimated normals $\mathbf{n}_1$ and $\mathbf{n}_2$ change with respect to the 3D points $\mathbf{P}_{\text{world}}$ in their finite-difference neighborhood. For simplicity, we will take $\mathbf{n}_{\mathbf{X}} \in \{\mathbf{n}_1, \mathbf{n}_2\}$, $\text{STEP_X} \in \{\text{STEP_1}, \text{STEP_2}\}$ as an example for the subsequent derivation. For a normal estimate $\mathbf{n}_{\mathbf{X}}$ calculated from points $\mathbf{P}$ in a $\text{STEP_X} \times \text{STEP_X}$ window, the gradient w.r.t. each involved point $\mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}$ is derived from the cross product operation.

From Alg. 1, $\mathbf{v}_x^{\mathbf{n}_{\mathbf{X}}} = \mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}(u + \text{STEP_X}, v) - \mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}(u - \text{STEP_X}, v)$ and $\mathbf{v}_y^{\mathbf{n}_{\mathbf{X}}} = \mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}(u, v + \text{STEP_X}) - \mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}(u, v - \text{STEP_X})$, the normal is $\mathbf{n}_{\mathbf{X}} = \mathbf{v}_x^{\mathbf{n}_{\mathbf{X}}} \times \mathbf{v}_y^{\mathbf{n}_{\mathbf{X}}}$. The gradients of $\mathbf{n}_{\mathbf{X}}$ w.r.t. the vectors $\mathbf{v}_x^{\mathbf{n}_{\mathbf{X}}}$ and $\mathbf{v}_y^{\mathbf{n}_{\mathbf{X}}}$ are:

$$\frac{\partial \mathbf{n}_{\mathbf{X}}}{\partial \mathbf{v}_x^{\mathbf{n}_{\mathbf{X}}}} = \mathbf{v}_y^{\mathbf{n}_{\mathbf{X}}}, \quad \frac{\partial \mathbf{n}_{\mathbf{X}}}{\partial \mathbf{v}_y^{\mathbf{n}_{\mathbf{X}}}} = -\mathbf{v}_x^{\mathbf{n}_{\mathbf{X}}}. \quad (35)$$

These gradients are then distributed to the specific points that contributed to $\mathbf{v}_x^{\mathbf{n}_{\mathbf{X}}}$ and $\mathbf{v}_y^{\mathbf{n}_{\mathbf{X}}}$:

$$\begin{aligned} \frac{\partial \mathbf{n}_{\mathbf{X}}}{\partial \mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}(u + \text{STEP_X}, v)} &= \mathbf{v}_x^{\mathbf{n}_{\mathbf{X}}}, \\ \frac{\partial \mathbf{n}_{\mathbf{X}}}{\partial \mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}(u - \text{STEP_X}, v)} &= -\mathbf{v}_x^{\mathbf{n}_{\mathbf{X}}}, \\ \frac{\partial \mathbf{n}_{\mathbf{X}}}{\partial \mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}(u, v + \text{STEP_X})} &= \mathbf{v}_y^{\mathbf{n}_{\mathbf{X}}}, \\ \frac{\partial \mathbf{n}_{\mathbf{X}}}{\partial \mathbf{P}_{\text{world}}^{\mathbf{n}_{\mathbf{X}}}(u, v - \text{STEP_X})} &= -\mathbf{v}_y^{\mathbf{n}_{\mathbf{X}}}. \end{aligned} \quad (36)$$

This process is repeated for *both* normal estimates $\mathbf{n}_1$ and $\mathbf{n}_2$ and their respective neighborhoods $\mathbf{P}_{\text{world}}^{n_1}$ and $\mathbf{P}_{\text{world}}^{n_2}$, and the gradients are accumulated atomically.

3D Points w.r.t Depth. The Jacobian matrix $\frac{\partial \mathbf{P}_{\text{world}}}{\partial \tilde{\mathbf{D}}(u, v)}$ is derived from the backprojection and transformation chain $\mathbf{P}_{\text{world}} = \mathbf{R}_{\text{cam}}^{\text{world}} \mathbf{P}_{\text{cam}}(\tilde{\mathbf{D}}(u, v))$:

$$\frac{\partial \mathbf{P}_{\text{world}}}{\partial \tilde{\mathbf{D}}(u, v)} = \mathbf{R}_{\text{cam}}^{\text{world}} \frac{\partial \mathbf{P}_{\text{cam}}}{\partial \tilde{\mathbf{D}}(u, v)} = \mathbf{R}_{\text{cam}}^{\text{world}} \begin{bmatrix} (u - c_x)/f_x \\ (v - c_y)/f_y \\ 1 \\ 0 \end{bmatrix}. \quad (37)$$

6.5. Differentiable Pruning

Gradient Propagation. During backpropagation of k , the gradient can be propagated to the gradient factor k through the rendering equation. The chain rule for the gradient of the loss with respect to a specific k_i can be conceptually

expressed as

$$\frac{\partial \mathcal{L}_K(u, v)}{\partial k_i} = \frac{\partial \mathcal{L}_K(u, v)}{\partial \tilde{\mathbf{K}}(u, v)} \cdot \frac{\partial \tilde{\mathbf{K}}(u, v)}{\partial k_i}, \quad (38)$$

where $\frac{\partial \tilde{\mathbf{K}}(u, v)}{\partial k_i}$ can be derived from (8) in main text.

Detailed Analysis of optimal state $k = 1$. The gradient factor k serves as a core indicator for Gaussian pruning. Its deviation from 1 directly reflects the optimization state of Gaussians, and the underlying causes along with pruning rationale are elaborated as follows.

• **Gaussians with $k > 1$: Anomalies from Gradient Fluctuations Caused by Data Noise.** Gaussians with extremely large k -values are induced by gradient anomalies during multi-view optimization. During the rendering of specific views, data noise such as sensor noise and texture inconsistencies at scene edges leads to significant fluctuations in pixel gradients. These anomalous gradients propagate to the opacity α through backpropagation, causing α to deviate from its optimal value. This deviation results in unexpected optimization under specific view. Such abnormal k -values are usually accompanied by extreme Gaussian parameters including position, scale as shown in Fig. 7 in the main text.

• **Gaussians with $k < 1$: Redundancy Due to Insufficient Rendering Participation.** Gaussians with small k -values can be identified based on long-term optimization, especially considering the periodic k -reset strategy. During training, we periodically reset the k -value of all Gaussians to 0.9. This initial value is designed to provide a starting point for optimization driven by $\mathcal{L}_K$. The expectation is that after multiple optimization iterations, well-optimizing Gaussians will adjust their k -values from 0.9 toward 1. The ideal state is when $k = 1$, which makes the rendering formula of $\tilde{\mathbf{K}}$ satisfy

$$\tilde{\mathbf{K}}(u, v) = \sum_{i \in \mathcal{N}} \alpha_i \cdot 1 \cdot \prod_{j=1}^{i-1} (1 - \alpha_j) = 1. \quad (39)$$

If a Gaussian's k -value remains near 0.9 or changes minimally after a long optimization horizon, it indicates the Gaussian has not deeply participated in the rendering of any view. Specifically, it lacks sufficient effective alpha-rendering contributions to pixels across views. This situation will occur because the Gaussian is located in unobservable regions such as persistent occlusions. Such Gaussians are redundant, and their exclusion does not affect the accuracy of scene reconstruction as they contribute little to the rendering of $\tilde{\mathbf{K}}$ or RGB images.

Pruning Mechanism Based on k . In summary, well-optimized Gaussians can adjust their k -values toward 1 under the constraint of $\mathcal{L}_K$, and their α and geometric parameters can well match the scene without the need for extreme k -compensation. In contrast, Gaussians with $|k - 1| > \mathcal{T}_k$ are either anomalous with $k > 1$ or redundant with $k < 1$. By pruning these Gaussians, we can effectively reduce the

total number of Gaussians while preserving the quality of scene reconstruction, thereby improving the overall computational efficiency.

Algorithm 4: Tile-based Backward Gradient Propagation Pipeline

Input: Forward rendering results: $\tilde{\mathbf{D}}, \tilde{\mathbf{N}}, \tilde{\mathbf{O}}, \tilde{\mathbf{K}}$;
 Loss gradients: $\frac{\partial \mathcal{L}_{depth}}{\partial \tilde{\mathbf{D}}}, \frac{\partial \mathcal{L}_{normal}}{\partial \tilde{\mathbf{N}}}, \frac{\partial \mathcal{L}_{seg}}{\partial \tilde{\mathbf{O}}}, \frac{\partial \mathcal{L}_K}{\partial \tilde{\mathbf{K}}}$;
Output: Parameter gradients: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}}, \frac{\partial \mathcal{L}}{\partial \mathbf{R}}, \frac{\partial \mathcal{L}}{\partial \mathbf{s}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{\sigma}}, \frac{\partial \mathcal{L}}{\partial \mathbf{K}}$

```

1 for each pixel  $(x, y)$  do in parallel
  // Restore Saved Forward Values
2  Restore:  $T \leftarrow T_{final}$ ;
3  for each Gaussian  $i$  intersecting pixel  $pix$  (in
    reverse depth order) do
4    ...
5     $T \leftarrow T / (1 - \alpha_i)$ ;
6     $\mathbf{w}_\alpha \leftarrow \alpha_i \cdot T$ ;
    // Compute Gradients for
    Rendered Attributes
7    for each  $ch$  in  $C_o$  do
8       $\frac{\partial \mathcal{L}_{all}}{\partial \mathbf{o}_{i[ch]}} \leftarrow \underbrace{\frac{\partial \mathcal{L}_{seg}}{\partial \tilde{\mathbf{O}}_{[ch]}} \cdot \mathbf{w}_\alpha}_{(12)}$ ;
9       $\frac{\partial \mathcal{L}_{all}}{\partial \tilde{\mathbf{k}}_i} \leftarrow \underbrace{\frac{\partial \mathcal{L}_K}{\partial \tilde{\mathbf{K}}} \cdot \mathbf{w}_\alpha}_{(38)}$ ;
    // Depth Gradient Propagation
10      $\frac{\partial \mathcal{L}_{all}}{\partial d_i} \leftarrow \underbrace{\left( \frac{\partial \mathcal{L}_{normal}}{\partial \tilde{\mathbf{N}}} + \frac{\partial \mathcal{L}_{depth}}{\partial \tilde{\mathbf{D}}} \right) \cdot \mathbf{w}_\alpha}_{(31)}$ ;
    // Propagate Depth Gradient
    to Ellipsoid Parameters
11      $\frac{\partial \mathcal{L}_{all}}{\partial \mathbf{s}} = \frac{\partial \mathcal{L}_{all}}{\partial d_i} \cdot \underbrace{\frac{\partial d_i}{\partial \mathbf{s}}}_{(24)}$ ;
12      $\frac{\partial \mathcal{L}_{all}}{\partial \boldsymbol{\mu}} = \frac{\partial \mathcal{L}_{all}}{\partial d_i} \cdot \underbrace{\frac{\partial d_i}{\partial \boldsymbol{\mu}}}_{(25)}$ ;
13      $\frac{\partial \mathcal{L}_{all}}{\partial \mathbf{q}} = \frac{\partial \mathcal{L}_{all}}{\partial d_i} \cdot \underbrace{\frac{\partial d_i}{\partial \mathbf{R}} \cdot \frac{\partial \mathbf{R}}{\partial \mathbf{q}}}_{(26)}$ ;
14     ...

```

6.6. Tile-Based Gradient Backpropagation

In the previous sections, we obtain the losses for the semantic logits, depth, normal, and gradient factor map with respect to their respective ground truths, which are given by $\frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{O}}}, \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{D}}}, \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{N}}}$, and $\frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{K}}}$. Corresponding to Alg. 2, we optimize the Gaussian parameters through the backward prop-

agation of gradients via the alpha rendering process. The backward propagation pipeline is illustrated in Alg. 4. We omit the gradient propagation process for RGB (indicated as ...), which is consistent with the original implementation [26].

7. More Experiments

7.1. RGB Rendering Evaluation

To validate the effectiveness of our method in providing valuable information for NVS, we compare its performance on novel view RGB rendering with state-of-the-art frameworks [1] [45] [74] [26] [41] [34]. All methods utilize their provided original code and unified original data. The qualitative results of the RGB rendering are shown in Fig. 8, and the quantitative comparisons are presented in Tab. 5. It can be observed that our method achieves state-of-the-art results in novel view synthesis for certain scenes. By incorporating geometric constraints into the framework, our algorithm achieves SOTA geometric performance while maintaining RGB reconstruction quality and significantly improving memory and rendering efficiency.

7.2. Statistical Outlier Analysis

To quantify the quality of predicted point clouds relative to ground truth data in Replica [53], we employ a statistical approach based on Z-score analysis of point-to-point distances. This method allows us to identify and measure outliers in predicted point clouds with respect to the ground truth mesh points.

Distance. For each point p_i in the predicted point cloud P , we compute the minimum Euclidean distance to the ground truth point cloud G :

$$d_i = \min_{g \in G} |p_i - g|, \quad (40)$$

where $|\cdot|$ denotes the Euclidean norm. This computation is efficiently implemented using a KD-tree data structure for nearest neighbor search.

Statistical Measures. From the computed distances $d_1, d_2, \dots, d_n$ where n is the number of points in the predicted point cloud, we calculate the following statistical measures:

$$\mu_d = \frac{1}{n} \sum_{i=1}^n d_i, \quad (41)$$

$$\sigma_d = \sqrt{\frac{1}{n} \sum_{i=1}^n (d_i - \mu_d)^2}, \quad (42)$$

where μ_d represents the **Mean** distance and σ_d represents the standard deviation (**Std**) of distances.

For each distance d_i , we compute its Z-score, which measures how many standard deviations the distance is

Table 5. Quantitative comparison of RGB rendering on widely used real and simulated datasets. Some metrics (*) are extracted from the original data in respective papers. The results ranked from best to worst are highlighted as **first**, **second**, and **third**.

Dataset	Mip-NeRF360			Tanks & Temples			Deep Blending			Replica		
Method Metrics	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
iNPG* [45]	26.43	0.725	0.339	21.72	0.723	0.330	23.62	0.797	0.423	-	-	-
Mip-NeRF* [1]	29.23	0.844	0.207	22.22	0.759	0.257	29.40	0.901	0.245	-	-	-
RaySplats* [3]	27.31	0.846	0.237	22.20	0.829	0.202	29.57	0.900	0.320	-	-	-
GeoGaussian [34]	26.26	0.813	0.219	20.00	0.802	0.261	26.07	0.873	0.310	24.55	0.946	0.114
3DGS [26]	28.69	0.870	0.182	23.14	0.841	0.183	29.41	0.903	0.243	36.50	0.940	0.120
Scaffold-GS [41]	28.84	0.848	0.220	23.96	0.853	0.177	30.21	0.906	0.254	37.45	0.951	0.090
2DGS [18]	27.34	0.814	0.234	22.89	0.833	0.206	27.96	0.873	0.281	36.47	0.943	0.107
Normal-GS [62]	29.34	0.869	0.194	24.22	0.854	0.174	30.19	0.910	0.252	38.19	0.977	0.043
Ours	30.10	0.887	0.148	23.57	0.860	0.168	29.66	0.899	0.231	36.68	0.962	0.083



Figure 8. Qualitative comparison results on NVS demonstrate that our RGB rendering achieves better performance in certain scenarios. The top-left corner of the image displays the PSNR value computed against the ground truth image.

from the mean:

$$z_i = \frac{|d_i - \mu_d|}{\sigma_d}. \quad (43)$$

Points are classified as outliers based on a threshold parameter τ ($\tau = 0.1$ in our experiments):

$$\mathcal{F}_i = \begin{cases} 1 & \text{if } z_i > \tau \\ 0 & \text{otherwise} \end{cases}. \quad (44)$$

The outlier ratio (**Radio**) is then calculated as the proportion of points classified as outliers:

$$\text{Radio} = \frac{1}{n} \sum_{i=1}^n \mathcal{F}_i. \quad (45)$$

Hausdorff Distance. In addition to the statistical outlier analysis, we compute the Hausdorff distance to measure the

Table 6. Quantitative comparisons of the ablation experiments on each module.

Datasets		Room0-Sequence1					Office3-Sequence3						
Settings	Metrics	PSNR↑	Abs.Rel.↓	CosSimi↑	mIoU↑	Count↓	FPS↑	PSNR↑	Abs.Rel.↓	CosSimi↑	mIoU↑	Count↓	FPS↑
RayCasting-Point (All)		31.341	0.0062	0.855	0.607	170859	183.8	36.892	0.0208	0.931	0.534	178905	184.8
		31.800	0.0071	0.849	0.607	174177	181.7	36.873	0.0204	0.932	0.534	179128	187.8
w/o prune		31.509	0.0070	0.851	0.606	229487	156.3	37.100	0.0315	0.928	0.534	237914	165.5
w/prune $\mathcal{T}_k=0.2$		31.273	0.0068	0.847	0.605	144185	204.1	36.925	0.0161	0.932	0.534	148167	189.7
w/o seg		31.605	0.0226	0.837	0.021	174230	188.0	36.833	0.0225	0.929	0.024	184952	168.9
w/o normal		32.750	0.0061	0.774	0.608	167494	206.5	37.488	0.00885	0.895	0.535	182019	203.8
w/o depth		32.161	0.0184	0.850	0.607	171462	189.7	37.318	0.0985	0.925	0.539	176346	184.1
w/o depth&normal		33.691	0.0214	0.658	0.609	167198	202.6	37.449	0.1106	0.782	0.539	180747	189.7

maximum deviation between point clouds:

$$H(G, P) = \max \left\{ \sup_{g \in G} \inf_{p \in P} |g - p|, \sup_{p \in P} \inf_{g \in G} |p - g| \right\}. \quad (46)$$

7.3. More Results on Geometric Rendering

We present additional comparative results of depth and surface normal predictions across multiple datasets in Fig. 9, which demonstrate the superiority of our method in geometric rendering. Particularly in texture-sparse regions, where other methods fail to accurately reconstruct object surfaces, our approach consistently ensures high consistency among RGB, depth, and normal estimations.

It is worth noting that in the Mip-NeRF [1], Tanks & Temples [28], and Deep Blending [16] datasets, the depth and normal information used for supervision is provided by Normal-GS [62] and may still be inaccurate in certain scenarios. Nevertheless, our method achieves competitive results in both depth and normal estimation.

7.4. Ablation on Algorithm Components

We test the quantitative rendering quality and runtime analysis with and without the differentiable pruning module, as shown in Tab. 6. It is evident that our differentiable pruning module effectively reduces the number of Gaussians used without significantly compromising image rendering quality, thereby improving rendering FPS and reducing stray points in the infinite distance.

Furthermore, Tab. 6 demonstrates the performance gains achieved by our ray-casting based method on depth and normal estimation, as well as the impact of integrating various loss functions on their respective evaluation metrics. It can be observed that our ray-casting based approach enhances the algorithm’s performance on geometric metrics.

Unlike methods that optimize for a single modality, our focus lies in achieving a consistent representation of all modalities in the scene. We therefore strive for a balanced performance across all metrics, which ultimately leads to a more faithful reconstruction of the real world.

8. Multimodal Viewer

We also extend a visualization rendering software based on the *SIBR_viewer* and a CUDA rendering pipeline we developed. This tool supports real-time visualization of RGB, depth, normal, and semantic maps, achieving a rendering speed of up to 200 FPS. The corresponding viewer will also be open-sourced on GitHub.

9. Limitations

Real-world environments exhibit complex optical interactions including extensive light reflections and inter reflections. Image captured by a camera is the product of multi-path light propagation and global illumination effects. Our current approach does not explicitly model these radiometric phenomena such as light reflection and composition. Furthermore, this work focuses exclusively on supervised offline reconstruction of static scenes. Extending the framework to incorporate temporal coherence for dynamic scene modeling remains a direction for future work.

References

- [1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *CVPR*, pages 5470–5479, 2022. 1, 2, 6, 7, 4, 5
- [2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. In *ICCV*, pages 19697–19705, 2023. 1, 2
- [3] Krzysztof Byrski, Marcin Mazur, Jacek Tabor, Tadeusz Dziarmaga, Marcin Kadziolka, Dawid Baran, and Przemysław Spurek. RaySplats: Ray Tracing based Gaussian Splatting. *arXiv preprint arXiv:2501.19196*, 2025. 5
- [4] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. HAC: Hash-Grid Assisted Context for 3D Gaussian Splatting Compression. In *ECCV*, pages 422–438, 2024. 2
- [5] Youyu Chen, Junjun Jiang, Kui Jiang, Xiao Tang, Zhihao Li, Xianming Liu, and Yinyu Nie. DashGaussian: Optimizing 3D Gaussian Splatting in 200 Seconds. In *CVPR*, pages 11146–11155, 2025. 2
- [6] Zi-Ting Chou, Sheng-Yu Huang, I Liu, Yu-Chiang Frank Wang, et al. GSNeRF: Generalizable Semantic Neural Ra-

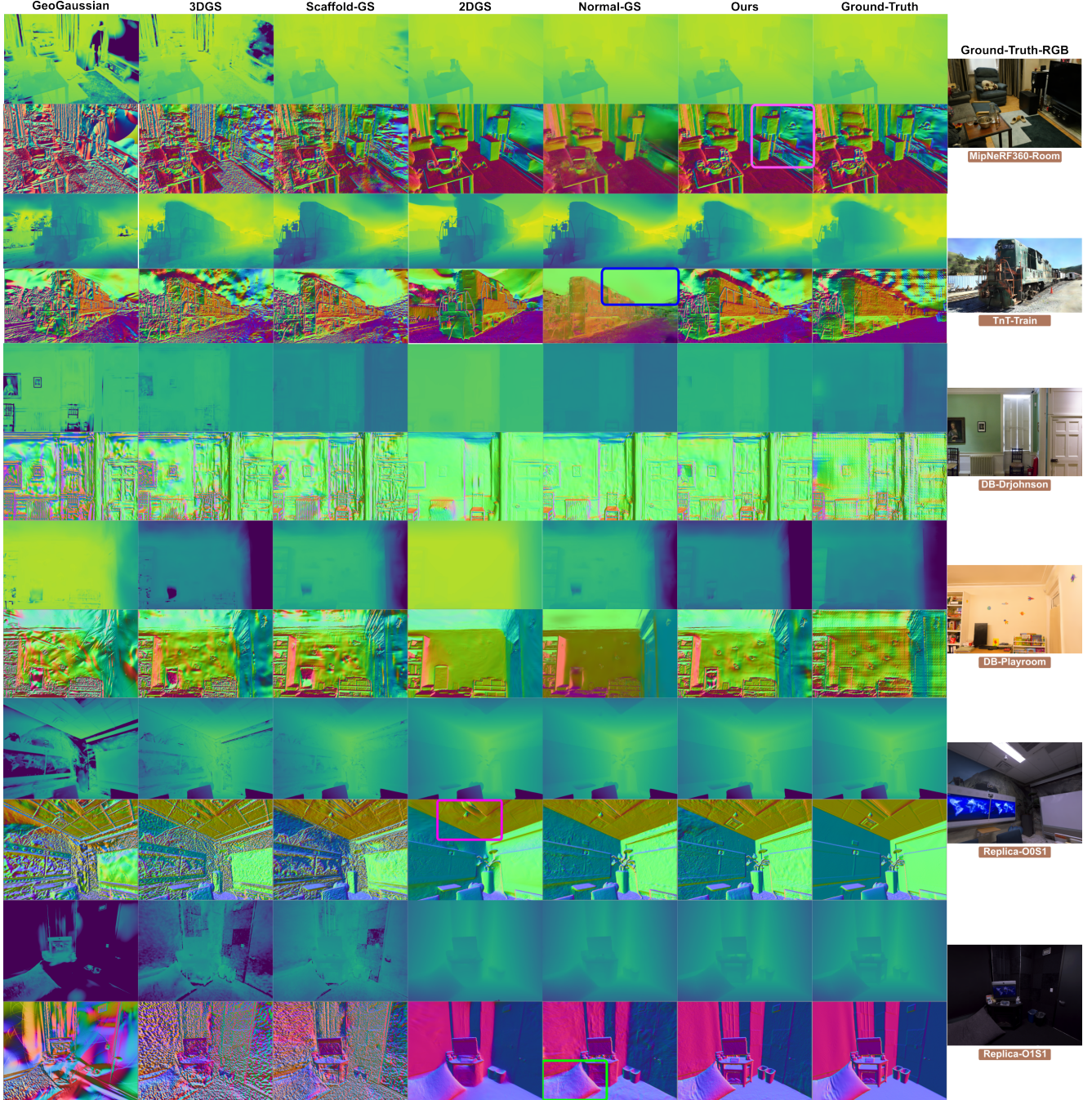


Figure 9. Qualitative comparison of depth and normal estimation accuracy across different methods on several datasets.

diance Fields with Enhanced 3D Scene Understanding. In *CVPR*, pages 20806–20815, 2024. 3

- [7] PyTorch Contributors. Installing C++ Distributions of PyTorch, 2024. 6
- [8] SuperSplat Contributors. SuperSplat - 3D Gaussian Splat Editor, 2025. 8
- [9] Pinxuan Dai, Jiamin Xu, Wenxiang Xie, Xinguo Liu, Huamin Wang, and Weiwei Xu. High-Quality Surface Reconstruction Using Gaussian Surfels. In *ACM SIGGRAPH*, pages 1–11, 2024. 3

- [10] Yinan Deng, Yufeng Yue, Jianyu Dou, Jingyu Zhao, Jiahui Wang, Yujie Tang, Yi Yang, and Mengyin Fu. OmniMap: A General Mapping Framework Integrating Optics, Geometry, and Semantics. *IEEE TRO*, 2025. 2, 3
- [11] Jiahua Dong and Yu-Xiong Wang. 3DGS-Drag: Dragging Gaussians for Intuitive Point-Based 3D Editing. In *ICLR*, 2025. 3
- [12] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, Zhangyang Wang, et al. LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS.

- NeurIPS*, 37:140138–140158, 2024. 2
- [13] Jian Gao, Chun Gu, Youtian Lin, Zhihao Li, Hao Zhu, Xun Cao, Li Zhang, and Yao Yao. Relightable 3D Gaussians: Realistic Point Cloud Relighting with BRDF Decomposition and Ray Tracing. In *ECCV*, pages 73–89, 2024. 3
- [14] Antoine Guédon and Vincent Lepetit. SUGAR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering. In *CVPR*, pages 5354–5363, 2024. 3
- [15] Jun Guo, Xiaojian Ma, Yue Fan, Huaping Liu, and Qing Li. Semantic Gaussians: Open-Vocabulary Scene Understanding with 3D Gaussian Splatting. *arXiv preprint arXiv:2403.15624*, 2024. 2, 3, 7
- [16] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep Blending for Free-Viewpoint Image-Based Rendering. *ACM SIGGRAPH*, 37(6):1–15, 2018. 6
- [17] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. *ACM SIGGRAPH*, 2024. 3
- [18] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In *ACM SIGGRAPH*, pages 1–11, 2024. 2, 6, 7, 5
- [19] Yuanhui Huang, Wenzhao Zheng, Yunpeng Zhang, Jie Zhou, and Jiwen Lu. GaussianFormer: Scene as Gaussians for Vision-Based 3D Semantic Occupancy Prediction. In *ECCV*, pages 376–393, 2024. 1
- [20] Yuanhui Huang, Amonnut Thammatadatrakoon, Wenzhao Zheng, Yunpeng Zhang, Dalong Du, and Jiwen Lu. GaussianFormer-2: Probabilistic Gaussian Superposition for Efficient 3D Occupancy Prediction. In *CVPR*, pages 27477–27486, 2025. 1
- [21] Hanqing Jiang, Xiaojun Xiang, Han Sun, Hongjie Li, Liyang Zhou, Xiaoyu Zhang, and Guofeng Zhang. GeoTexDensifier: Geometry-Texture-Aware Densification for High-Quality Photorealistic 3D Gaussian Splatting. *arXiv preprint arXiv:2412.16809*, 2024. 2
- [22] Minchao Jiang, Shunyu Jia, Jiaming Gu, Xiaoyuan Lu, Guangming Zhu, Anqi Dong, and Liang Zhang. VoteSplat: Hough Voting Gaussian Splatting for 3D Scene Understanding. *arXiv preprint arXiv:2506.22799*, 2025. 3
- [23] Yingwenqi Jiang, Jiadong Tu, Yuan Liu, Xifeng Gao, Xiaoxiao Long, Wenping Wang, and Yuexin Ma. GaussianShader: 3D Gaussian Splatting with Shading Functions for Reflective Surfaces. In *CVPR*, pages 5322–5332, 2024. 3, 7
- [24] Rui Jin, Yuman Gao, Yingjian Wang, Yuze Wu, Haojian Lu, Chao Xu, and Fei Gao. GS-Planner: A Gaussian-Splatting-Based Planning Framework for Active High-Fidelity Reconstruction. In *IROS*, pages 11202–11209, 2024. 3
- [25] Nikhil Varma Keetha, Jay Karhade, Krishna Murthy Jatavallabhula, Gengshan Yang, Sebastian Scherer, Deva Ramanan, and Jonathon Luiten. SplatTAM: Splat, Track & Map 3D Gaussians for Dense RGB-D SLAM. *CVPR*, pages 21357–21366, 2023. 3
- [26] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM TOG*, 42(4):139–1, 2023. 1, 2, 3, 5, 6, 7, 8, 4
- [27] Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A Hierarchical 3D Gaussian Representation for Real-time Rendering of Very Large Datasets. *ACM TOG*, 43(4):1–15, 2024. 1, 2
- [28] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *ACM TOG*, 36(4):1–13, 2017. 6
- [29] Zihang Lai. Exploring Simple Open-Vocabulary Semantic Segmentation. In *CVPR*, pages 30221–30230, 2025. 3
- [30] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3D Gaussian Representation for Radiance Field. In *CVPR*, pages 21719–21728, 2024. 2
- [31] Xiaohan Lei, Min Wang, Wengang Zhou, and Houqiang Li. GaussNav: Gaussian Splatting for Visual Navigation. *IEEE TPAMI*, 2025. 3
- [32] Boyi Li, Kilian Q Weinberger, Serge Belongie, Vladlen Koltun, and René Ranftl. Language-Driven Semantic Segmentation. *arXiv preprint arXiv:2201.03546*, 2022. 3
- [33] Mingrui Li, Shuhong Liu, Heng Zhou, Guohao Zhu, Na Cheng, Tianchen Deng, and Hongyu Wang. SGS-SLAM: Semantic Gaussian Splatting for Neural Dense SLAM. In *ECCV*, pages 163–179, 2024. 2, 3, 7
- [34] Yanyan Li, Chenyu Lyu, Yan Di, Guangyao Zhai, Gim Hee Lee, and Federico Tombari. GeoGaussian: Geometry-Aware Gaussian Splatting for Scene Rendering. In *ECCV*, pages 441–457, 2024. 2, 3, 6, 7, 8, 4, 5
- [35] Yue Li, Qi Ma, Runyi Yang, Huapeng Li, Mengjiao Ma, Bin Ren, Nikola Popovic, Nicu Sebe, Ender Konukoglu, Theo Gevers, et al. SceneSplat: Gaussian Splatting-Based Scene Understanding with Vision-Language Pretraining. *arXiv preprint arXiv:2503.18052*, 2025. 2
- [36] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *CVPR*, pages 8456–8455, 2023. 3
- [37] Feng Liang, Bichen Wu, Xiaoliang Dai, Kunpeng Li, Yinan Zhao, Hang Zhang, Peizhao Zhang, Peter Vajda, and Diana Marculescu. Open-Vocabulary Semantic Segmentation with Mask-Adapted CLIP. In *CVPR*, pages 7061–7070, 2023. 3
- [38] Yang Liu, Chuanchen Luo, Lue Fan, Naiyan Wang, Junran Peng, and Zhaoxiang Zhang. CityGaussian: Real-Time High-Quality Large-Scale Scene Rendering with Gaussians. In *ECCV*, pages 265–282, 2024. 2
- [39] Yang Liu, Chuanchen Luo, Zhongkai Mao, Junran Peng, and Zhaoxiang Zhang. CityGaussianV2: Efficient and Geometrically Accurate Reconstruction for Large-Scale Scenes. *arXiv preprint arXiv:2411.00771*, 2024. 2
- [40] Guanxing Lu, Shiyi Zhang, Ziwei Wang, Changliu Liu, Jiwen Lu, and Yansong Tang. ManiGaussian: Dynamic Gaussian Splatting for Multi-Task Robotic Manipulation. In *ECCV*, pages 349–366. Springer, 2024. 1
- [41] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-GS: Structured 3D Gaussians for View-adaptive Rendering. In *CVPR*, pages 20654–20664, 2024. 2, 3, 5, 6, 7, 8, 4

- [42] Huaishao Luo, Junwei Bao, Youzheng Wu, Xiaodong He, and Tianrui Li. SegCLIP: Patch Aggregation with Learnable Centers for Open-Vocabulary Semantic Segmentation. In *ICML*, pages 23033–23044, 2023. 3
- [43] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020. 1, 2
- [44] Nicolas Moenne-Loccoz, Ashkan Mirzaei, Or Perel, Riccardo de Lutio, Janick Martinez Esturo, Gavriel State, Sanja Fidler, Nicholas Sharp, and Zan Gojcic. 3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes. *ACM SIGGRAPH Asia*, 2024. 3
- [45] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM TOG*, 41(4):1–15, 2022. 2, 7, 4, 5
- [46] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis. In *CVPR*, pages 10349–10358, 2024. 2
- [47] Michael Oechsle, Songyou Peng, and Andreas Geiger. UniSurf: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction. In *ICCV*, pages 5589–5599, 2021. 3
- [48] Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. LangSplat: 3D Language Gaussian Splatting. In *CVPR*, pages 20051–20060, 2024. 3
- [49] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*, pages 8748–8763, 2021. 3
- [50] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-GS: Towards Consistent Real-Time Rendering with LOD-Structured 3D Gaussians. *arXiv preprint arXiv:2403.17898*, 2024. 2
- [51] Ji Shi, Xianghua Ying, Ruohao Guo, Bowei Xing, and Wenzhen Yue. Normal-NeRF: Ambiguity-Robust Normal Estimation for Highly Reflective Scenes. In *AAAI*, pages 6869–6877, 2025. 2, 3
- [52] Yahao Shi, Yanmin Wu, Chenming Wu, Xing Liu, Chen Zhao, Haocheng Feng, Jian Zhang, Bin Zhou, Errui Ding, and Jingdong Wang. GIR: 3D Gaussian Inverse Rendering for Relightable Scene Factorization. *IEEE TPAMI*, 2025. 3
- [53] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The Replica Dataset: A Digital Replica of Indoor Spaces. *arXiv preprint arXiv:1906.05797*, 2019. 7, 4
- [54] Matthew Tancik, Vincent Casser, Xinchun Yan, Sabeek Pradhan, Ben P. Mildenhall, Pratul Srinivasan, Jonathan T. Barron, and Henrik Kretschmar. Block-NeRF: Scalable Large Scene Neural View Synthesis. In *CVPR*, pages 8238–8248, 2022. 2
- [55] Shengji Tang, Weicai Ye, Peng Ye, Weihao Lin, Yang Zhou, Tao Chen, and Wanli Ouyang. HiSplat: Hierarchical 3D Gaussian Splatting for Generalizable Sparse-View Reconstruction. *ICLR*, 2025. 2, 7
- [56] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields. In *CVPR*, pages 5481–5490, 2022. 1
- [57] Evangelos Ververas, Rolandos Alexandros Potamias, Jifei Song, Jiankang Deng, and Stefanos Zafeiriou. SAGS: Structure-Aware 3D Gaussian Splatting. In *ECCV*, pages 221–238, 2024. 3
- [58] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-View Reconstruction. *arXiv preprint arXiv:2106.10689*, 2021. 3
- [59] Peihao Wang, Yuehao Wang, Dilin Wang, Sreyas Mohan, Zhiwen Fan, Lemeng Wu, Ruisi Cai, Yu-Ying Yeh, Zhangyang Wang, Qiang Liu, et al. Steepest Descent Density Control for Compact 3D Gaussian Splatting. In *CVPR*, pages 26663–26672, 2025. 2
- [60] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. NeuS2: Fast Learning of Neural Implicit Surfaces for Multi-View Reconstruction. In *ICCV*, pages 3295–3306, 2023. 3
- [61] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image Quality Sssessment: from Error Visibility to Structural Similarity. *IEEE TIP*, 13(4):600–612, 2004. 6, 7
- [62] Meng Wei, Qianyi Wu, Jianmin Zheng, Hamid RezaTofighi, and Jianfei Cai. Normal-GS: 3D Gaussian Splatting with Normal-Involved Rendering. *arXiv preprint arXiv:2410.20593*, 2024. 2, 3, 6, 7, 8, 5
- [63] Tianci Wen, Zhiang Liu, and Yongchun Fang. SEGSLAM: Structure-Enhanced 3D Gaussian Splatting SLAM with Appearance Embedding. *arXiv preprint arXiv:2501.05242*, 2025. 3
- [64] Qi Wu, Janick Martinez Esturo, Ashkan Mirzaei, Nicolas Moenne-Loccoz, and Zan Gojcic. 3DGUT: Enabling Distorted Cameras and Secondary Rays in Gaussian Splatting. *CVPR*, pages 26036–26046, 2025. 3
- [65] Tong Wu, Jia-Mu Sun, Yu-Kun Lai, Yuewen Ma, Leif Kobbelt, and Lin Gao. DeferredGS: Decoupled and Editable Gaussian Splatting with Deferred Shading. *arXiv preprint arXiv:2404.09412*, 2024. 3
- [66] Yusen Xie, Zhenmin Huang, Jin Wu, and Jun Ma. GS-LIVM: Real-Time Photo-Realistic LiDAR-Inertial-Visual Mapping with Gaussian Splatting. *ICCV*, 2025. 2
- [67] Yunzhi Yan, Haotong Lin, Chenxu Zhou, Weijie Wang, Haiyang Sun, Kun Zhan, Xianpeng Lang, Xiaowei Zhou, and Sida Peng. Street Gaussians: Modeling Dynamic Urban Scenes with Gaussian Splatting. In *ECCV*, pages 156–173, 2024. 1, 2
- [68] Runyi Yang, Zhenxin Zhu, Zhou Jiang, Baijun Ye, Xiaoxue Chen, Yifei Zhang, Yuantao Chen, Jian Zhao, and Hao Zhao. Spectrally Pruned Gaussian Fields with Neural Compensation. *arXiv preprint arXiv:2405.00676*, 2024. 2
- [69] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume Rendering of Neural Implicit Surfaces. *NeurIPS*, 34: 4805–4815, 2021. 3

- [70] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. BakedSDF: Meshing Neural SDFs for Real-Time View Synthesis. In *ACM SIGGRAPH*, pages 1–9, 2023. [3](#)
- [71] Kai Ye, Chong Gao, Guanbin Li, Wenzheng Chen, and Baoquan Chen. GeoSplatting: Towards Geometry Guided Gaussian Splatting for Physically-Based Inverse Rendering. *arXiv preprint arXiv:2410.24204*, 2024. [3](#)
- [72] Keyang Ye, Qiming Hou, and Kun Zhou. 3D Gaussian Splatting with Deferred Reflection. In *ACM SIGGRAPH*, pages 1–10, 2024. [3](#)
- [73] Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. Gaussian Grouping: Segment and Edit Anything in 3D Scenes. In *ECCV*, pages 162–179, 2024. [3](#)
- [74] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance Fields Without Neural Networks. In *CVPR*, page 6, 2021. [2](#), [4](#)
- [75] Mulin Yu, Tao Lu, Linning Xu, Lihan Jiang, Yuanbo Xiangli, and Bo Dai. GSDF: 3DGS Meets SDF for Improved Neural Rendering and Reconstruction. *NeurIPS*, 37:129507–129530, 2024. [3](#)
- [76] Hao Zhang, Feng Li, Xueyan Zou, Shilong Liu, Chunyuan Li, Jianwei Yang, and Lei Zhang. A Simple Framework for Open-Vocabulary Segmentation and Detection. In *ICCV*, pages 1020–1031, 2023. [3](#), [6](#)
- [77] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*, pages 586–595, 2018. [7](#)
- [78] Zheng Zhang, Wenbo Hu, Yixing Lao, Tong He, and Hengshuang Zhao. Pixel-GS: Density Control with Pixel-Aware Gradient for 3D Gaussian Splatting. In *ECCV*, pages 326–342, 2024. [2](#)
- [79] Wenzhao Zheng, Junjie Wu, Yao Zheng, Sicheng Zuo, Zixun Xie, Longchao Yang, Yong Pan, Zhihui Hao, Peng Jia, Xi'anpeng Lang, et al. GaussianAD: Gaussian-Centric End-to-End Autonomous Driving. *arXiv preprint arXiv:2412.10371*, 2024. [1](#)
- [80] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew J. Davison. In-Place Scene Labelling and Understanding with Implicit Scene Representation. In *ICCV*, 2021. [2](#), [3](#)
- [81] Hongyu Zhou, Jiahao Shao, Lu Xu, Dongfeng Bai, Weichao Qiu, Bingbing Liu, Yue Wang, Andreas Geiger, and Yiyi Liao. HuGS: Holistic Urban 3D Scene Understanding via Gaussian Splatting. In *CVPR*, pages 21336–21345, 2024. [3](#)
- [82] Shijie Zhou, Haoran Chang, Sicheng Jiang, Zhiwen Fan, Zehao Zhu, Dejia Xu, Pradyumna Chari, Suyu You, Zhangyang Wang, and Achuta Kadambi. Feature 3DGS: Supercharging 3D Gaussian Splatting to Enable Distilled Feature Fields. In *CVPR*, pages 21676–21685, 2024. [3](#)
- [83] Siting Zhu, Guangming Wang, Xin Kong, Dezhi Kong, and Hesheng Wang. 3D Gaussian Splatting in Robotics: A Survey. *arXiv preprint arXiv:2410.12262*, 2024. [1](#)