MeshSplatting: Differentiable Rendering with Opaque Meshes

Supplementary Material

0.1. Methodology.

Mesh creation & refinement - Figure 1. We provide an additional visualization that illustrates both the visual and geometric improvements achieved across the different stages. Figure 1 shows how the geometry evolves by rendering the mesh without color in *Blender*. Fine-tuning the output of the restricted Delaunay triangulation not only enhances visual fidelity but also leads to more accurate geometry.

Restricted Delaunay triangulation.. As mentioned earlier, to construct the restricted Delaunay triangulation we first compute the standard Delaunay tetrahedralization of the input vertices, using the implementation in SciPy [9]. For each triangular face F in this tetrahedralization, we identify the two tetrahedra T_1^F and T_2^F that are adjacent to F; faces on the boundary of the tetrahedralization that have only a single incident tetrahedron are discarded. We then compute the circumcenters of T_1^F and T_2^F, which are vertices of the dual Voronoi diagram of the Delaunay tetrahedralization. The dual edge associated with F is obtained by connecting the circumcenters of T_1^F and T_2^F.

After computing the dual edges E associated with the Delaunay faces, we determine which of these edges intersect the triangles in our optimized triangle soup. To perform intersection tests efficiently, we build a bounding volume hierarchy over these triangles. During traversal, when testing an edge in E against a BVH node, we first check for overlap between their axis-aligned bounding boxes. If the node is internal, we continue the traversal to its children; if it is a leaf node, we apply a precise ray–triangle intersection test based on the Möller–Trumbore algorithm [8]. Whenever an intersection is detected, we mark the corresponding Delaunay face as part of the output mesh. We implement this detection pipeline in Taichi [4], whose automatic CPU/GPU parallelization yields a substantial speedup.

Hyerparameters - Table 1. The code will be released together with all hyperparameters used, ensuring that all results are fully reproducible. For completeness, we also list the most important hyperparameters here.

0.2. Addition experimental results

Detailed results - Table 2 & Table 3 & Table 4 & Figure 6 & Figure 7. We provide the per-scene results of MeshSplatting to facilitate detailed comparison and reproducibility. In addition, we provide further qualitative results demonstrating that MeshSplatting produces renderings with fewer artifacts and less noise compared to the current state of the art, MiLo. While MiLo generates high-quality meshes and therefore achieves a strong PSNR on the T&T dataset, its outputs exhibit more noise, leading to a higher LPIPS and a lower SSIM compared to MeshSplatting.

	Outdoor	Indoor
feature learning rate	0.0016	0.004
opacity learning rate	0.03	0.05
vertex pos. learning rate	0.0015	0.0015
densification start	500	500
densification end	10000	10000
densification interval	500	500
start pruning	4000	3000
pruning threshold	0.235	0.235
mesh creation	11000	11000
β_z	0.00025	0.00025
β_n	0.0001	0.0001
β_d	0.01	0.0
β_o	2e-06	0.0

Table 1. **Hyperparameters** List of the most important hyperparameters.

	Truck	Train
PSNR↑	22.32	18.72
LPIPS↓	0.237	0.337
SSIM↑	0.799	0.693

Table 2. Per scene LPIPS, PSNR, and SSIM scores for the Truck and Train scenes of the T&T dataset.

	Bicycle	Flowers	Garden	Stump	Treehill
PSNR↑	23.04	19.34	24.70	24.78	20.53
LPIPS↓	0.348	0.417	0.217	0.316	0.428
SSIM↑	0.641	0.480	0.762	0.678	0.540

Table 3. Per-scene PSNR, LPIPS, and SSIM scores for outdoor MipNeRF360 scenes.

	Room	Counter	Kitchen	Bonsai
PSNR↑	28.52	26.51	27.42	28.19
LPIPS↓	0.271	0.279	0.227	0.294
SSIM↑	0.873	0.846	0.858	0.876

Table 4. Per-scene PSNR, LPIPS, and SSIM scores for indoor MipNeRF360 scenes.

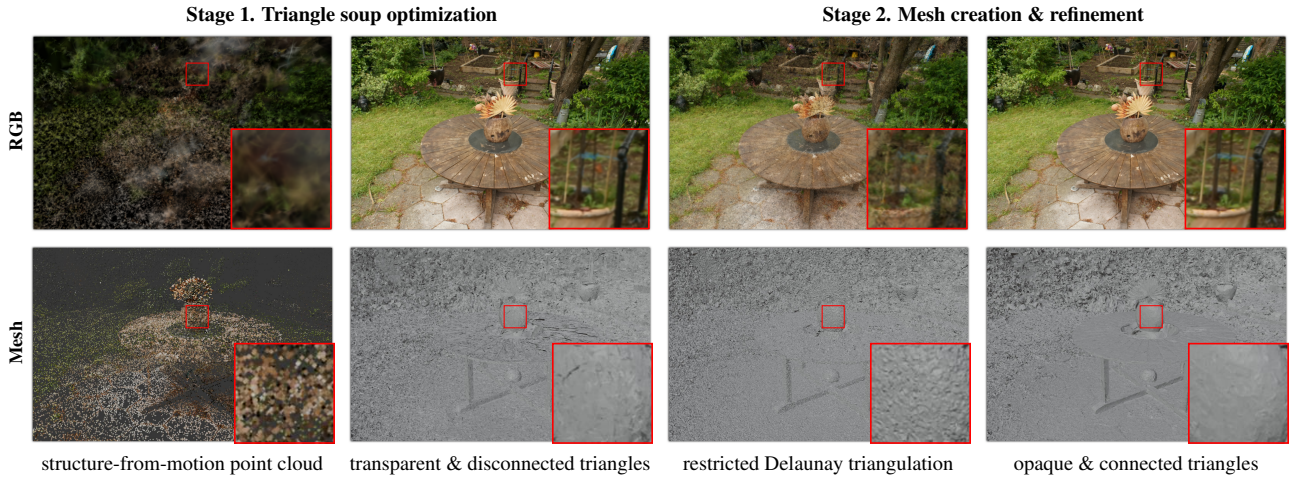


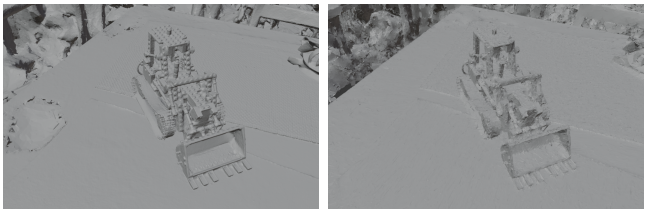
Figure 1. **From soups to triangle meshes.** The geometry improves significantly after refining the mesh.

Surface reconstruction - Table 6. MeshSplatting attains the best Chamfer distance on 5 of the 15 scenes. It achieves a significantly better mean score than Triangle Splatting and 3DGS. Compared to methods specifically tailored for mesh extraction, our results remain competitive and are on par with 2DGS. Even though our focus is mesh-based novel view synthesis, these results show that MeshSplatting also produces accurate surface meshes.

Fig. 2 shows the visual comparison of the reconstructed meshes with the current SOTA Milo. We acknowledge that our surface reconstruction metrics are slightly weaker than some surface reconstruction-focused methods, as our optimization is driven by rendering losses rather than explicit surface reconstruction objectives. Nevertheless, our geometric accuracy remains competitive with prior splatting-based approaches such as 2DGS, while significantly improving efficiency and memory. We agree that reducing surface noise and further improving geometric fidelity are important directions for future work, for example, by incorporating stronger geometric regularizers or topology-aware smoothing. Most downstream applications, such as physical interaction and ray tracing, remain fully compatible, since they rely on local surface geometry rather than strict manifoldness. As needed, standard mesh cleanup steps can be applied as lightweight post-processing.

	2M	3M	4M	5M
PSNR↑	22.14	+0.15	+0.36	+0.46
LPIPS↓	0.39	-0.02	-0.05	-0.06

Table 5. **Number of vertices *vs* visual quality.** MeshSplatting scales effectively with the number of vertices, showing consistent improvements in visual quality as the vertex count increases. All improvements are shown relative to 2M.



(a) Milo (b) MeshSplatting

Figure 2. **Surface quality.** While **2**× faster and using **50%** memory, MeshSplatting achieves competitive surface reconstruction. Please zoom in to see details.

Impact of triangle count on visual quality - Table 5. We evaluate the visual quality across different triangle counts on the outdoor scenes of MipNeRF360, as these scenes offer more room for improvement compared to indoor ones. Increasing the number of triangles consistently enhances visual quality, with clear gains in both PSNR and especially in LPIPS. The results show that MeshSplatting scales well with triangle count, with visual quality improving consistently as more triangles are used.

Object extraction - Figure 3. Given a 2D mask from SAMv2, we identify all visible triangles contributing to the masked pixels and aggregate them across all training views, yielding the complete set of triangles belonging to the selected object. The entire process takes up to two minutes, depending on the scene and the number of input views. Concretely, we first store the object masks for each training image, then iterate over all views to mark triangles as part of the object whenever they render at least one pixel within the mask. The extracted triangles can then be removed or exported as a standalone submesh without retraining.

0.3. Additional ablations - Table 7

Hard pruning step. An effective pruning strategy is crucial for achieving high visual quality with opaque primi-

Method	24	37	40	55	63	65	69	83	97	105	106	110	114	118	122	Mean
3DGS [7]	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
2DGS [5]	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
GOF [10]	0.50	0.82	0.37	0.37	1.12	0.74	0.73	1.18	1.29	0.68	0.77	0.90	0.42	0.66	0.49	0.74
RaDe-GS [11]	0.46	0.73	0.33	0.38	0.79	0.75	0.76	1.19	1.22	0.62	0.70	0.78	0.36	0.68	0.47	0.68
MiLo [2]	0.43	0.74	0.34	0.37	0.80	0.74	0.70	1.21	1.22	0.66	0.62	0.80	0.37	0.76	0.48	0.68
Triangle Splatting [3]	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
MeshSplatting	0.77	0.72	0.74	0.60	0.89	1.00	0.81	1.09	1.19	0.58	0.68	0.93	0.63	0.66	0.59	0.79

Table 6. **Chamfer distance on the DTU dataset** [6]. MeshSplatting achieves performance comparable to concurrent methods.

Figure 3. **Object extraction.** Additional visual examples demonstrating the object extraction capabilities of MeshSplatting. From left to right: generated RGB image of the object, estimated surface normals, and resulting mesh representation.

	PSNR ↑	LPIPS ↓	SSIM ↑
Baseline	24.78	0.31	0.728
w/o hard pruning	-0.67	+0.02	-0.021
w/o stage 2	-8.56	+0.25	-0.260
w/o supersampling	-0.80	+0.04	-0.040
w/o w pruning	-0.62	+0.05	0.045
w/o SH	-2.07	+0.06	-0.07
prune w/o conn.	-0.19	+0.01	-0.01
w/o $\mathcal{L}_d$	+0.05	-0.04	+0.006
w/o $\mathcal{L}_z$	+0.02	-0.01	+0.002
w/o $\mathcal{L}_n$	+0.10	-0.02	+0.004
w/o sigma decay	-7.96	+0.27	-0.329
cosine opacity schedule	-0.20	+0.01	-0.01
cosine σ schedule	-0.76	+0.03	-0.028

Table 7. **Detailed ablations (Mip-NeRF360).** We isolate the impact of each design choice by removing them individually.

tives. Without the hard pruning step at iteration $5k$, visual quality degrades noticeably. Unnecessary triangles remain in the scene and can no longer be removed.

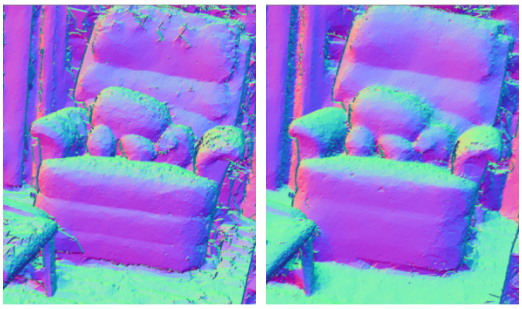
W/o stage 2. Running restricted Delaunay triangulation directly on the SfM point cloud produces a connected mesh that is not yet geometrically consistent, leaving many regions, particularly in the background, uncovered. Training with only connected triangles restricts the degrees of freedom of both vertices and triangles, making optimization considerably harder and leading to a drop in visual quality.

W/o supersampling. To reduce aliasing from opaque triangles, similar to Chen et al. [1], we render at $s\times$ the target resolution, and then downsample to the final resolution using area interpolation, which averages over input pixel regions to implement a box anti-aliasing filter. By disabling supersampling during the final training iterations and testing, visual quality decreases. This occurs because the representation uses fully opaque triangles, and metrics such as PSNR penalize small pixel-level shifts. When supersampling followed by average downsampling is applied, transitions between triangles become smoother, leading to more continuous transitions and higher visual quality. We applied anti-aliased rendering for both training and testing.

W/o *w* pruning. When pruning is based only on *opacity* rather than the maximum blending weight w, many unnecessary triangles remain in the scene. As these triangles become opaque, they introduce artifacts and reduce visual quality.

W/o SH. When training with only RGB colors, the visual quality drops significantly. The opaque and connected triangles struggle to accurately reproduce fine texture details. Real-world scenes exhibit complex spatial variations in color and illumination, which would require an impractically large number of triangles to model using RGB alone. Incorporating spherical harmonics enables each triangle to capture part of this variation, resulting in a noticeably improved appearance. This observation suggests that a more expressive appearance model could further enhance visual fidelity. Future work could explore learning per-triangle textures either jointly during training or as a post-processing refinement, as done in recent mesh-based methods. Such an approach could further narrow the visual quality gap between 3D Gaussian-based representations and fully opaque triangle-based ones.

Pruning w/o connectivity.. During the *first stage*, we exploit triangle connectivity by applying midpoint subdivision while ensuring that all four newly created triangles remain connected. This approach reduces the number of introduced vertices by half: subdividing each triangle would yield 12 vertices, whereas maintaining shared connectivity requires



(a) Without $\mathcal{L}_z$ (b) With $\mathcal{L}_z$

Figure 4. **Impact of the $\mathcal{L}_z$ regularization term.** With the regularization we obtain smoother surfaces.

(a) With noise (b) After training

Figure 5. **Noisy vertices.** Even with geometric noises on the vertices, MeshSplatting remains robust and delivers high visual quality.

only 6. By connecting triangles, we can allocate a larger portion of the memory budget to rendering more vertices and triangles, which leads to improved visual quality.

W/o $\mathcal{L}_z$ - Figure 4. Even with normal regularization, some triangles remain misoriented. By pushing the vertices closer to the surface, we improve both mesh smoothness and normal consistency. The vertex regularization term $\mathcal{L}_{\text{vertex}}$ reduces artifacts and promotes smoother surfaces, particularly in background regions with weaker supervision.

Cosine scheduler. Compared to a linear schedule for increasing $opacity$ and decreasing σ, we also experimented with a cosine scheduler. However, performance, particularly for σ, drops noticeably. The main reason is that the linear schedule maintains higher σ values for longer, allowing gradients to remain stable during the final iterations, whereas the cosine scheduler reaches low σ values earlier, causing gradients to vanish prematurely. Finally, we also analyze the impact of initialization. Instead of starting from soft triangles (i.e., $\sigma = 1.0$) that gradually converge toward solid ones, we evaluate the case where training begins directly with fully solid triangles. In this case, gradients can only flow through the opacity term, causing optimization to stagnate. As a result, both performance and visual quality degrade noticeably due to severe vanishing gradients.

Noisy input to restricted Delaunay triangulation.. We analyze the impact of noisy vertex positions as input to the restricted Delaunay triangulation. We inject scale-normalized Gaussian noise with a standard deviation equal to 1% of the scene extent into triangle-soup vertices prior to triangulation. Fig. 5 shows that even under strong geometric noise, the method exhibits only a minor performance degradation, with only +0.22 PSNR and -0.02 in LPIPS. In practice, before triangulation we already remove low-opacity triangles, and enforce depth and normal regularization's that removes floating artifacts and large geometric outliers during the triangle-soup stage.

0.4. Mesh connectivity.

The resulting mesh exhibits a *vertex–face ratio* of 0.48, which is close to the theoretical 0.5 expected for a closed manifold surface, indicating that the global mesh topology is already compact and near-manifold despite local non-manifold regions remaining. For reference, a completely isolated mesh with no vertices-sharing, would have a ratio of 1.5. We further analyze vertex connectivity by measuring the number of incident triangles (vertex valence) across the reconstructed mesh. The mean valence is 6.2 with a median of 5, which aligns with the expected value of 6 for regular triangular tessellations but with a broader distribution. Roughly 35% of vertices exhibit low valence (≤ 4), indicating remaining boundary regions and small disconnected fragments, while about 30% fall within the regular interior range (5–8). A small fraction ($< 10\%$) shows high valence (> 12), corresponding to locally dense or merged zones. This distribution shows that while the mesh is largely connected and compact, it remains not fully watertight.

0.5. Limitations

Limitations. MeshSplatting achieves high visual quality and accurate reconstruction in regions where the initial point cloud is dense. In contrast, background areas with sparse coverage still exhibit incomplete geometry and reduced fidelity. Moreover, when moving outside the orbit of training views, the visual quality degrades. While the softness and opacity of Gaussian-based approaches may still provide slightly plausible results in such cases, our use of opaque triangles makes the artifacts more pronounced. Furthermore, transparent objects such as glasses or bottles remain difficult to represent using only opaque triangles. Finally, MeshSplatting does not explicitly enforce watertightness; however, the resulting meshes can be directly used in many downstream applications, offering an effective trade-off between simplicity, usability, and high visual quality. Future work could incorporate additional regularization terms to constrain vertex motion and prevent self-intersections or overlaps.

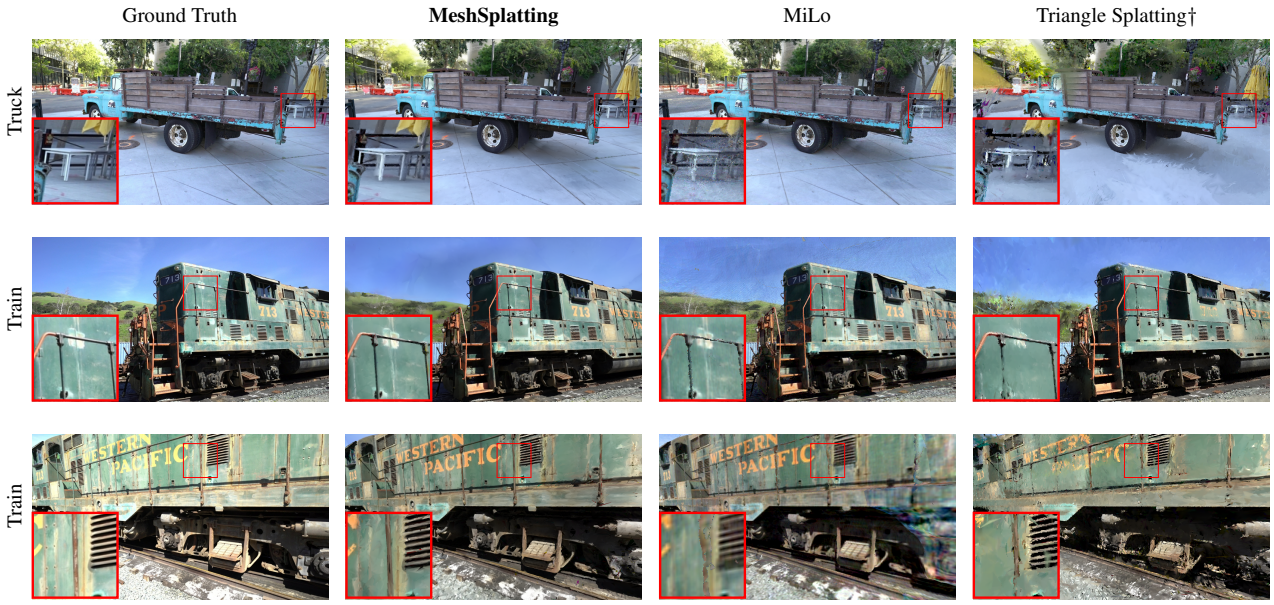


Figure 6. **More qualitative results on the *Tanks and Temples* dataset.** MeshSplatting reconstructs more details and finer structures compared to concurrent works. While MiLo achieves a higher PSNR, MeshSplatting produces fewer artifacts and less noisy renderings, which results in a significantly lower LPIPS and a higher SSIM.

Figure 7. **More qualitative results on the *MipNeRf-360*.** MeshSplatting is able to reconstruct fine details with only *opaque triangles.*

References

[1] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pages 16569–16578, Vancouver, Can., 2023. 3

[2] Antoine Guédon, Diego Gomez, Nissim Maruani, Bingchen Gong, George Drettakis, and Maks Ovsjanikov. MILo: Mesh-in-the-loop Gaussian splatting for detailed and efficient surface reconstruction. *ACM Trans. Graph.*, 44(6):1–15, 2025. 3

[3] Jan Held, Renaud Vandeghen, Adrien Deliège, Daniel Hamdi, Abdullah Rebain, Silvio Giancola, Anthony Cioppa, Andrea Vedaldi, Bernard Ghanem, Andrea Tagliasacchi, and Marc Van Droogenbroeck. Triangle splatting for real-time radiance field rendering. In *Int. Conf. 3D Vis. (3DV)*, pages 1–10, Vancouver, Can., 2026. 3

[4] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi. *ACM Trans. Graph.*, 38(6):1–16, 2019. 1

[5] 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. 3

[6] 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. 3

[7] 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. 3

[8] Tomas Möller and Ben Trumbore. Fast, minimum storage ray/triangle intersection. *ACM SIGGRAPH 2005 Courses*, page 7, 2005. 1

[9] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, Aditya Vijaykumar, Alessandro Pietro Bardelli, Alex Rothberg, Andreas Hilboll, Andreas Kloeckner, Anthony Scopatz, Antony Lee, Ariel Rokem, C. Nathan Woods, Chad Fulton, Charles Masson, Christian Häggström, Clark Fitzgerald, David A. Nicholson, David R. Hagen, Dmitrii V. Pasechnik, Emanuele Olivetti, Eric Martin, Eric Wieser, Fabrice Silva, Felix Lenders, Florian Wilhelm, G. Young, Gavin A. Price, Gert-Ludwig Ingold, Gregory E. Allen, Gregory R. Lee, Hervé Audren, Irvin Probst, Jörg P. Dietrich, Jacob Silterra, James T. Webber, Janko Slavič, Joel Nothman, Johannes Buchner, Johannes Kulick, Johannes L. Schönberger, José Vinícius de Miranda Cardoso, Joscha Reimer, Joseph Harrington, Juan Luis Cano Rodríguez, Juan Nunez-Iglesias, Justin Kuczynski, Kevin Tritz, Martin Thoma, Matthew Newville, Matthias Kümmerer, Maximilian Bolingbroke, Michael Tartre, Mikhail Pak, Nathaniel J. Smith, Nikolai Nowaczyk, Nikolay Shebanov, Oleksandr Pavlyk, Per A. Brodtkorb, Perry Lee, Robert T. McGibbon, Roman Feldbauer, Sam Lewis, Sam Tygier, Scott Sievert, Sebastiano Vigna, Stefan Peterson, Surhud More, Tadeusz Pudlik, Takuya Oshima, Thomas J. Pingel, Thomas P. Robitaille, Thomas Spura, Thouis R. Jones, Tim Cera, Tim Leslie, Tiziano Zito, Tom Krauss, Utkarsh Upadhyay, Yaroslav O. Halchenko, and Yoshiki Vázquez-Baeza. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat. Methods*, 17(3):261–272, 2020. 1

[10] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient adaptive surface reconstruction in unbounded scenes. *ACM Trans. Graph.*, 43(6):1–13, 2024. 3

[11] Baowen Zhang, Chuan Fang, Rakesh Shrestha, Yixun Liang, Xiaoxiao Long, and Ping Tan. RaDe-GS: Rasterizing depth in Gaussian splatting. *arXiv*, abs/2406.01467, 2024. 3