

BillBoard Splatting (BBSplat): Learnable Textured Primitives for Novel View Synthesis

David Svitov^{1,2} Pietro Morerio² Lourdes Agapito³ Alessio Del Bue²

¹Università degli Studi di Genova, Genoa, Italy

²Istituto Italiano di Tecnologia (IIT), Genoa, Italy

³Department of Computer Science, University College London

{david.svitov, pietro.morerio, alessio.delbue}@iit.it l.agapito@cs.ucl.ac.uk

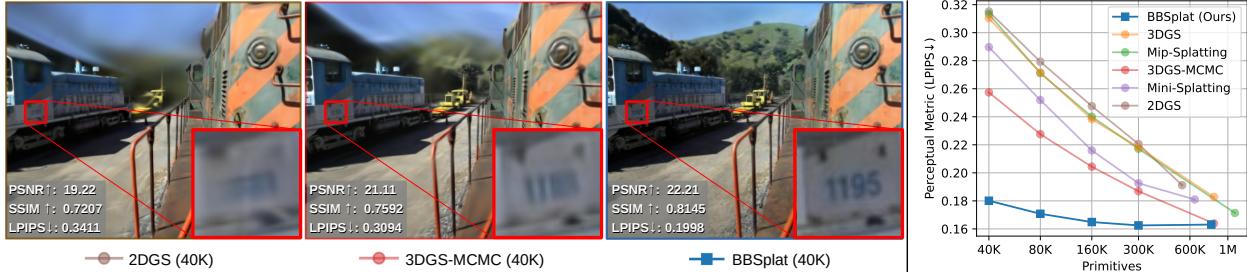


Figure 1. **BBSplat has a competitive advantage on novel view synthesis (NVS) with less primitives.** Left: BBSplat allows for more detailed NVS than 3DGS and 2DGS with equal number of Gaussians, e.g. background and planar regions have higher quality as they are better modeled by BBSplat’s textured primitives. Right: The plot shows perceptual similarity score (the lower, the better) at a varying number of primitives. BBSplat show better performance than state-of-the-art methods, with a more evident gap with less primitives.

Abstract

We present billboard Splatting (BBSplat) - a novel approach for 3D scene representation based on textured geometric primitives. BBSplat represents the scene as a set of optimizable textured planar primitives with learnable RGB textures and alpha-maps to control their shape. BBSplat primitives can be used in any Gaussian Splatting pipeline as drop-in replacements for Gaussians. Our method’s qualitative and quantitative improvements over 3D and 2D Gaussians are most noticeable when fewer primitives are used, when BBSplat achieves over 1200 FPS. Our novel regularization term encourages textures to have a sparser structure, unlocking an efficient compression that leads to a reduction in storage space of the model. Our experiments show the efficiency of BBSplat on standard datasets of real indoor and outdoor scenes such as Tanks&Temples, DTU, and Mip-NeRF-360. We demonstrate improvements on PSNR, SSIM, and LPIPS metrics compared to the state-of-the-art, especially for the case when fewer primitives are used, which, on the other hand, leads to up to 2× inference speed improvement for the same rendering quality. Project page: david-svitov.github.io/BBSplat_project_page.

1. Introduction

Novel view synthesis (NVS) is a crucial technology for a variety of applications, including virtual reality, computer gaming, and cinematography. Several efforts have been dedicated to deploy methods that are more efficient while providing a better quality of the synthesized images. Specifically, the choice of geometric primitives used to represent the scene plays a key role in defining the advantages and drawbacks of different NVS methods. Recent breakthroughs on neural representations [4, 5, 7, 8, 18, 30, 31, 34, 35, 47, 51] have been sided by recent methods based on Gaussian Splatting [10, 13, 19, 23, 24, 36, 52, 54], demonstrating most efficient way for novel view rendering.

Indeed, NeRF-based methods [4, 5, 34, 35] still achieve the best NVS quality for challenging real-world captures by using implicit scene representations such as the weights of an MLP. However, image rendering with a NeRF is less efficient as it requires repeated MLP inferences to predict colors along camera rays. An alternative to neural rendering, 3D Gaussian Splatting (3DGS) [23] uses faster rendering based on projecting explicit primitives on the screen surface while preserving high-quality NVS. In practice, 3DGS

uses Gaussian-distributed radiance around explicit 3D scene points as primitives.

Recently, 2D Gaussian Splatting (2DGS) [19] proposed the use of flat Gaussians, oriented in 3D, to represent the scene more efficiently. Since 2D Gaussians are effectively tangent to object surfaces, they allow more accurate surface extraction. Despite proving their efficiency in mesh extraction task, 2D primitives result in a downgrade of rendering metrics compared to 3D primitives such as 3D Gaussians. In this work, we aim to make 2D primitives suitable for high-quality NVS by introducing a new primitive representation.

Our proposed geometric primitives for NVS take inspiration from the classic *billboards* used for extreme 3D model simplification [9] by replacing mesh during a greedy optimization process. A 3D scene can be efficiently rendered by using a “billboard cloud” of given textured planar primitives with alpha channels. Using billboards, we can efficiently model planar surfaces, such as a painting on a wall or scene background, while dramatically reducing the number of geometric primitives required, up to an order of magnitude with respect to 3DGS/2DGS (check Fig. 1 for a comparison). In combination with efficient texture sampling implemented on the GPU, this leads to inference speed improvement without a drop in the rendering quality.

We define *billboards* with 2D Gaussians parameters (rotation, scaling, 3D center location, and spherical harmonics) while also introducing RGB texture and alpha map to control pixel-wise color and shape (Fig. 2(b2)). The alpha map defines the billboard silhouette and models the arbitrary shape of primitives. Similarly, the RGB texture stores color for each point of the billboard. In this way, we can use fewer primitives to represent high-frequency details (Fig. 1). The key aspect of our approach, BillBoard Splatting (BBSplat), is a method to learn billboard parameters from a set of calibrated images.

To ensure rendering efficiency, we implemented texture sampling and a back-propagation process in CUDA. In this way, the use of textures does not result in overtime compared to the 2DGS Gaussians’ rasterization process. To tackle the challenge of storing all billboard textures, we compress them by representing each texture as sparse offsets from colors calculated by spherical harmonics [2] and by further quantizing them to 8 bits. Then, we can efficiently utilize dictionary-based compression algorithms [11, 41] for quantized textures.

To summarise, our contributions are as follows:

- We propose BBSplat with optimizable textured primitives to learn 3D scene representation with photometric losses. BBSplat allows up to $\times 2$ acceleration for NVS compared to the state-of-the-art for the same rendering quality.
- We developed an algorithm to efficiently represent and store textures for billboards. BBSplat demonstrates sig-

nificant storage cost reduction. As a result, for some scenarios, storing the scene represented with billboards is more efficient than 3DGS or 2DGS.

- We implemented BBSplat with CUDA and demonstrated its efficiency in extensive experiments on several real-scene open datasets.

2. Related work

Reconstruction of 3D scene for novel-view synthesis is a long-standing problem that used to be solved with structure from motion approaches (SfM) [15, 40, 44, 45, 56]. The significant increase in quality was achieved with the appearance of implicit scene representation [30, 34, 37].

Nowadays, Neural Radiance Fields (NeRF) [34] is the most commonly used implicit method for NVS. It uses a neural network to predict points’ colors and transparency based on points coordinates along the camera ray. The biggest drawback of such representation is rendering speed, as it requires multiple inferences of an MLP. InstantNGP [35] achieves the most acceleration of NeRF up to 150 FPS by using multi-scale feature grids.

To overcome the rendering speed issue, the 3D Gaussian Splatting (3DGS) [23] method proposed an explicit method for 3D scene representation. Based on the point cloud reconstructed by SfM [40], they train the color and orientation of ellipsoids with Gaussian distributed transparency. For NVS, these ellipsoids are splatted on the screen surface providing high rendering speed. As a result, 3DGS has become popular for many practical real-time applications such as human avatars [29, 46, 50] or SLAM [32, 53].

Recent works further improve 3DGS in terms of rendering quality and speed. Thus, in [6] authors revise the densification algorithm to reduce the number of artifacts by using loss-based splitting of Gaussians. In 3DGS-MCMC [24], authors use Markov Chain Monte Carlo interpretation of sampling algorithm to more efficiently distribute Gaussians over the scene, even with random initialization. RadSplat [36] uses pre-trained NeRF representation as a prior during 3DGS training to improve rendering quality in challenging scenarios. Mip-Splatting [52] solves the issue of gaps between Gaussian produced due to changes in camera distance by introducing a 3D smoothing filter that constrains Gaussian size.

Some of the recent works also propose to disentangle appearance and geometry for 3DGS. Texture-GS [49] introduces an MLP to establish correspondence between 3D Gaussian surface and texture map. In practice this approach allows one to edit the texture in the scene but decrease rendering speed to three times. Another approach [20] encodes color and opacity as spherical harmonics on the surface of 3D Gaussian, which allows one to improve reconstruction fidelity.

2D Gaussian splatting (2DGS) [19] was proposed to in-

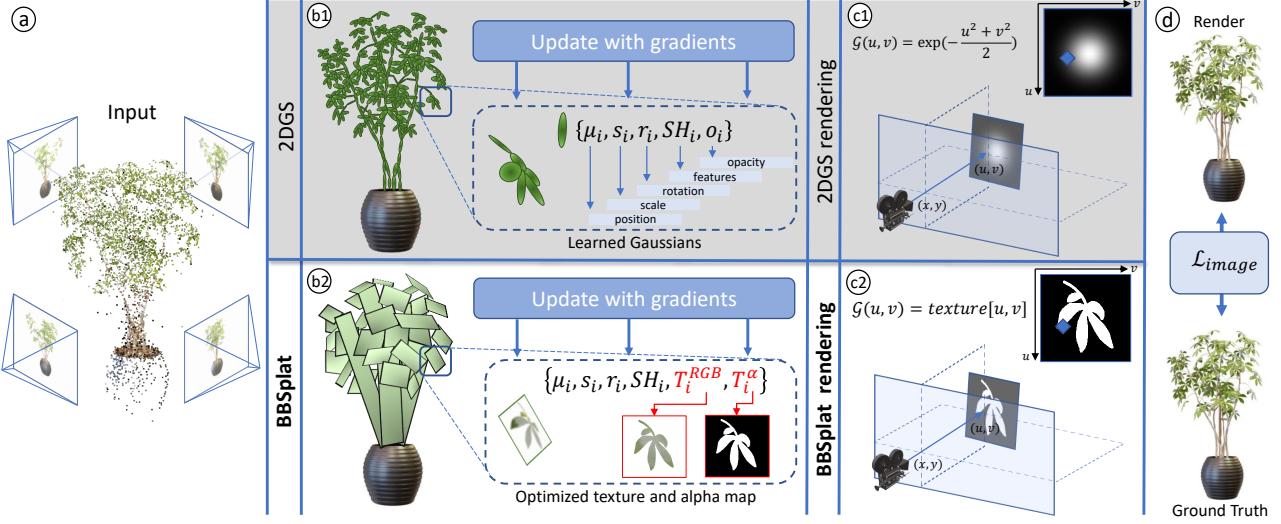


Figure 2. Method description. a) As input we use point cloud and camera positions predicted with COLMAP [40]. b) Our BBSplat parametrization (b2) extends Gaussian primitives parametrization (b1) with two textures for each point: RGB texture for colors and alpha texture for transparency. c) As defined in 2DGS splatting technique (c1), we find the ray-plane intersection, but instead of calculating Gaussian opacity, we sample color and opacity from the texture (c2). d) To train our 3D scene representation, we utilize only photometric losses.

crease the quality of mesh surface reconstruction from such scene representation. Specifically, 2DGS uses ellipses instead of ellipsoids as they align along the object’s surface and make mesh extraction operation more straightforward. For faster rendering of such flat primitives, 2DGS utilizes a ray-splat intersection algorithm [42, 48]. While 2DGS increases the accuracy of mesh reconstruction, its fidelity falls short behind 3DGS, especially for in-the-wild scenes. In this work, we investigate the ability of flat primitives to be used for high-quality novel-view rendering.

We noted that existing approaches use a tremendous number of Gaussians to represent the scene, which affects rendering speed. Billboard clouds [9] propose an extreme 3D model simplification by replacing it with a set of textured planes. This representation is widely used in computer graphics for real-time forest rendering [3, 14, 27] and found implementation in such tasks as automatic paper slice-form design [28, 33, 39] and level of detail control [17]. In this work, we take inspiration from the billboard clouds technique [9] and propose arbitrarily shaped textured primitives as an extension of the 2DGS approach, resulting in faster and more qualitative rendering of real scenes reconstructed from the set of photographs.

3. Method

3.1. Preliminaries: 2D Gaussian Splatting

Recently proposed 2D Gaussian splatting (2DGS) [19] uses 2D flat ellipses (in contrast with 3D ellipsoids for 3DGS) orientated along the object’s surface to represent the scene.

These ellipses have Gaussian distributed transparency and are parametrized as $\{\mu_i, s_i, r_i, o_i, \text{SH}_i\}$ which corresponds to position, 2D scale, quaternion rotation, global opacity, and spherical harmonics of the primitive as in Fig. 2(b1). Here spherical harmonics [2, 12] define view-dependent color. In practice, to calculate ellipses orientation, 2DGS uses local tangent planes defined with transformation matrix H_i as:

$$\mathcal{P}_i(u, v) = H_i(u, v, 1, 1)^T = \begin{bmatrix} R_i S_i & \mu_i \\ 0 & 1 \end{bmatrix} (u, v, 1, 1)^T, \quad (1)$$

where matrices R_i and S_i produced from quaternion rotation r_i and scale s_i , and (u, v) is the point coordinate on the plane \mathcal{P}_i . Then, 2DGS utilizes (u, v) coordinates of points within a plane to calculate 2D Gaussian intensity as follows:

$$g(u, v) = \exp\left(-\frac{u^2 + v^2}{2}\right) \quad (2)$$

To project Gaussians on the screen, 2DGS finds (u, v) coordinates corresponding to the screen pixel as the intersection of a camera-direction ray with a plane \mathcal{P}_i . For efficiency, 2DGS utilizes an explicit ray-splat intersection algorithm proposed by [42].

3.2. Billboard Splatting

In this work, we propose to further improve NVS efficiency and quality by introducing billboard splatting. These billboard primitives are based on planes, but leverage learnable textures instead of utilizing Gaussian distribution (Eq. (2))

to calculate their color and transparency. We inherit 2DGS parametrization for these planes: $\{\mu_i, s_i, r_i, \text{SH}_i\}$, where μ_i is the center of i -th plane, s_i is scales along two axis, r_i is rotation quaternion, SH_i is spherical harmonics coefficients. Instead of using global Gaussian opacity o_i we set transparency at each point of a plane as T_i^α texture (Fig. 2 (b2)). This way, each billboard can have an arbitrary shape. Additionally, we parametrize billboards with T_i^{RGB} - RGB texture to control the color of all points of the billboard.

We use explicit ray-splat intersection algorithm [42] to find the corresponding plane coordinate $\mathbf{u} = (u, v)$ for $\mathbf{x} = (x, y)$ screen coordinate. Specifically, we parametrize the camera ray as an intersection of two 4D homogeneous planes: $h_x = (-1, 0, 0, x)^T$ and $h_y = (0, -1, 0, y)^T$. We then transform these planes to the uv -coordinate system of plane H to find intersection point in (u, v) coordinates:

$$h_u = (WH)^T h_x \quad h_v = (WH)^T h_y, \quad (3)$$

where $W \in 4 \times 4$ is the transformation matrix from world space to screen space. Then we calculate the (u, v) coordinate of the intersection point as:

$$u(x) = \frac{h_u^2 h_v^4 - h_u^4 h_v^2}{h_u^1 h_v^2 - h_u^2 h_v^1} \quad v(x) = \frac{h_u^4 h_v^1 - h_u^1 h_v^4}{h_u^1 h_v^2 - h_u^2 h_v^1}, \quad (4)$$

where h_u^i, h_v^i are the i -th value of the 4D homogeneous plane parameters.

This way, we get (u, v) coordinates in the $[-1; 1]$ range with 0 corresponding to the plane center μ_i . After that, we rescale them to the range $[0; S_T]$, where S_T corresponds to texture size in texels. Using rescaled (u, v) coordinates, we sample *color* and *opacity* of the ray-splat intersection point (Fig. 2 (c2)) and accumulate them along the ray as:

$$c(x) = \sum_{i=1}^{i-1} c_i[\mathbf{u}(x)] T_i^\alpha[\mathbf{u}(x)] \prod_{j=1}^{i-1} (1 - T_j^\alpha[\mathbf{u}(x)]), \quad (5)$$

where c_i is the view-dependent color calculated with SH_i and sampled color $T_i^{\text{RGB}}[\mathbf{u}(x)]$. To sample textures' values $T[\cdot]$, we utilize *bilinear sampling* [43]. This way, we take into account neighboring texels and can calculate gradients for billboard position μ_i with respect to the texture. We adopt PyTorch [38] CUDA implementation of *bilinear sampling* to sample texture values and calculate gradients. Namely, we redistribute input gradient values between texels that contributed to the sampled value according to their contribution weights (*i.e.* bilinear coefficients).

To calculate value of view-dependent colors c_i at each uv-point \mathbf{u} of i -th billboard, we integrate sampled texture colors $T_i^{\text{RGB}}[\mathbf{u}]$ as offsets to the colors predicted with SH_i and view-direction vector \vec{d}_i :

$$c_i[\mathbf{u}] = T_i^{\text{RGB}}[\mathbf{u}] + \text{RGB}(\text{SH}_i, \vec{d}_i). \quad (6)$$

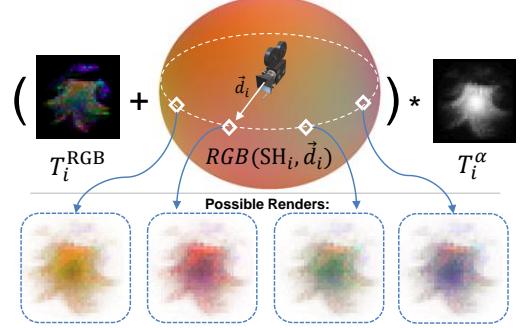


Figure 3. **Texture rasterization.** During rendering to get final billboard colors we add texture T_i^{RGB} colors to base color calculated with SH_i and view-direction vector \vec{d}_i . It results in textured billboards handling light effects. Here we showcase four possible renderings for different camera directions.

Here SH_i is a spherical harmonic features [2] for each primitive, and \vec{d}_i is the view-direction vector from the camera position to billboard center position μ_i . We transform SH_i to RGB color space and define this operation as $\text{RGB}(\cdot, \cdot)$. This way, we can handle view-dependent light effects for textured planes (Fig. 3). Representing the final color as a combination of spherical harmonics and sampled color was initially proposed in [49] and helps to disentangle texture and light effects. In our case it also allows T_i^{RGB} to have a sparse structure, which is more efficient to store as described in Section 3.5.

Final parametrization of billboard is as follows: $\{\mu_i, s_i, r_i, \text{SH}_i, T_i^{\text{RGB}}, T_i^\alpha\}$, where $\{\mu_i, s_i, r_i, \text{SH}_i\}$ follows 2DGS parametrization (Section 3.1) and the values $\{T_i^{\text{RGB}}, T_i^\alpha\}$ correspond to our proposed textures for color and opacity.

3.3. Training

To train billboard splatting representation of the scene, we initialize them with sparse point cloud obtained using SfM [40]. Optionally, we add 10K evenly distributed points on the circumscribing sphere using the Fibonacci algorithm [16] to represent the sky and far away objects. We use only photometric losses to fit a scene as in Gaussian splatting pipelines. For textures, we propose regularizations to avoid their overfitting and get a more sparse structure that reduces storage costs. In this section, we describe the losses and regularizations we use.

Photometric losses. We utilize photometric losses to train our billboard representation. Namely we apply \mathcal{L}_1 and structure similarity D-SSIM losses:

$$\mathcal{L}_{\text{image}} = (1 - \lambda_{\text{SSIM}})\mathcal{L}_1 + \lambda_{\text{SSIM}}\mathcal{L}_{\text{SSIM}}. \quad (7)$$

Therefore, we need only a set of images to train the BB-Splat representation of the scene.

Regularizations. Due to the large number of parameters, textures tend to overfit training images, which results in noisy rendering output for novel views. To alleviate this issue, we propose a simple yet efficient regularization in which we push billboards with low impact on the renderings (*i.e.* small or far billboards affecting few pixels) to have Gaussian distributed transparency.

First, we define per-texture visibility weight w_i for all N billboards based on their impact I_i on the rendered image. We define the impact for i -th billboard as a sum over alpha-blending values corresponding to its rendered pixels \mathcal{R}_i :

$$I_i = \sum_{x \in \mathcal{R}_i} \left(T_i^\alpha [\mathbf{u}(x)] \cdot \prod_{j=1}^{K_x} \left(1 - T_j^\alpha [\mathbf{u}(x)] \right) \right), \quad (8)$$

$$w_i = \begin{cases} \sigma - \min(I_i, \sigma), & \text{if } I_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

Alpha-blending values are calculated similarly to 2DGS and take into account K_x overlapping planes at pixel x . We use the maximum impact threshold $\sigma = 500$ to regularize only clearly visible billboards. The choice of the threshold is discussed in the Sec. 6.2 of supp. mat.. By ignoring billboards with $I_i \leq 0$, we do not regularize invisible in the frame primitives, avoiding over-regularization in underrepresented areas. The proposed criterion is more efficient than the one utilizing projected radii as in 2DGS since they only take hitting frustum into account and ignore overlapping.

Finally, we define the regularization term to push RGB textures close to zero and alpha-maps close to a Gaussian distribution \mathcal{G} based on their visibility weight w_i :

$$\mathcal{L}_{\text{RGB}} = \frac{1}{N} \sum_{i=0}^N w_i \|T_i^{\text{RGB}}\|, \quad (9)$$

$$\mathcal{L}_\alpha = \frac{1}{N} \sum_{i=0}^N w_i \|T_i^\alpha - \mathcal{G}\|,$$

$$\mathcal{L}_{\text{texture}} = \lambda_{\text{RGB}} \mathcal{L}_{\text{RGB}} + \lambda_\alpha \mathcal{L}_\alpha.$$

With this regularization, the final loss to train BBSplat 3D scene representation is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{image}} + \mathcal{L}_{\text{texture}}. \quad (10)$$

3.4. Adaptive density control

Most 3DGS-based methods use adaptive density control to adjust the number of Gaussians. Specifically, they perform splitting, cloning, and pruning operations according to user-set policies. An alternative approach is proposed in [24], where they reformulate Gaussian Splatting densification as a Markov Chain Monte Carlo (MCMC) sampling. MCMC

introduces noise in the training process and replaces splitting and cloning with a deterministic state transition.

We also adopt and modify the MCMC sampling technique for BBSplat. In the sampling stage, MCMC clones Gaussians and adjusts their opacity o_i and scale set by co-variation matrix Σ_i to preserve the rendering output state:

$$o_{1,\dots,N}^{\text{new}} = 1 - \sqrt[N]{1 - o_N^{\text{old}}}, \quad (11)$$

$$\Sigma_{1,\dots,N}^{\text{new}} = (o_N^{\text{old}})^2 \left(\sum_{i=1}^N \sum_{k=0}^{i-1} \left(\binom{i-1}{k} \frac{(-1)^k (o_N^{\text{new}})^{k+1}}{\sqrt{k+1}} \right) \right)^{-2} \Sigma_N^{\text{old}}. \quad (12)$$

While this adjustment is well fit for Gaussians, we noticed that scale adjustment in Eq. (12) is unsuitable for arbitrarily shaped primitives and violates the rendering state preservation. Instead, we adjust only T_i^α to preserve overall transparency along the ray, because it controls transparency in all points on a plane and therefore does not require additional scale modification:

$$T_{1,\dots,N}^\alpha = 1 - \sqrt[N]{1 - T_N^\alpha}. \quad (13)$$

This renders parameters adjustment more straightforward compared to Eq. (12). Finally, where MCMC compares opacity o_i with a predefined threshold $\gamma = 5e-3$ to determine “dead” Gaussians, we compare the average value of the alpha-map: $\bar{T}_i^\alpha < \gamma$.

3.5. Texture compression

Using textured splats leads to new challenges of efficiently storing them. For each point, we use two textures of size $S_T \times S_T$ with one channel for transparency and three channels for color. Considering that textures are stored with 4-byte float point values, we need $(S_T * S_T + S_T * S_T * 3) * 4$ bytes of memory to store each texture.

To reduce storage costs, we first apply quantization [21] of texture parameters. Let us define normalized in $[0; 1]$ range texture values as ϵ . For storing BBSplat, we quantize texture values by rescaling $\hat{\epsilon} = \lfloor \epsilon * 255 \rfloor$ where $\lfloor \cdot \rfloor$ is the rounding operation. When loading textures, we apply an inverted operation $\epsilon = \hat{\epsilon}/255$ for dequantization. This way, we can store each texel value $\hat{\epsilon}$ as a single byte, reducing storage costs four times.

Our final color representation described in Eq. (6) as a sum of color calculated with SH and texture sampled offsets allows us to additionally reduce storage costs. Our regularization term $\mathcal{L}_{\text{texture}}$ proposed in Eq. (9) allows us to get sparser T^{RGB} values and T^α closer to a Gaussian distribution \mathcal{G} . We increase the sparsity of T^α by subtracting

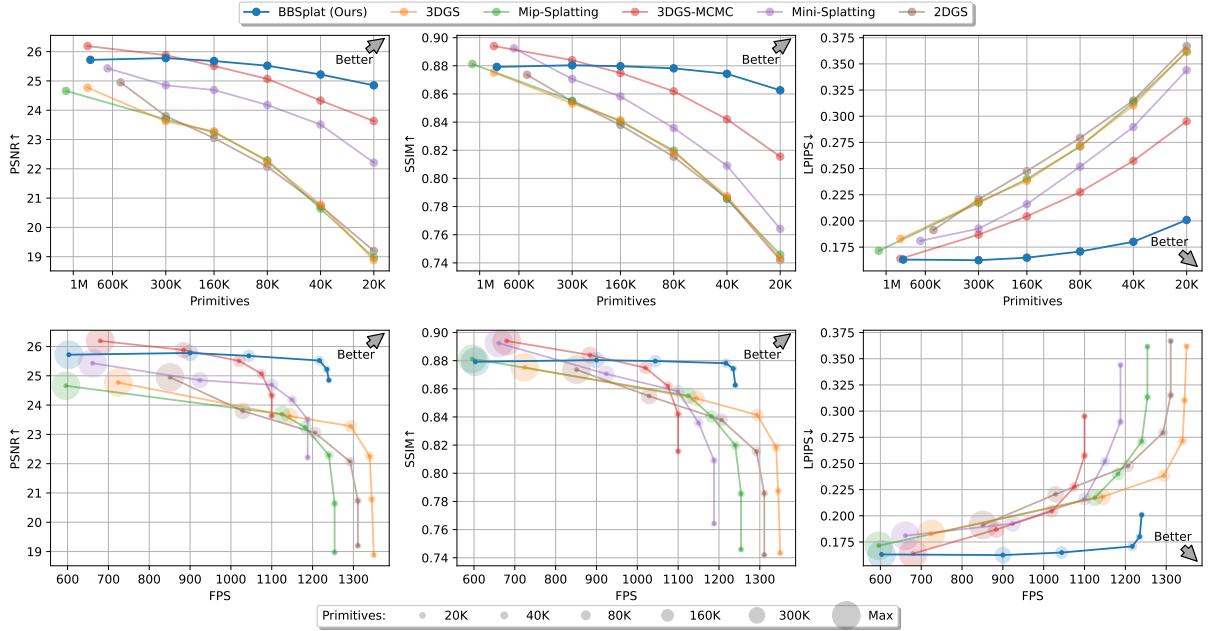


Figure 4. **Tanks&Temples metrics.** Top row: We report quantitative metrics values depending on the number of primitives for each method. BBSplat outperforms state-of-the-art for the lower number of primitives. Bottom row: We demonstrate that lower number of primitives (indicated by marker size) leads to rendering speed (frames per second - FPS) improvement. BBSplat demonstrates a better metrics/speed ratio for the NVS task (better results direction shown by the grey arrow).

the Gaussian pattern $T_\alpha - \mathcal{G}$ and ensure more zero values in the texture. The sparse nature of textures allows us to use efficient dictionary-based compression methods [11, 41] (e.g. ZIP [1]) to further reduce storage costs. Our texture compression pipeline allows us to reduce memory consumption ~ 7 times on average.

4. Experiments

We compared our billboard splatting representation against five state-of-the-art 3D scene representation methods: 3DGS [23], Mip-Splatting [52], 3DGS-MCMC [24], Mini-Splatting [10], and 2DGS [19]. For 2DGS, we conducted two sets of experiments: with and without depth-normal regularizations, as they affect metrics differently for various datasets. In the following section, we provide both quantitative and qualitative comparisons for indoor and outdoor real scenes from standard datasets: Tanks&Temples [26], Mip-NeRF-360 [5], and DTU [22]. Please check supplementary material for additional images, quantitative results, and further details on experimental implementation and evaluation.

4.1. Implementation

We use $\lambda_{\text{SSIM}} = 0.2$ for the structure similarity in the Equation (7), and $\lambda_{\text{RGB}} = \lambda_\alpha = 1e-4$ for the Equation (9). We initialize T_{RGB} and T_α with zeros and 2D Gaussian transparency respectively, and keep them fixed for the first 500 iterations to adjust orientations and colors initialized with

SfM. The densification strategy described in Sec. 3.4 is used from 500 until 25,000 iterations to adjust the number of billboards. Overall, we train BBSplat for 30,000 steps and finally, we fine-tune the spherical harmonics SH_i for an additional 2,000 iterations to adjust them for textures.

For training, we use the Adam [25] optimizer with hyper-parameters recommended for the Gaussian splatting pipelines [23]. We set RGB and α textures size $S_T = 16$, resulting in 16×16 textures to get the best trade-off between quality and memory requirements. To optimize RGB-texture we set learning rate $lr_{\text{RGB}} = 2.5e-3$ and for α -texture we set $lr_\alpha = 1e-3$. Additionally, we found it beneficial for convergence to increase spherical harmonics learning rate to $lr_{\text{SH}} = 5e-3$.

We implemented texture sampling with back-propagation in the CUDA code to achieve fast inference.

4.2. Results and Evaluation

We compared with previous state-of-the-art methods on real-world scenes using standard datasets. We evaluated our method on 4 indoor scenes from the Mip-NeRF-360 dataset [5] and 5 outdoor scenes from the Tanks&Temples dataset [26]. Additionally, we evaluated our method on 15 scenes from DTU dataset [22]. Overall, we provide experiments for 24 real scenes from three datasets.

We used train/test split for all datasets following 3DGS methodology where each 8th image was taken for the test.

	Tanks&Temples (outdoor)							DTU							Mip-Nerf-360 (indoor)						
	Points↓	FPS↑	Storage↓	PSNR↑	SSIM↑	LPIPS↓	Points↓	FPS↑	Storage↓	PSNR↑	SSIM↑	LPIPS↓	Points↓	FPS↑	Storage↓	PSNR↑	SSIM↑	LPIPS↓			
3DGS [23]	80K	1251 ± 187	19 MB	22.25	0.8185	0.2715	30K	1483 ± 228	7 MB	27.28	0.9115	0.2022	160K	1374 ± 127	38 MB	27.56	0.8978	0.1867			
Mip-Splatting [52]	80K	1162 ± 180	19 MB	22.29	0.8198	0.2711	30K	1355 ± 195	7 MB	26.89	0.9072	0.2045	160K	1254 ± 104	38 MB	27.25	0.8948	0.1899			
3DGS-MCMC [24]	80K	1158 ± 106	19 MB	25.07	0.8619	0.2276	30K	1186 ± 181	7 MB	30.96	0.9365	0.1741	160K	1109 ± 110	38 MB	30.71	0.9264	0.1417			
Mini-Splatting [10]	80K	1113 ± 152	19 MB	24.18	0.8358	0.2519	30K	1151 ± 188	7 MB	29.21	0.9236	0.1836	160K	1143 ± 102	38 MB	30.23	0.9178	0.1437			
2DGS [19]	80K	1269 ± 113	18 MB	17.83	0.6918	0.4222	30K	1258 ± 191	7 MB	27.75	0.9105	0.2039	160K	1174 ± 83	37 MB	26.32	0.8755	0.2177			
2DGS† [19]	80K	1292 ± 105	18 MB	22.07	0.8154	0.2793	30K	1249 ± 187	7 MB	28.69	0.9184	0.1965	160K	1207 ± 84	37 MB	27.96	0.8971	0.1888			
Ours	80K	1217 ± 92	49 MB	25.52	0.8782	0.1708	30K	1197 ± 181	15 MB	30.14	0.9387	0.1472	160K	1093 ± 76	98 MB	31.33	0.9356	0.1078			

Table 1. **Quantitative results with the same number of primitives.** We report metrics for the following datasets: Tanks&Temples [26], Mip-NeRF-360 [5], and DTU [22]. We fixed a number of primitives for all methods to provide metrics for the same training scenario. †-denotes the method’s version without using depth-normal regularization.

	Tanks&Temples (outdoor)							DTU							Mip-Nerf-360 (indoor)						
	Points↓	FPS↑	Storage↓	PSNR↑	SSIM↑	LPIPS↓	Points↓	FPS↑	Storage↓	PSNR↑	SSIM↑	LPIPS↓	Points↓	FPS↑	Storage↓	PSNR↑	SSIM↑	LPIPS↓			
3DGS [23]	830K	724 ± 75	196 MB	24.77	0.8752	0.1828	300K	964 ± 135	71 MB	29.18	0.9335	0.1573	1233K	599 ± 35	292 MB	31.63	0.9369	0.1144			
Mip-Splatting [52]	1114K	596 ± 49	268 MB	24.66	0.8813	0.1714	431K	787 ± 95	104 MB	29.39	0.9333	0.1362	1562K	467 ± 19	375MB	31.72	0.9405	0.1025			
3DGS-MCMC [24]	830K	680 ± 35	196 MB	26.19	0.8940	0.1639	300K	833 ± 133	71 MB	31.41	0.9511	0.1143	1233K	519 ± 34	292 MB	32.30	0.9448	0.1013			
Mini-Splatting [10]	193K	992 ± 159	46 MB	24.62	0.8782	0.1776	215K	852 ± 142	51 MB	27.74	0.9257	0.1301	293K	1001 ± 78	70 MB	30.97	0.9352	0.1105			
2DGS [19]	460K	947 ± 80	107 MB	21.75	0.7807	0.3091	151K	1101 ± 194	35 MB	29.83	0.9339	0.1628	764K	715 ± 46	163 MB	30.26	0.9178	0.1519			
2DGS† [19]	542K	851 ± 72	126 MB	24.95	0.8736	0.1912	135K	1109 ± 193	31 MB	29.85	0.9343	0.1612	674K	763 ± 51	157 MB	31.37	0.9319	0.1243			
Ours-small	160K	1044 ± 89	93 MB	25.68	0.8797	0.1649	30K	1197 ± 181	15 MB	30.14	0.9387	0.1472	160K	1093 ± 76	98 MB	31.33	0.9356	0.1078			
Ours-big	300K	900 ± 95	163 MB	25.78	0.8804	0.1625	60K	1178 ± 180	29 MB	30.10	0.9371	0.1449	300K	920 ± 69	173 MB	31.51	0.9371	0.1044			

Table 2. **Quantitative results with a large number of primitives.** We report metrics for the following datasets: Tanks&Temples [26], Mip-NeRF-360 [5], and DTU [22]. We provide metrics for the recommended number of Gaussians for each method, which results in lower FPS and increased storage space. †- indicates method version without using depth-normal regularizations.

	PSNR↑	L1↓	LPIPS↓	FPS↑
Texture-GS	30.03	0.0135	0.1440	267
Ours-small	30.81	0.0123	0.1055	1230

Table 3. **Comparison with Texture-GS [49] on DTU.** While Texture-GS also applies textures for Gaussians, it does not perform well for NVS.

For all three datasets, we used images downsampled to 800 pixels on the larger side. All experiments are conducted on a single NVIDIA GeForce RTX 4090 GPU.

Quantitative comparison. We report PSNR, SSIM, and LPIPS [55] metrics (Tabs. 1 and 2) widely used to evaluate quality in the NVS task. PSNR and SSIM are traditionally used to demonstrate similarity with ground truth images; LPIPS utilize neural networks and is known for the best correlation with human perception. To assess model efficiency, we also report rendering speed as frames per second (FPS) and required storage space along with a number of used primitives. In the provided experiments, we evaluated metrics for different numbers of primitives by limiting the growth of their number during training as proposed in [6].

In an experiment in Table 1, we showcase metrics for the fixed number of primitives depending only on their type. For the same amount of primitives, our method reaches better PSNR, SSIM and LPIPS than state-of-the-art Gaussian-based representations both for indoor and outdoor scenes. While improving state-of-the-art objective metrics, for the same number of primitives BBSplat requires more storage space. In the next experiment, we demonstrate inference speed and storage efficiency for on-par metrics.

In Table 2, we compare against other methods for the

maximum recommended amount of Gaussians to achieve the best possible quality for each method. For our method, we provide two scenarios with fewer (*small*) and more (*big*) billboards. We outperform 3DGS and 2DGS for all three metrics, at the same time showing inference speed and storage space improvement. BBSplat demonstrates metrics on par with the best competitor, 3DGS-MCMC, achieving up to two times faster rendering of the scenes with three times lower storage space requirement.

In Table 3, we compare with Texture-GS [49] as it also proposes to employ textures, although in conjunction with a 3DGS model. The main objective of Texture-GS is texture editing: indeed we verify that Texture-GS performance are reduced for the NVS task, especially in FPS count. We report metrics only for the DTU dataset since Texture-GS is unable to process large scenes, as it is limited by spherical texture space definition.

In Figure 4, we show the metrics as a function of the number of primitives (Gaussians or BBSplats) and as a function of FPS. To plot the leftmost point we used the maximum number of Gaussians requested by each method, then for subsequent points we fixed the number of primitives for all scenes at the chosen value. It can be seen that our method gives a flatter curve of metrics change, which means a minimal decrease in metrics when the number of primitives decreases to 80K. Thus, the proposed implementation allows us to achieve an inference speed of up to 1250 FPS while maintaining high rendering quality.

Qualitative comparison. We provide qualitative results in Figure 5 for indoor and outdoor scenes from different datasets. We show novel views for basic 3DGS and 2DGS

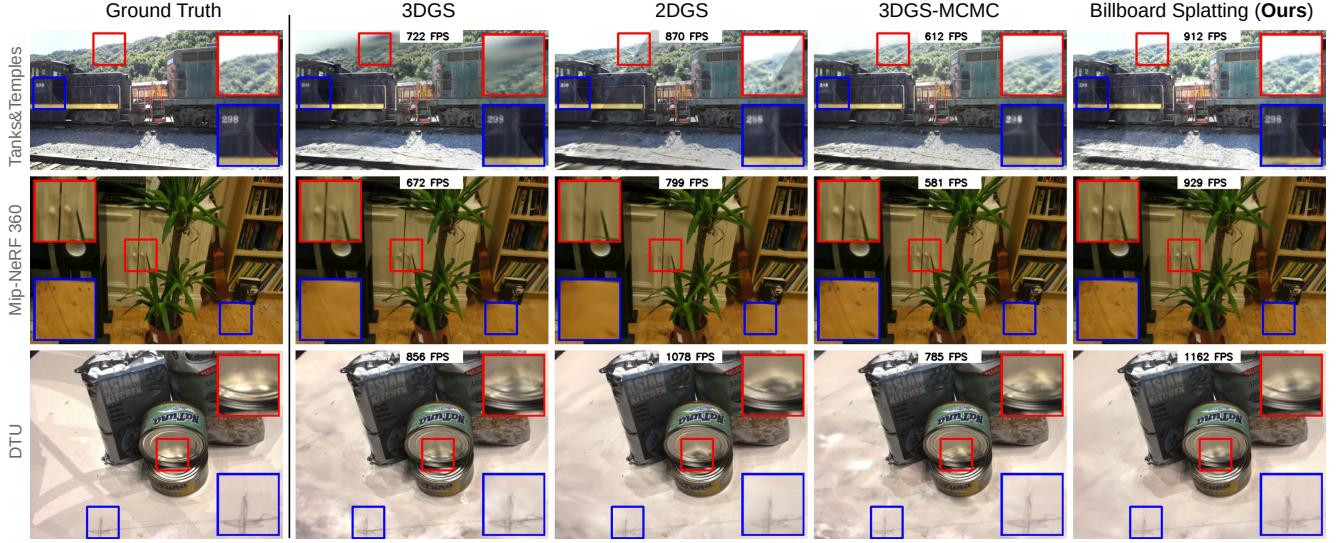


Figure 5. **Qualitative results.** We provide rendering results of scenes from each dataset: Tanks&Temples [Train], Mip-NeRF-360 [Room], and DTU [Scan97]. For competitors, we use the maximum recommended by the method number of Gaussians. Our results are reported for 300K billboards. More renderings can be found in the Sec. 7.1 of supp. mat..

	PSNR↑	SSIM↑	LPIPS↓	Storage↓
w/o T^{RGB}	25.17	0.8684	0.1774	60 MB
w/o T^α	25.26	0.8710	0.1858	72 MB
w/o $\mathcal{L}_{\text{texture}}$	25.54	0.8791	0.1655	131 MB
w/o compress	25.71	0.8818	0.1630	662 MB
Full	25.68	0.8797	0.1649	93 MB

Table 4. **Ablation study on Tanks&Temples dataset.** We report the quantitative metrics for our method without the use of color or transparency textures, and without texture regularization and compression post-processing.

representations and 3DGS-MCMC as they are the best scoring competitors. While the quality of the central object of the scene remains similar for all methods, in most scenarios BBSplat represents background objects with more details. For Tanks&Temples and Mip-NeRF-360 dataset, we reached quality on par with state-of-the-art while providing faster inference speed and higher level of detail. For the DTU dataset, we also demonstrate better handling of challenging shadow effects. For more results, please refer to Sec. 7.1 of supp. mat..

4.3. Ablation study

In Table 4, we provide an ablation study of different aspects of our approach. Namely the use of RGB and alpha textures, texture regularization, and compression post-processing.

The use of both alpha and RGB textures is essential for rendering quality. Excluding T^α significantly increases LPIPS, while excluding T^{RGB} reduces PSNR and SSIM. Texture regularization term preserves over-fitting and leads to additional metrics improvement. It also encourages spar-

sity in textures and reduces storage costs in combination with compression. The use of compression leads to a slight metric reduction but allows to significantly reduce storage space requirements. Therefore, all parts of the proposed approach are essential to get the best result. However, we note that metrics could be additionally improved by disabling compression.

5. Conclusion

We have proposed BBSplat - a novel method for 3D scene representation. The proposed method uses textured primitives of arbitrary learnable shapes and allows a two times faster generation of photorealistic novel views. We developed specialized regularization term and compression technique to reduce storage space by leveraging the sparse nature of the proposed textured representation. In extensive experiments on real data, we demonstrated the efficiency of the proposed method both quantitatively and qualitatively. In particular, we showed the best trade-off between objective metrics and inference speed.

Limitations and future work. One of the main limitations of the proposed method is storage space. While we make a significant step towards its reduction, there is still room for improvement in future works. Another limitation is training time, as it takes about 40 minutes to fit a scene, compared with 5 minutes for 3DGS. This slowdown is caused by backpropagation to the textures, and while it still significantly outperforms NeRF-based methods in terms of speed, it could be a bottleneck for some applications. In future works, we are going to focus on resolving these limitations.

Acknowledgements. We thank Matteo Toso, Milind Gajanan Padalkar, Matteo Bortolon, and Fabio Poiesi for their feedback and helpful suggestions on improving the paper. We also thank Andrea Tagliasacchi for insightful discussions and Matteo Bortolon for the help with CUDA implementation. D. Svitov thanks the ELLIS network for organizing the PhD program that made this work possible.

References

- [1] Zip file format specification. <https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.3.10.TXT>. Online. 6
- [2] Edward H Adelson, James R Bergen, et al. *The plenoptic function and the elements of early vision*. Vision and Modeling Group, Media Laboratory, Massachusetts Institute of . . . , 1991. 2, 3, 4
- [3] Guanbo Bao, Xiaopeng Zhang, Wujun Che, and Marc Jaeger. Billboards for tree simplification and real-time forest rendering. In *2009 Third International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications*, pages 433–440. IEEE, 2009. 3
- [4] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, pages 5855–5864, 2021. 1
- [5] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 1, 6, 7, 3
- [6] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kortschieder. Revising densification in gaussian splatting. *ECCV*, 2024. 2, 7
- [7] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorrf: Tensorial radiance fields. In *ECCV*, pages 333–350. Springer, 2022. 1
- [8] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *CVPR*, pages 16569–16578, 2023. 1
- [9] Xavier Décoret, Frédéric Durand, François X Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In *SIGGRAPH*, pages 689–696. 2003. 2, 3
- [10] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. 2024. 1, 6, 7
- [11] Robert M Fano. *The transmission of information*. Massachusetts Institute of Technology, Research Laboratory of Electronics . . . , 1949. 2, 6
- [12] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, pages 5501–5510, 2022. 3
- [13] Yang Fu, Sifei Liu, Amey Kulkarni, Jan Kautz, Alexei A. Efros, and Xiaolong Wang. Colmap-free 3d gaussian splatting. *CVPR*, pages 20796–20805, 2023. 1
- [14] Anton L Fuhrmann, Eike Umlauf, and Stephan Mantler. Extreme model simplification for forest rendering. *NPH*, 5:57–67, 2005. 3
- [15] Simon Fuhrmann, Fabian Langguth, and Michael Goesele. Mve-a multi-view reconstruction environment. *GCH*, 3:4, 2014. 2
- [16] Álvaro González. Measurement of areas on a sphere using fibonacci and latitude-longitude lattices. *Mathematical geosciences*, 42:49–64, 2010. 4, 1, 2
- [17] Mathias Holst and Heidrun Schumann. Surfel-based billboard hierarchies for fast rendering of 3d-objects. In *PBG@Eurographics*, pages 109–118, 2007. 3
- [18] Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields. In *ICCV*, pages 19774–19783, 2023. 1
- [19] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH*. Association for Computing Machinery, 2024. 1, 2, 3, 6, 7
- [20] Zhentao Huang and Minglun Gong. Textured-gs: Gaussian splatting with spatially defined color and opacity. 2024. 2
- [21] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, pages 2704–2713, 2018. 5
- [22] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *CVPR*, pages 406–413. IEEE, 2014. 6, 7, 3
- [23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *TOG*, 42(4):1–14, 2023. 1, 2, 6, 7, 3
- [24] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. *NIPS*, 2024. 1, 2, 5, 6, 7, 3
- [25] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6
- [26] Arno Knapsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *TOG*, 36(4), 2017. 6, 7, 3
- [27] J Dylan Lacewell, Dave Edwards, Peter Shirley, and William B Thompson. Stochastic billboard clouds for interactive foliage rendering. *Journal of graphics tools*, 11(1):1–12, 2006. 3
- [28] Tuong-Vu Le-Nguyen, Kok-Lim Low, Conrado Ruiz, and Sang N Le. Automatic paper sliceform design from 3d solid models. *TVCG*, 19(11):1795–1807, 2013. 3
- [29] Jiahui Lei, Yufu Wang, Georgios Pavlakos, Lingjie Liu, and Kostas Daniilidis. Gart: Gaussian articulated template models. In *CVPR*, pages 19876–19887, 2024. 2
- [30] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *TOG*, 38(4):65:1–65:14, 2019. 1, 2
- [31] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *TOG*, 40(4), 2021. 1

- [32] Hidenobu Matsuki, Riku Murai, Paul H. J. Kelly, and Andrew J. Davison. Gaussian Splatting SLAM. 2024. 2
- [33] James McCrae, Karan Singh, and Niloy J Mitra. Slices: a shape-proxy based on planar sections. *TOG*, 30(6):168, 2011. 3
- [34] 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. *Communications of the ACM*, 65(1):99–106, 2021. 1, 2
- [35] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *TOG*, 41(4):102:1–102:15, 2022. 1, 2
- [36] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radspat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. *arXiv.org*, 2024. 1, 2
- [37] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, pages 165–174, 2019. 2
- [38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 4, 1
- [39] Conrado R Ruiz Jr, Sang N Le, Jinze Yu, and Kok-Lim Low. Multi-style paper pop-up designs from 3d models. In *Computer Graphics Forum*, pages 487–496. Wiley Online Library, 2014. 3
- [40] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *CVPR*, 2016. 2, 3, 4
- [41] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948. 2, 6
- [42] Christian Sigg, Tim Weyrich, Mario Botsch, and Markus H Gross. Gpu-based ray-casting of quadratic surfaces. In *PBG@SIGGRAPH*, pages 59–65, 2006. 3, 4
- [43] PR Smith. Bilinear interpolation of digital images. *Ultramicroscopy*, 6(2):201–204, 1981. 4
- [44] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *SIGGRAPH*, pages 835–846. 2006. 2
- [45] Noah Snavely, Steven M Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International journal of computer vision*, 80:189–210, 2008. 2
- [46] David Svitov, Pietro Morerio, Lourdes Agapito, and Alessio Del Bue. Haha: Highly articulated gaussian human avatars with textured mesh prior. *ACCV*, 2024. 2
- [47] Daniel Watson, William Chan, Ricardo Martin Bralla, Jonathan Ho, Andrea Tagliasacchi, and Mohammad Norouzi. Novel view synthesis with diffusion models. In *ICLR*, 2023. 1
- [48] Tim Weyrich, Simon Heinze, Timo Aila, Daniel B Fasnacht, Stephan Oetiker, Mario Botsch, Cyril Flraig, Simon Mall, Kaspar Rohrer, Norbert Felber, et al. A hardware architecture for surface splatting. *TOG*, 26(3):90–es, 2007. 3
- [49] Tian-Xing Xu, Wenbo Hu, Yu-Kun Lai, Ying Shan, and Song-Hai Zhang. Texture-gs: Disentangling the geometry and texture for 3d gaussian splatting editing. *ECCV*, 2024. 2, 4, 7
- [50] Yuelang Xu, Benwang Chen, Zhe Li, Hongwen Zhang, Lizhen Wang, Zerong Zheng, and Yebin Liu. Gaussian head avatar: Ultra high-fidelity head avatar via dynamic gaussians. In *CVPR*, 2024. 2
- [51] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxtels: Radiance fields without neural networks. *CVPR*, pages 5491–5500, 2021. 1
- [52] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. *CVPR*, 2024. 1, 2, 6, 7
- [53] Vladimir Yugay, Yue Li, Theo Gevers, and Martin R. Oswald. Gaussian-slam: Photo-realistic dense slam with gaussian splatting, 2023. 2
- [54] Jiahui Zhang, Fangneng Zhan, Muyu Xu, Shijian Lu, and Eric Xing. Freg3: 3d gaussian splatting with progressive frequency regularization. In *CVPR*, pages 21424–21433, 2024. 1
- [55] 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
- [56] Liang Zhao, Shoudong Huang, and Gamini Dissanayake. Linear sfm: A hierarchical approach to solving structure-from-motion problems by decoupling the linear and nonlinear components. *ISPRS journal of photogrammetry and remote sensing*, 141:275–289, 2018. 2

BillBoard Splatting (BBSplat): Learnable Textured Primitives for Novel View Synthesis

Supplementary Material

	10	100	500	1000	1500
PSNR↑	25.95	25.96	25.99	25.92	25.92
SSIM↑	0.8800	0.8798	0.8798	0.8798	0.8795
LPIPS↓	0.1365	0.1375	0.1379	0.1392	0.1396
Size(MB)↓	139	122	102	92	87
FPS↑	995	1033	1067	1081	985

Table 5. **Selecting σ threshold value.** We search for the best σ threshold in Equation (9) using “Truck” scene from the Tanks&Temples dataset.

	$8^2 \times 640K$	$16^2 \times 160K$	$32^2 \times 40K$	$64^2 \times 10K$
PSNR↑	26.04	25.99	25.28	23.34
SSIM↑	0.8807	0.8798	0.8662	0.8000
LPIPS↓	0.1393	0.1379	0.1505	0.2382
Size(MB)↓	207	102	82	68
FPS↑	618	1067	1211	1267

Table 6. **Selecting texture size and number of primitives.** We search for best texture size and primitives number for the fixed parameters amount (4 channels x 40.96M). We evaluate metrics using the “Truck” scene from the Tanks&Temples dataset.

6. Implementation details

In this section, we provide more details on the method’s implementation. In the following subsections, we describe the implementation of backpropagation for the bilinear texture sampling with billboard position adjustment and discuss the choice of hyper-parameters to train billboard splatting.

6.1. Texture sampling

We leverage *bilinear sampling* to get texture values from T_i^{RGB} and T_i^α in the $\mathbf{u} = (u, v)$ point of a billboard. Our implementation utilizes GPU programmed with CUDA to accelerate sampling and loss backpropagation to the texture. We based our implementation on the PyTorch[38] CUDA code for bilinear sampling and integrated it into 2DGS CUDA implementation as described below.

In Algorithm 1, we provide pseudo-code for the bilinear sampling gradients backpropagation. The algorithm accepts the gradient value that should be distributed among neighboring texels, (u, v) coordinate of the billboard point where the gradient is calculated, corresponding source texture, and output tensor to store texture gradients. The function redistributes and stores weighted values of the gradient in four neighboring texels and calculate the gradient $dL/d\mathbf{u}$ with respect to point (u, v) position.

In Algorithm 2, we demonstrate how we prepare input

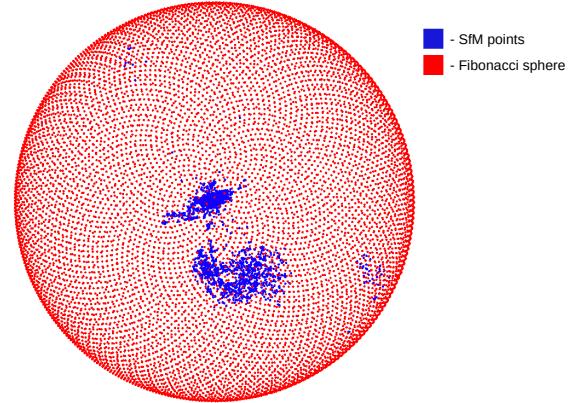


Figure 6. **Sky-sphere around the scene.** We sample additional points on the sphere around all SfM points with the Fibonacci algorithm [16].

parameters for Algorithm 1 to get $dL/d\mathbf{u}$ values based on RGB texture and alpha map. Calculated $dL/d\mathbf{u}$ value is then used to adjust billboard position and rotation. The proposed implementation can be easily integrated with the 2DGS approach, which we encapsulate here as subprogram \mathcal{F} . The function \mathcal{F} encapsulates gradients calculation for transformation matrix Σ encoding translation and rotation of the plane.

6.2. Hyper-parameters

We apply learning rate exponential decay for billboards position from $lr_\mu = 1.6e-4$ to $lr_\mu = 1.6e-6$ with 0.01 multiplier. To optimize spherical harmonics and scales we set $lr_{\text{SH}} = ls_s = 5e-3$ while using smaller learning rate for rotation $le_r = 1e-3$.

We select σ threshold value in Equation (9) based on the metrics provided in Table 5. The σ threshold is used to determine the minimum billboard visibility from which we start regularizing its textures. We set $\sigma = 500$ as it provides the best PSNR value and achieves the best trade-off between other metrics and storage space. Over-regularization results in more billboards having Gaussian distributed transparency, which leads to more compact storing of them but reduces NVS quality. Under-regularization allows to achieve higher SSIM and LPIPS at the cost of storage space and inference speed.

Table 6 shows metrics for different texture sizes and number of billboards. We select texture size to be the power of two (8, 16, 32, 64) as it is more suitable for processing

Algorithm 1 Function for bilinear gradient redistribution

```

function BILINEAR( $\frac{dL}{d\omega}$ ,  $(u, v)$ ,  $T$ ,  $\frac{dL}{dT}$ )
     $x = ((u + 1)/2)(T_S - 1)$ 
     $y = ((v + 1)/2)(T_S - 1)$ 
     $ne_x = \lfloor x \rfloor + 1 \quad se_x = \lfloor x \rfloor + 1 \quad sw_x = \lfloor x \rfloor$ 
     $ne_y = \lfloor y \rfloor \quad se_y = \lfloor y \rfloor + 1 \quad sw_y = \lfloor y \rfloor + 1$ 
     $nw = (se_x - x)(se_y - y) \quad ne = (x - sw_x)(sw_y - y)$ 
     $sw = (ne_x - x)(y - ne_y) \quad se = (x - nw_x)(y - nw_y)$ 

    Redistribute gradients
     $\frac{dL}{dT}[nw_x, nw_y] = \frac{dL}{d\omega} \cdot nw$ 
     $\frac{dL}{dT}[ne_x, ne_y] = \frac{dL}{d\omega} \cdot ne$ 
     $\frac{dL}{dT}[sw_x, sw_y] = \frac{dL}{d\omega} \cdot sw$ 
     $\frac{dL}{dT}[se_x, se_y] = \frac{dL}{d\omega} \cdot se$ 

    Calculate position gradient
     $g_x = -T[nw_x, nw_y](se_y - y)\frac{dL}{d\omega}$ 
     $g_y = -T[nw_x, nw_y](se_y - x)\frac{dL}{d\omega}$ 
     $g_x += T[ne_x, ne_y](sw_y - y)\frac{dL}{d\omega}$ 
     $g_y -= T[ne_x, ne_y](x - sw_x)\frac{dL}{d\omega}$ 
     $g_x -= T[sw_x, sw_y](y - ne_y)\frac{dL}{d\omega}$ 
     $g_y += T[sw_x, sw_y](ne_x - x)\frac{dL}{d\omega}$ 
     $g_x += T[se_x, se_y](y - nw_y)\frac{dL}{d\omega}$ 
     $g_y += T[se_x, se_y](x - nw_x)\frac{dL}{d\omega}$ 
     $g_x = g_x(T_s - 1)/2 \quad g_y = g_y(T_s - 1)/2$ 

    return  $(g_x, g_y)$  ▷ Values for  $\frac{dL}{du}$ 
end function

```

with GPU, and set the corresponding number of billboards to result in 40.96M parameters for one texture channel to not exceed GPU capacity. While 32×32 and 16×16 both provide suitable metrics, we chose 16×16 as slightly more accurate. As an alternative, 32×32 can be used to achieve faster inference.

6.3. Initialization

In scenarios when we need fewer SfM points to initialize a small number of billboards, we subsample them with an iterative farthest point sampling strategy. For outdoor scenes, we optionally add 10K evenly distributed points on the large sphere using the Fibonacci algorithm [16] to represent the sky and far away objects. We select sphere radius as the distance from the scene center to the furthest SfM point to guarantee that all SfM points are included in the sphere. For scenarios when we use 20K points and less, we reduce the number of sphere points to 2K, and completely disable it when we use less than 10K points. In Figure 6, we provide a visualization of such a sphere.

Algorithm 2 Billboards position optimization

```

Require:  $\frac{dL}{dT}$  ▷ Gradient of loss  $L$  w.r.t image  $I$ 
Require:  $\mathbf{u}$  ▷ Plane intersection  $(u, v)$  point
Require:  $\mathcal{B}$  ▷ Billboards sorted along the ray
Require:  $\mathcal{T}$  ▷ Forward pass accumulated transparency

for  $i \in \mathcal{B}$  do
    Sample alpha and color
     $\alpha = T_i^\alpha[\mathbf{u}]$ 
     $c = T_i^{\text{RGB}}[\mathbf{u}] + \text{SH}_i$ 
     $\hat{c} = \alpha' c' + (1 - \alpha')c$  ▷ Accumulated color

    Update transparency
     $\mathcal{T} = \mathcal{T}/(1 - \alpha)$ 

    Calculate gradients
     $\frac{dL}{dc} = \alpha \mathcal{T} \frac{dL}{dT}$ 
     $\frac{dL}{d\alpha} = (c - \hat{c}) \mathcal{T} \frac{dL}{dT}$ 

    Redistribute gradients
     $\frac{dL}{du} = \text{BILINEAR}(\frac{dL}{dc}, \mathbf{u}, T_i^{\text{RGB}}, \frac{dL}{dT})$ 
     $\frac{dL}{du} += \text{BILINEAR}(\frac{dL}{d\alpha}, \mathbf{u}, T_i^\alpha, \frac{dL}{dT})$ 

    Adjust position
     $\frac{dL}{d\Sigma} = \mathcal{F}(\frac{dL}{du})$  ▷ Calculate  $\frac{dL}{d\Sigma}$  as in 2DGs
     $\alpha' = \alpha$ 
     $c' = c$ 
end for

```

7. Additional details on experiments and results

In this section, we provide more qualitative results for our method along with per-scene quantitative metrics for averaged values reported in the paper.

7.1. Visual results

In Figure 8, we provide an additional visual comparison of BBSplat with state-of-the-art NVS methods. Our renderings demonstrate exceeding quality for similar or higher FPS.

Figure 7 demonstrates BBSplat training progress for 500, 5000, 15000, and 30000 iterations, showing the change of transparency from Gaussian distribution to arbitrary shapes.

7.2. Detailed results

In Table 7, we report metrics for all scenes. For competitors, we provide metrics for the maximum recommended number of Gaussians, and for ours with 300K billboards. The averaged values for these three datasets are reported in the main paper. For examples of renderings from different angles, see the video in the supp. mat..

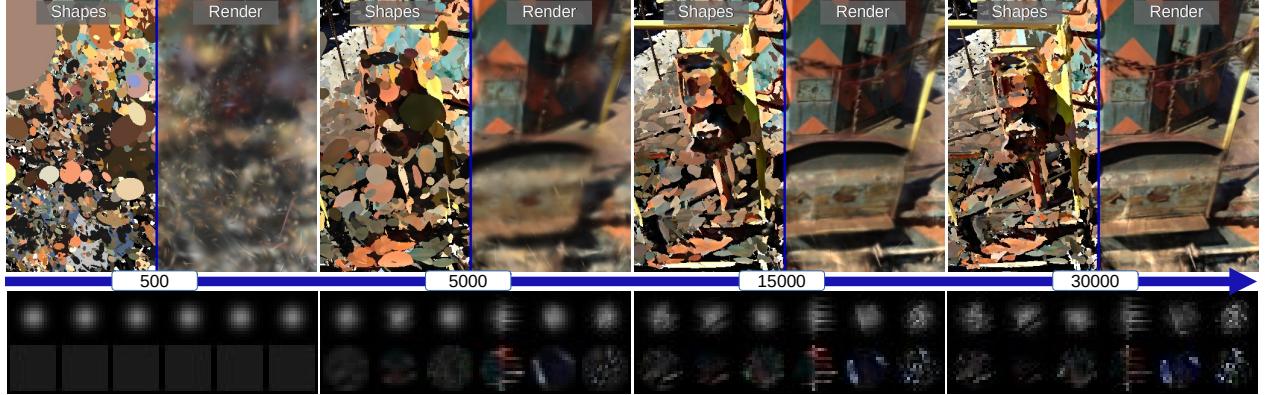


Figure 7. **BBsplat optimization process.** Top: We demonstrate billboard silhouettes and rendering results during the training process for 500, 5000, 15000, and 30000 iterations. To visualize silhouettes we disabled color textures and cut alpha-maps to the threshold. Bottom: We show examples of alpha-maps and RGB textures corresponding to training steps.

	3DGS [23]	3DGS-MCMC [24]	2DGS† [19]	BBsplat (Ours)	
	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	
Tanks&Temples	Train	22.57 / 0.8248 / 0.1968	23.26 / 0.8531 / 0.1735	22.03 / 0.8146 / 0.2144	23.04 / 0.8336 / 0.1696
	Truck	26.04 / 0.8851 / 0.1489	27.23 / 0.9024 / 0.1230	26.07 / 0.8814 / 0.1612	26.06 / 0.8800 / 0.1336
	Francis	28.00 / 0.9127 / 0.2349	29.24 / 0.9217 / 0.2275	28.23 / 0.9118 / 0.2424	30.08 / 0.9231 / 0.1960
	Horse	25.26 / 0.9054 / 0.1272	27.53 / 0.9227 / 0.1108	25.70 / 0.9044 / 0.1321	25.85 / 0.9027 / 0.1311
	Lighthouse	21.99 / 0.8482 / 0.2061	23.71 / 0.8701 / 0.1846	22.70 / 0.8556 / 0.2061	23.85 / 0.8627 / 0.1822
Mip-NeRF-360	Room	32.31 / 0.9401 / 0.1295	33.19 / 0.9507 / 0.1075	32.33 / 0.9370 / 0.1365	32.30 / 0.9454 / 0.1062
	Kitchen	32.12 / 0.9370 / 0.1114	32.93 / 0.9429 / 0.1056	31.50 / 0.9303 / 0.1248	32.00 / 0.9365 / 0.1084
	Bonsai	32.26 / 0.9495 / 0.0977	32.77 / 0.9554 / 0.0861	32.03 / 0.9460 / 0.1055	31.85 / 0.9464 / 0.0940
	Counter	29.84 / 0.9210 / 0.1191	30.30 / 0.9303 / 0.1059	29.60 / 0.9142 / 0.1302	29.88 / 0.9201 / 0.1091
DTU	Scan24	25.64 / 0.9399 / 0.0780	29.28 / 0.9531 / 0.0605	26.59 / 0.9402 / 0.0889	26.89 / 0.9332 / 0.0914
	Scan37	25.81 / 0.9285 / 0.0973	27.58 / 0.9448 / 0.0757	25.81 / 0.9341 / 0.0870	24.73 / 0.9165 / 0.1019
	Scan40	24.73 / 0.9278 / 0.1126	29.35 / 0.9528 / 0.0776	26.04 / 0.9218 / 0.1165	26.92 / 0.9323 / 0.0992
	Scan55	28.48 / 0.9199 / 0.1750	30.66 / 0.9470 / 0.1093	28.59 / 0.9187 / 0.1834	29.50 / 0.9322 / 0.1331
	Scan63	30.48 / 0.9677 / 0.0577	32.99 / 0.9767 / 0.0434	31.62 / 0.9671 / 0.0567	29.51 / 0.9528 / 0.0661
	Scan65	27.69 / 0.9131 / 0.1913	30.65 / 0.9417 / 0.1332	29.30 / 0.9191 / 0.1982	29.59 / 0.9278 / 0.1476
	Scan69	26.88 / 0.9142 / 0.1564	27.85 / 0.9306 / 0.1266	27.26 / 0.9154 / 0.1624	26.79 / 0.9168 / 0.1390
	Scan83	29.64 / 0.9397 / 0.1876	30.59 / 0.9468 / 0.1670	29.69 / 0.9336 / 0.1975	30.08 / 0.9409 / 0.1831
	Scan97	26.84 / 0.9180 / 0.1670	27.68 / 0.9246 / 0.1588	27.03 / 0.9172 / 0.1742	28.52 / 0.9272 / 0.1600
	Scan105	30.23 / 0.9414 / 0.1511	32.26 / 0.9507 / 0.1381	30.06 / 0.9364 / 0.1659	31.23 / 0.9465 / 0.1479
	Scan106	32.93 / 0.9443 / 0.1828	35.33 / 0.9628 / 0.1133	33.89 / 0.9469 / 0.1851	34.14 / 0.9502 / 0.1764
	Scan110	31.67 / 0.9396 / 0.1945	33.49 / 0.9553 / 0.1530	31.88 / 0.9391 / 0.2011	32.52 / 0.9433 / 0.1812
	Scan114	29.76 / 0.9266 / 0.1861	31.50 / 0.9518 / 0.1091	31.26 / 0.9366 / 0.1792	31.28 / 0.9383 / 0.1703
	Scan118	33.40 / 0.9376 / 0.2074	36.00 / 0.9647 / 0.1142	34.40 / 0.9442 / 0.2063	35.37 / 0.9503 / 0.1813
	Scan122	33.58 / 0.9436 / 0.2151	35.88 / 0.9626 / 0.1343	34.42 / 0.9435 / 0.2161	34.34 / 0.9487 / 0.1943

Table 7. **Quantitative metrics for all scenes.** We report PSNR, SSIM, and LPIPS for each scene in all three datasets: Tanks&Temples [26], Mip-NeRF-360 [5], and DTU [22]. For comparison, we provide metrics for basic 3DGS [23] and 2DGS [19] methods, and for 3DGS-MCMC [24] as the best-scoring competitor.

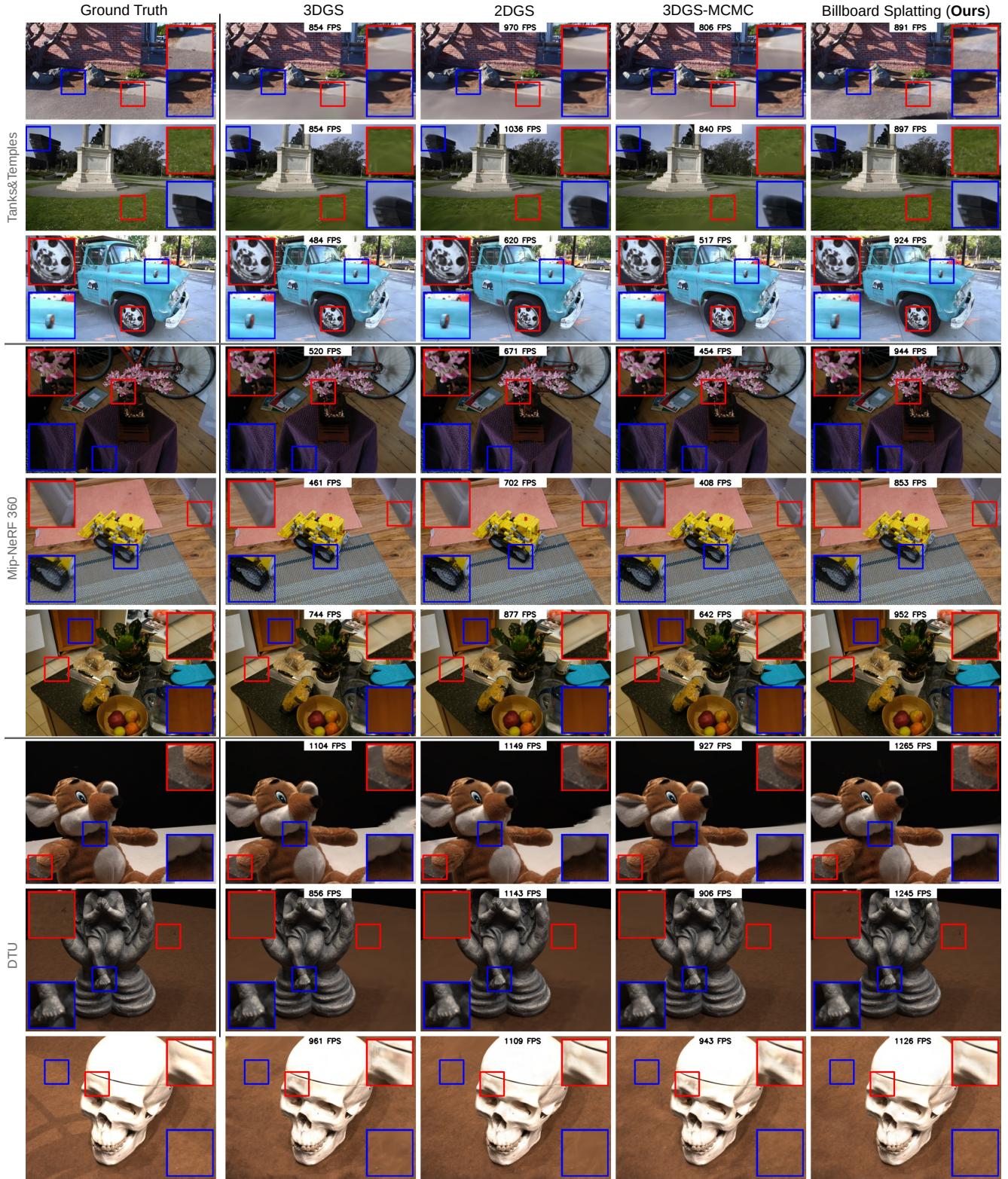


Figure 8. **Qualitative results.** We provide rendering results of three more scenes from each dataset: Tanks&Temples [Lighthouse, Francis, Truck], Mip-NeRF-360 [Bansai, Kitchen, Counter], and DTU [Scan105, Scan118, Scan65]. For competitors, we use the maximum number of Gaussians recommended by the method. Our results are reported for 300K billboards.