

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

Table 6. **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

Table 7. **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[41] 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 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

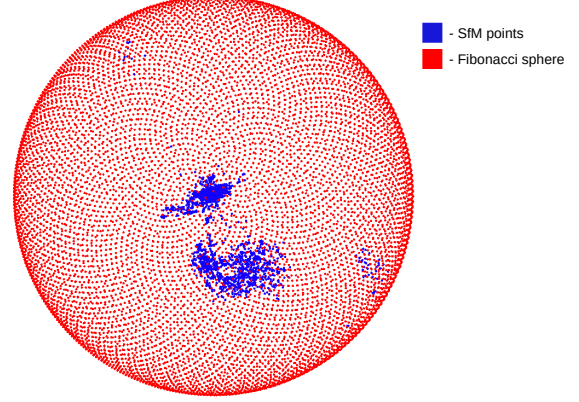


Figure 6. **Sky-sphere around the scene.** We sample additional points on the sphere around all SfM points with the Fibonacci algorithm [16].

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 $lr_{\text{SH}} = lr_s = 5e-3$ while using smaller learning rate for rotation $lr_r = 1e-3$.

We select σ threshold value in Equation (9) based on the metrics provided in Table 6. 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 7 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 with GPU, and set the corresponding number of billboards to result in 40.96M parameters for one texture channel to

Algorithm 1 Function for bilinear gradient redistribution

```
function BILINEAR( $\frac{dL}{d\omega}, (u, v), T, \frac{dL}{dT}$ )  
   $x = ((u + 1)/2)(T_S - 1)$   
   $y = ((v + 1)/2)(T_S - 1)$   
  
   $ne_x = \lfloor x \rfloor + 1$     $se_x = \lfloor x \rfloor + 1$     $sw_x = \lfloor x \rfloor$   
   $ne_y = \lfloor y \rfloor$       $se_y = \lfloor y \rfloor + 1$     $sw_y = \lfloor y \rfloor + 1$   
  
   $nw = (se_x - x)(se_y - y)$     $ne = (x - sw_x)(sw_y - y)$   
   $sw = (ne_x - x)(y - ne_y)$     $se = (x - nw_x)(y - nw_y)$   
  
  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  
   $g_x = -T[nw_x, nw_y](se_y - y)\frac{dL}{d\omega}$   
   $g_y = -T[nw_x, nw_y](se_y - x)\frac{dL}{d\omega}$   
   $g_x += T[ne_x, ne_y](sw_y - y)\frac{dL}{d\omega}$   
   $g_y -= T[ne_x, ne_y](x - sw_x)\frac{dL}{d\omega}$   
   $g_x -= T[sw_x, sw_y](y - ne_y)\frac{dL}{d\omega}$   
   $g_y += T[sw_x, sw_y](ne_x - x)\frac{dL}{d\omega}$   
   $g_x += T[se_x, se_y](y - nw_y)\frac{dL}{d\omega}$   
   $g_y += T[se_x, se_y](x - nw_x)\frac{dL}{d\omega}$   
  
   $g_x = g_x(T_S - 1)/2$     $g_y = g_y(T_S - 1)/2$   
  
  return ( $g_x, g_y$ ) ▷ Values for  $\frac{dL}{du}$   
end function
```

not exceed GPU capacity. While 32×32 and 16×16 both provide suitable metrics, we chose 16×16 as slightly more accurate.

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:  $\frac{dL}{dI}$  ▷ 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  
     $\frac{dL}{dc} = \alpha\mathcal{T}\frac{dL}{dI}$   
     $\frac{dL}{d\alpha} = (c - \hat{c})\mathcal{T}\frac{dL}{dI}$   
  
    Redistribute gradients  
     $\frac{dL}{du} = \text{BILINEAR}(\frac{dL}{dc}, \mathbf{u}, T_i^{\text{RGB}}, \frac{dL}{dT_i^{\text{RGB}}})$   
     $\frac{dL}{du} += \text{BILINEAR}(\frac{dL}{d\alpha}, \mathbf{u}, T_i^\alpha, \frac{dL}{dT_i^\alpha})$   
  
    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 9, we provide an additional visual comparison of BBSplat with state-of-the-art NVS methods. Our renderings demonstrate exceeding quality.

In Figure 8, we demonstrate additional renderings with ray-tracing effects and extracted meshes for the DTU dataset.

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 9, we report scene-by-scene results on the experiments provided in Table 1. We demonstrate state-of-the-art metrics quality for the Mip-NeRF-360 dataset on the indoor scenes while falling shortly behind for the outdoor scenes.

	PSNR \uparrow	L1 \downarrow	LPIPS \downarrow
Texture-GS	30.03	0.0135	0.1440
BBSplat(Ours)	30.81	0.0123	0.1055

Table 8. **Comparison with Texture-GS [57] on DTU.** While Texture-GS also applies textures for Gaussians, it does not perform well for NVS.

This is caused by a huge number of small monochromatic details (*e.g.* leafs) that are more efficient to represent with Gaussian primitives. For the Tanks&Temples outdoor scenes and DTU indoor scenes, we provide the best NVS for all cases.

7.3. Comparison to Texture-GS

In Table 8, we compare with Texture-GS [57] as it also proposes to employ textures, although in conjunction with a 3DGS model. 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.

7.4. Limitations and future work

One of the limitations of the proposed method is the NVS for the large outdoor scenes. While we provide high rendering quality for Tanks&Temples, there is still room for improvement in future works for the Mip-NeRF-360 dataset. 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. Another limitation is that the use of a higher amount of primitives for outdoor scenes leads to over-fitting. In future works, we are going to focus on resolving these limitations.

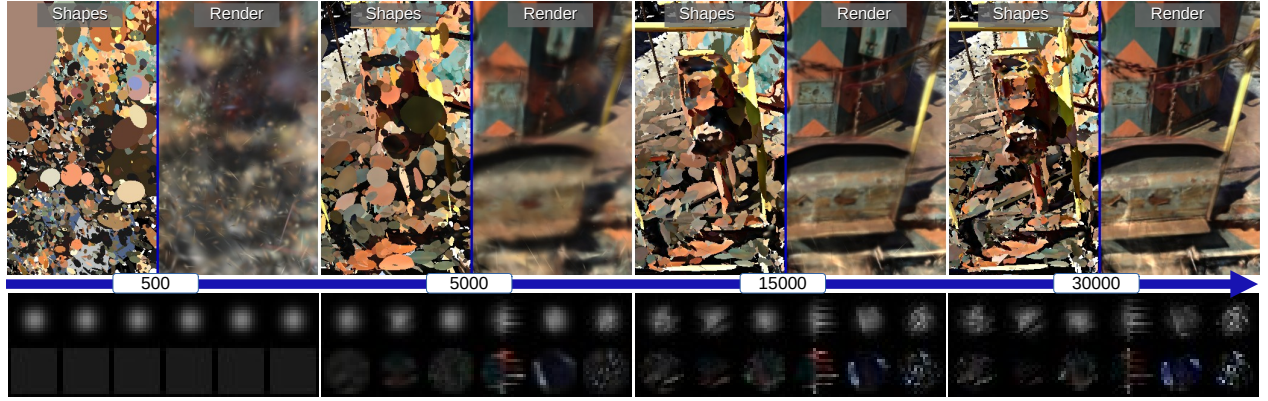


Figure 7. **BBSplat optimization process.** Top: We demonstrate billboard silhouettes and rendering results during the training process for 500, 5000, 15000, and 30000 iterations. To visualize silhouettes we disabled color textures and cut alpha-maps to the threshold. Bottom: We show examples of alpha-maps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

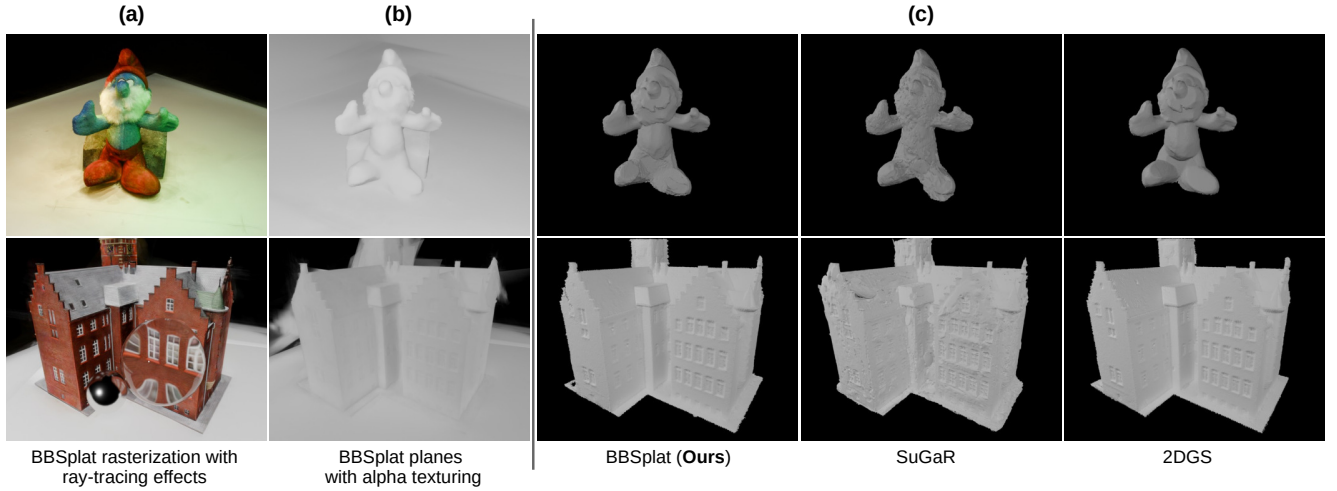


Figure 8. **Ray-tracing and mesh extraction.** a) BBSplat representation of the scene as explicit textured planes allows the implementation of ray-tracing effects during rasterization (*e.g.* relighting or mirror effects). BBSplat representation can be rasterized as splatting or with classic rasterization tools (*e.g.* Blender). b) Rasterization of planes with alpha texture provide accurate objects suitable for mesh extraction. c) Alternatively, our representation can be used to extract accurate 3D meshes, surpassing the accuracy of 3D Gaussians-based methods (*e.g.* SuGaR).

		3DGS [24]			3DGS-MCMC [25]			2DGS [†] [20]			BBSplat (Ours)		
		PSNR↑ / SSIM↑ / LPIPS↓			PSNR↑ / SSIM↑ / LPIPS↓			PSNR↑ / SSIM↑ / LPIPS↓			PSNR↑ / SSIM↑ / LPIPS↓		
Tanks&Temples	Train	19.81	/ 0.7229	/ 0.3347	21.21	/ 0.7522	/ 0.3120	19.35	/ 0.7149	/ 0.3417	21.38	/ 0.7702	/ 0.2604
	Truck	20.43	/ 0.7511	/ 0.3192	23.91	/ 0.8245	/ 0.2513	20.77	/ 0.7479	/ 0.3283	24.19	/ 0.8418	/ 0.1957
	Francis	24.16	/ 0.8456	/ 0.3368	27.82	/ 0.8813	/ 0.2964	22.61	/ 0.8356	/ 0.3485	28.56	/ 0.9079	/ 0.2334
	Horse	19.17	/ 0.7831	/ 0.2973	24.93	/ 0.8695	/ 0.2080	19.92	/ 0.7775	/ 0.3116	24.72	/ 0.8847	/ 0.1604
	Lighthouse	20.93	/ 0.8044	/ 0.2773	22.25	/ 0.8163	/ 0.2668	21.45	/ 0.8053	/ 0.2817	22.75	/ 0.8332	/ 0.2214
Mip-NeRF-360	Room	29.24	/ 0.8924	/ 0.2721	30.93	/ 0.9056	/ 0.2541	29.76	/ 0.8914	/ 0.2731	31.13	/ 0.9146	/ 0.2150
	Kitchen	24.56	/ 0.8712	/ 0.2129	29.16	/ 0.9003	/ 0.1787	25.93	/ 0.8745	/ 0.2095	30.35	/ 0.9116	/ 0.1430
	Bonsai	25.47	/ 0.8856	/ 0.2986	30.48	/ 0.9230	/ 0.2447	26.24	/ 0.8895	/ 0.2944	29.62	/ 0.9085	/ 0.2334
	Counter	26.36	/ 0.8716	/ 0.2628	28.28	/ 0.8910	/ 0.2350	26.65	/ 0.8701	/ 0.2649	28.30	/ 0.8941	/ 0.2061
	Bicycle	23.04	/ 0.5912	/ 0.4354	23.87	/ 0.6675	/ 0.3652	22.74	/ 0.5738	/ 0.4460	23.28	/ 0.6520	/ 0.3096
	Garden	23.98	/ 0.7093	/ 0.3400	25.06	/ 0.7512	/ 0.2974	24.26	/ 0.7053	/ 0.3419	25.75	/ 0.7873	/ 0.1907
DTU	Stump	24.08	/ 0.6387	/ 0.4035	25.82	/ 0.7201	/ 0.3311	24.05	/ 0.6399	/ 0.4047	24.00	/ 0.6547	/ 0.3376
	Scan24	22.09	/ 0.8450	/ 0.2590	27.19	/ 0.8745	/ 0.2319	21.83	/ 0.8423	/ 0.2681	27.47	/ 0.8850	/ 0.1677
	Scan37	20.01	/ 0.8097	/ 0.2942	24.18	/ 0.8449	/ 0.2673	20.83	/ 0.8187	/ 0.2881	24.93	/ 0.8670	/ 0.2127
	Scan40	19.21	/ 0.7676	/ 0.3535	22.46	/ 0.8132	/ 0.3196	20.47	/ 0.7717	/ 0.3493	25.56	/ 0.8709	/ 0.2081
	Scan55	25.25	/ 0.7934	/ 0.3831	27.88	/ 0.8221	/ 0.3775	25.67	/ 0.7959	/ 0.3857	27.74	/ 0.8511	/ 0.2746
	Scan63	23.91	/ 0.9098	/ 0.2156	31.08	/ 0.9411	/ 0.1792	30.11	/ 0.9228	/ 0.1963	30.32	/ 0.9422	/ 0.1652
	Scan65	27.76	/ 0.8476	/ 0.3648	29.41	/ 0.8585	/ 0.3605	29.32	/ 0.8513	/ 0.3688	29.67	/ 0.8754	/ 0.2803
	Scan69	26.20	/ 0.8485	/ 0.3545	26.97	/ 0.8592	/ 0.3483	26.34	/ 0.8465	/ 0.3576	26.71	/ 0.8660	/ 0.2845
	Scan83	28.36	/ 0.8629	/ 0.3888	29.62	/ 0.8679	/ 0.3818	28.69	/ 0.8590	/ 0.3921	28.97	/ 0.8685	/ 0.3604
	Scan97	22.90	/ 0.8263	/ 0.3762	27.67	/ 0.8588	/ 0.3478	24.45	/ 0.8360	/ 0.3673	27.39	/ 0.8579	/ 0.3268
	Scan105	24.58	/ 0.8429	/ 0.3781	30.10	/ 0.8687	/ 0.3579	23.92	/ 0.8362	/ 0.3797	29.35	/ 0.8765	/ 0.3154
	Scan106	30.46	/ 0.8780	/ 0.3632	33.38	/ 0.8983	/ 0.3455	30.39	/ 0.8723	/ 0.3629	32.93	/ 0.9020	/ 0.3133
	Scan110	30.71	/ 0.8779	/ 0.3773	31.55	/ 0.8898	/ 0.3666	31.12	/ 0.8806	/ 0.3771	31.42	/ 0.8798	/ 0.3324
	Scan114	28.58	/ 0.8642	/ 0.3540	29.82	/ 0.8807	/ 0.3462	28.65	/ 0.8636	/ 0.3547	29.72	/ 0.8870	/ 0.2976
	Scan118	31.77	/ 0.8856	/ 0.3561	34.06	/ 0.9010	/ 0.3482	32.49	/ 0.8886	/ 0.3572	34.41	/ 0.9090	/ 0.3077
	Scan122	29.83	/ 0.8640	/ 0.3603	34.19	/ 0.8949	/ 0.3459	33.52	/ 0.8956	/ 0.3521	34.41	/ 0.9069	/ 0.3092

Table 9. **Quantitative metrics for all scenes.** We report PSNR, SSIM, and LPIPS for each scene in all three datasets: Tanks&Temples [27], Mip-NeRF-360 [5], and DTU [23]. For comparison, we provide metrics for basic 3DGS [24] and 2DGS [20] methods, and for 3DGS-MCMC [25] as the best-scoring competitor.

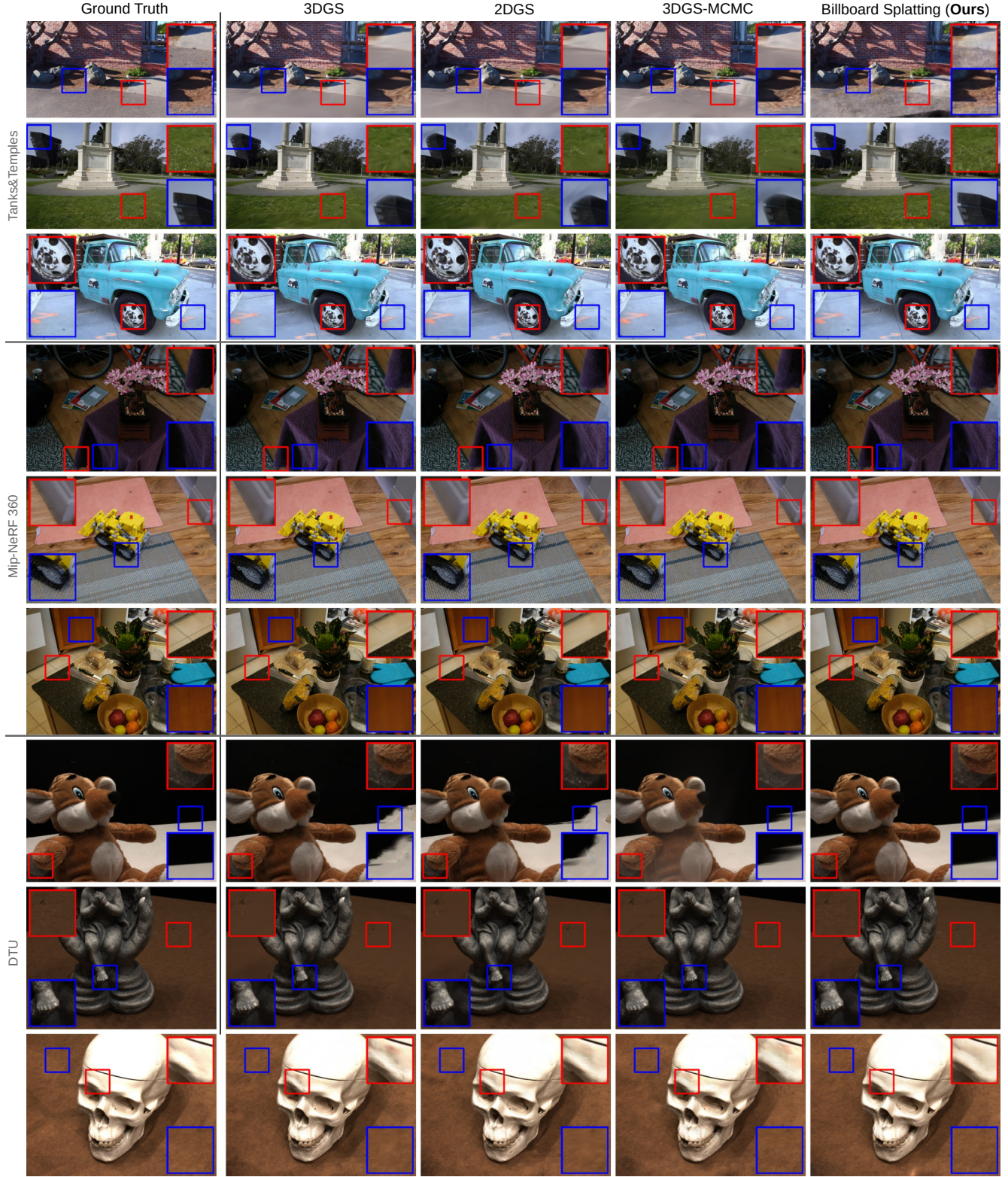


Figure 9. **Qualitative results.** We provide rendering results of three more scenes from each dataset: Tanks&Temples [Lighthouse, Francis, Truck], Mip-NeRF-360 [Bansai, Kitchen, Counter], and DTU [Scan105, Scan118, Scan65]. For competitors, we use the maximum number of Gaussians recommended by the method.