

1. Supplementary Material

1.1. Methodology

Depth-dependant scaling. In Triangle Splatting, the influence $I(p)$ depends only on the normalized ratio $\phi / \min \phi$. Since a projection is a uniform in-plane scaling, a single exponent σ suffices for all depths. Indeed, if we scale a triangle by $a > 0$, then each projected vertex v_i and pixel p transforms as $v_i \mapsto a v_i$ and $p \mapsto a p$, so the signed-distance $d_i(p)$ to each edge satisfies $d'_i(p') = a d_i(p)$, implying $\phi'(p') = \max(d'_i(p')) = a \phi(p)$ and $\min \phi' = a \min \phi$. Hence

$$\frac{\phi'(p')}{\min \phi'} = \frac{a \phi(p)}{a \min \phi} = \frac{\phi(p)}{\min \phi}, \quad (1)$$

and so $I'(p') = (\phi'(p') / \min \phi')^\sigma = (\phi(p) / \min \phi)^\sigma = I(p)$.

Tile assignment in Triangle Splatting Triangle Splatting is a tile-based renderer that assigns triangles to pixel tiles by computing their screen-space intersections. Unlike 3D Gaussian Splatting, where the shape has a soft spatial extent, Triangle Splatting is precisely bounded by the projection of its three vertices. A simple and efficient initial guess for determining tile coverage is to compute the minimum and maximum x and y coordinates of the projected vertices. While this method is computationally simple, it is conservative, resulting in unnecessary computations for pixels that are not influenced. As σ increases or opacity o decreases, the influence region may not reach the triangle’s vertices, causing the rasterizer to process pixels that have no contribution. To avoid this, we compute a more precise bounding box by determining the maximum distance d a pixel can be from the triangle edge while still contributing at least τ_{cutoff} influence. We define the influence function as:

$$\tau_{\text{cutoff}} = \left(\frac{d}{L(s)} \right)^\sigma \cdot o. \quad (2)$$

where $L(s)$ is the signed distance from the triangle’s edges to the incenter. Rearranging gives:

$$d = L(s) \cdot \left(\frac{\tau_{\text{cutoff}}}{o} \right)^{\frac{1}{\sigma}}. \quad (3)$$

We then subtract the distance d from the offset in each edge equation, effectively tightening the triangle’s boundary. The minimum and maximum of the updated edge intersections define a more accurate bounding box. As σ increases or o decreases, this bounding box shrinks accordingly, significantly reducing unnecessary rasterization.

Depth sorting. During rasterization, the triangles are currently sorted based on their center, which can lead to popping and blending artifacts during view rotation. Instead,

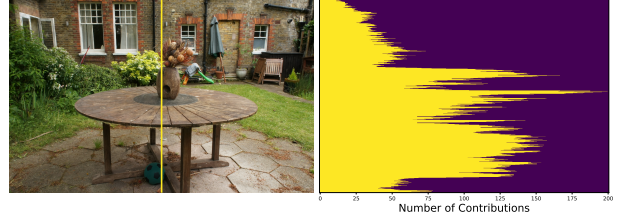


Figure 1. **Number of contributions per pixel.** In background regions where the initial point cloud is sparse, triangles reduce their σ to increase coverage across their interior. This leads to more solid shapes and, consequently fewer contributions per pixel.

future work can implement per-pixel sorting of the triangles, as proposed in [6], where Radl *et al.* introduce a hierarchical tile-based rasterization approach that performs local per-pixel sorting to ensure consistent visibility and eliminate popping artifacts.

Method	Outdoor	Indoor
feature_lr	0.0025	0.0025
opacity_lr	0.014	0.014
lr_convex_points_init	0.0018	0.0015
lr_sigma	0.0008	0.0008
lambda_normals	0.0001	0.00004
lambda_opacity	0.0055	0.0055
lambda_size	1e-8	5e-8
max_noise_factor	1.5	1.5
opacity_dead	0.014	0.014
split_size	24.0	24.0
importance_threshold	0.022	0.0256

Table 1. **Hyperparameters**

Densification. We prioritize sampling triangles with low σ values, corresponding to more *solid* triangles. Our window function ensures that each triangle’s influence is strictly limited to its projected area, preventing any influence beyond its geometric bounds. In regions with high triangle density, multiple shapes contribute to each pixel, allowing individual triangles to adopt larger σ values and produce softer, more diffuse effects. Conversely, in sparse regions where fewer triangles are present, each triangle must account for more of the pixel-wise reconstruction, leading it to adopt a smaller σ and thus contribute more across its surface. Figure 1 visualizes the number of contributions per pixel. In sparse background regions, triangles adopt lower σ values, resulting in solid shapes and fewer overlapping contributions per pixel. In contrast, densely sampled regions exhibit higher per-pixel contributions due to many overlapping triangles. As we prioritize adding new triangles to low-

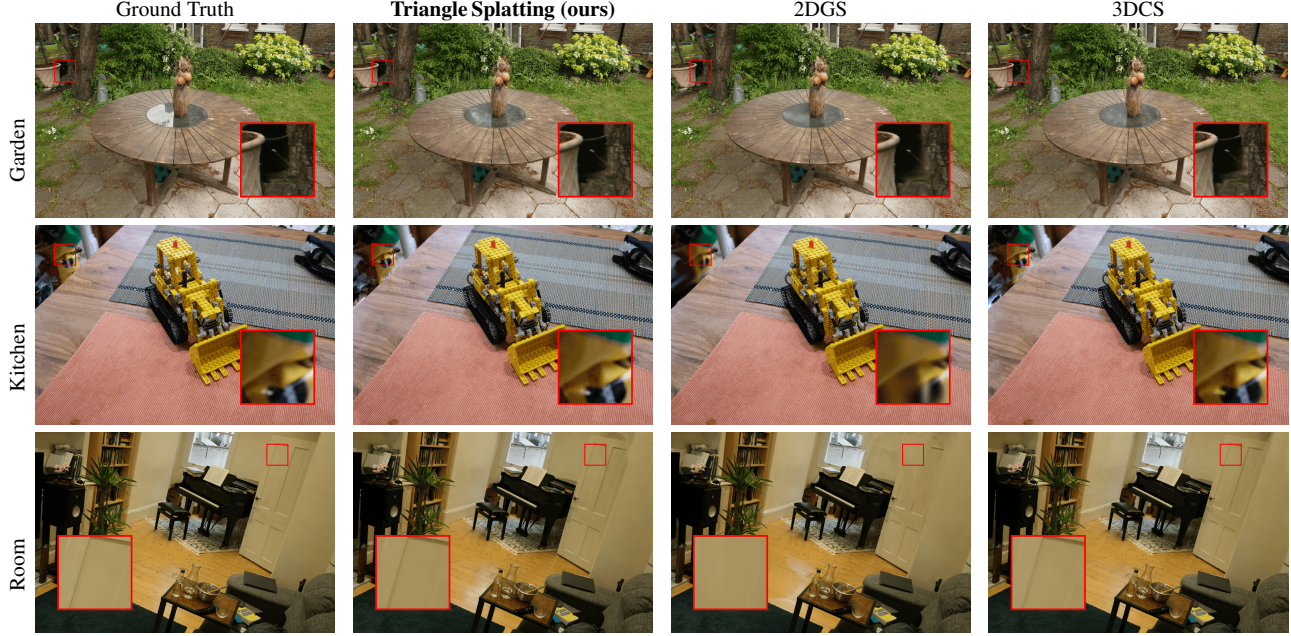


Figure 2. **Qualitative results.** We visually compare our method to 2DGS [3] and 3DCS [2]. Triangle Splatting captures finer details and produces more accurate renderings of real-world scenes, with less blurry results than 2DGS, and a higher visual quality than 3DCS [2].

density regions, we sample from a probability distribution defined over the inverse of σ .

1.2. Initialization & Hyperparameters.

Initialization. At initialization, each triangle is assigned a fixed opacity of 0.28, and a sigma value of 1.16. The scaling constant k of the convex hull is set to 2.2, which defines the spatial extent of the triangle in the 3D scene. These parameter values were determined empirically to ensure stable initialization and consistent rendering behavior across scenes.

Densification. We perform densification every 500 iterations, starting from iteration 500 until iteration 25,000. At each densification step, we increase the number of shapes by 30%.

All other hyperparameters are defined in Table 1.

1.3. More novel-view synthesis results

Tables 3 to 6 present a detailed analysis of the results on the Mip-NeRF360 and Tanks and Temples datasets, while Figure 2 presents additional qualitative results.

1.4. Geometry analysis of Triangle Splatting

Our triangle soup representation is already compatible with standard mesh-based renderers and can be rendered directly without modification. However, constructing a connected mesh from this representation still requires post-processing. A promising direction for future research is to design meshing strategies that take full advantage of the explicit triangle structure inherent to our approach. For completeness, we applied the meshing method used in 2D Gaussian Splatting and present the quantitative results on the DTU dataset [4] in Table 2 and some quantitative results in Figure 3. While this highlights compatibility with existing techniques, we advocate for future work to explore more principled and direct meshing methods tailored specifically to triangle-based representations. Figure 4 shows the normal map produced

Method	24	37	40	55	63	65	69	83	97	105	106	110	114	118	122	Mean
3DGS [5]	2.14	1.53	2.08	1.68	3.49	2.21	1.43	2.07	2.22	1.75	1.79	2.55	1.53	1.52	1.50	1.96
SuGaR [1]	1.47	1.33	1.13	0.61	2.25	1.71	1.15	1.63	1.62	1.07	0.79	2.45	0.98	0.88	0.79	1.33
2DGS [3]	0.48	0.91	0.39	0.39	1.01	0.83	0.81	1.36	1.27	0.76	0.70	1.40	0.40	0.76	0.52	0.80
Triangle Splatting	0.98	1.07	1.07	0.51	1.67	1.44	1.17	1.32	1.75	0.98	0.96	1.11	0.56	0.93	0.72	1.06

Table 2. **Chamfer distance on the DTU dataset[4].** We report the Chamfer distance on 15 scenes from DTU dataset.



Figure 3. **Mesh extraction from depth maps.** We extract meshes by applying TSDF fusion to the predicted depth maps, as followed in 2DGS [3].

by Triangle Splatting. The triangles are well aligned with the underlying geometry. For example, on the *Garden* table, all triangles share a consistent orientation and lie flat on the surface.

Method	LPIPS ↓		PSNR ↑		SSIM ↑	
	Truck	Train	Truck	Train	Truck	Train
3DGS [5]	0.148	0.218	25.18	21.09	0.879	0.802
2DGS [3]	0.173	0.251	25.12	21.14	0.874	0.789
3DCS [2]	0.125	0.187	25.65	22.23	0.882	0.820
Triangle Splatting	0.108	0.179	24.94	21.33	0.889	0.823

Table 3. LPIPS, PSNR, and SSIM scores for the Truck and Train scenes of the T&T dataset.

	Bicycle	Flowers	Garden	Stump	Treehill	Room	Counter	Kitchen	Bonsai
3DGS	0.205	0.336	0.103	0.210	0.317	0.220	0.204	0.129	0.205
2DGS	0.218	0.346	0.115	0.222	0.329	0.223	0.208	0.133	0.214
3DCS	0.216	0.322	0.113	0.227	0.317	0.193	0.182	0.117	0.182
Triangle Splatting	0.190	0.284	0.106	0.214	0.289	0.186	0.171	0.115	0.169

Table 4. LPIPS score for the MipNerf360 dataset.

	Bicycle	Flowers	Garden	Stump	Treehill	Room	Counter	Kitchen	Bonsai
3DGS	25.24	21.52	27.41	26.55	22.49	30.63	28.70	30.31	31.98
2DGS	24.87	21.15	26.95	26.47	22.27	31.06	28.55	30.50	31.52
3DCS	24.72	20.52	27.09	26.12	21.77	31.70	29.02	31.96	32.64
Triangle Splatting	24.90	20.85	27.20	26.29	21.94	31.05	28.90	31.32	31.95

Table 5. PSNR score for the MipNerf360 dataset.

	Bicycle	Flowers	Garden	Stump	Treehill	Room	Counter	Kitchen	Bonsai
3DGS	0.771	0.605	0.868	0.775	0.638	0.914	0.905	0.922	0.938
2DGS	0.752	0.588	0.852	0.765	0.627	0.912	0.900	0.919	0.933
3DCS	0.737	0.575	0.850	0.746	0.595	0.925	0.909	0.930	0.945
Triangle Splatting	0.765	0.614	0.863	0.759	0.611	0.926	0.911	0.929	0.947

Table 6. SSIM score for the MipNerf360 dataset.

1.5. Triangle splatting in a traditional renderer

Stochastic Triangle Splatting renderer. With the stochastic Triangle Splatting renderer, the optimized triangle soup (including opacity and σ values) can be loaded on a wide range of machines and hardware, running in real time even



Figure 4. **Normal map and rendered image.** The normal map reveals a smooth surface, with the triangle orientations consistently aligned to follow the local geometry.

Hardware	OS	TFLOPS	HD	Full HD	4k
MacBook M4	MacOS	8	500	370	160
RTX5000	Windows	11	570	380	290
RTX4090	Linux	48	2,400	1900	1050

Table 7. **FPS for different hardware and resolutions.** Evaluated on *Garden* ($\approx 2M$ triangles). OS stands for operating system.



Figure 5. **Byproduct of the triangle-based representation.** In a game engine, we render at 3,000 FPS at 1280×720 resolution on a RTX4090.

at high resolutions. We implemented a browser-based demo that runs on consumer laptops while still producing high visual quality. This prototype includes a simple temporal filtering scheme to reduce noise, **with the best quality obtained when the camera is static and a high number of samples per pixel are accumulated.** In practice, modern game engines already include advanced temporal anti-aliasing pipelines, which would further suppress noise and yield visual quality comparable to that achieved at the end of training.

The browser-based demo can be found under [the following link](#).

The code of the browser-based demo can be found under [the following link](#).

Mesh-based novel-view synthesis. To obtain higher visual quality, we naively prune low-opacity triangles. Specifically, during the final 5,000 training iterations, we remove all triangles with opacity below a threshold, retaining only

solid ones. Additionally, we introduce a loss term to encourage higher opacity and lower σ , ensuring that the final triangles are mostly solid and opaque. The conversion from an optimized triangle representation to a solid triangle representation is straightforward. The opacity and σ variables are set to 1 and 0, respectively, to obtain opaque and hard triangles. Finally, the vertex positions and the color of each triangle are stored in a format compatible with standard game engines (such as .off). No post-processing is required. Figure 5 presents qualitative results, while Table 7 reports rendering speeds across hardware and resolutions. Our method reaches 500 FPS at HD resolution on a consumer laptop and 2,400 FPS on an RTX 4090 within a game engine, highlighting both efficiency and practical usability. Compared to *stochastic rendering*, speed is higher since no sorting or additional computations are required, as all triangles are opaque and solid. Future work could explore training strategies tailored for game engine deployment, as our current setup targets novel-view synthesis rather than in-engine visual quality. This opens the door to integrating radiance fields directly into AR/VR and gaming pipelines.

References

- [1] Antoine Guédon and Vincent Lepetit. SuGaR: Surface-aligned Gaussian splatting for efficient 3D mesh reconstruction and high-quality mesh rendering. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 5354–5363, Seattle, WA, USA, 2024. Inst. Electr. Electron. Eng. (IEEE).
[2](#)
- [2] Jan Held, Renaud Vandeghen, Abdullah Hamdi, Adrien Delière, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, and Marc Van Droogenbroeck. 3D convex splatting: Radiance field rendering with 3D smooth convexes. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 21360–21369, Nashville, TN, USA, 2025. Inst. Electr. Electron. Eng. (IEEE). [2](#), [3](#)
- [3] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH Conf. Pap.*, pages 1–11, Denver, CO, USA, 2024. ACM. [2](#), [3](#)
- [4] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanaes. Large scale multi-view stereopsis evaluation. In *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 406–413, Columbus, OH, USA, 2014. Inst. Electr. Electron. Eng. (IEEE). [2](#)
- [5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):1–14, 2023.
[2](#), [3](#)
- [6] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted Gaussian splatting for view-consistent real-time rendering. *ACM Trans. Graph.*, 43(4):1–17, 2024.
[1](#)