Supplementary Materials: Appendix of GeoSplatting

A. Overview

In the appendix, we provide a comprehensive explanation about the details of our work, including detailed implementations and limitations of our method, as well as supplementary results from both quantitative and qualitative experiments.

The appendix begins with a detailed explanation of the MGadapter in Appendix B, followed by an overview of the implementation details for our loss functions in Appendix D. In Appendix E, we provide details of the appearance refinement technique. Furthermore, to explore the limitations of our method, we include a discussion on the input mask required during training in Appendix F and the impact of isosurface resolution in Appendix G. Finally, we demonstrate how name can be enhanced with an initial mesh in Appendix H and incorporated with path tracing in Appendix I.

B. Explanation of MGadapter

We first describe the details of our MGadapter. As discussed in Sec. 3.1, the MGadapter takes a triangle mesh as input and generates a set of Gaussian points corresponding to the shape of the underlying mesh. The core idea of MGadapter is to maintain shape consistency between the Gaussian points and the mesh guidance.

B.1. Overview

The straightforward implementation of our MGadapter involves sampling several Gaussian points on the mesh surface, which are then optimized in terms of scale and rotation to minimize the depth map difference between the Gaussian points and the target mesh. However, this approach introduces an additional optimization step that must be re-executed each time the mesh is modified, resulting in reduced optimization efficiency and an unstable training process.

Instead of maintaining shape consistency in real-time, we propose utilizing a predefined heuristic function $\mathcal{T}$ to achieve an approximate alignment. As described in Eq. 1 (Sec. 3.1), the MGadapter $\mathcal{T}$ takes arbitrary triangle meshes as input and outputs Gaussian point attributes, including position $\boldsymbol{\mu}$, scale $\mathbf{S}$, rotation $\mathbf{R}$, and normal $\mathbf{n}$. This process acts as a generalized adapter between the input meshes and the corresponding Gaussian points. We provide a detailed implementation in Sec. B.2, followed by an analysis of shape consistency between the underlying mesh and our generated Gaussian points in Sec. B.3. Then, we explain the OOM issue at the beginning of the training stage and propose the vertex-sample stage to address it in Sec. B.4.

B.2. Implementation of Eq. 1

Specifically, given the triangle mesh, each triangle face F_i comprises three vertices $\mathbf{P}_i = (\mathbf{p}_{i1}, \mathbf{p}_{i2}, \mathbf{p}_{i3})$ with their vertex normals $\mathbf{N}_i = (\mathbf{n}_{i1}, \mathbf{n}_{i2}, \mathbf{n}_{i3})$. We symmetrically sample 6 points on F_i with barycentric

coordinates:

$$\begin{aligned} \mathbf{b}_1 &= (u, u, 1-2u) \ , \quad \mathbf{b}_4 = (v, v, 1-2v) \ , \\ \mathbf{b}_2 &= (u, 1-2u, u) \ , \quad \mathbf{b}_5 = (v, 1-2v, v) \ , \\ \mathbf{b}_3 &= (1-2u, u, u) \ , \quad \mathbf{b}_6 = (1-2v, v, v) \ . \end{aligned} \tag{7}$$

And we can obtain 6 midpoints m_{jk}:

$$\begin{aligned} \mathbf{m}_{12} &= \frac{\mathbf{b}_1 + \mathbf{b}_2}{2} = \left(u, \frac{1-u}{2}, \frac{1-u}{2}\right) \ , \quad \mathbf{m}_{45} = \frac{\mathbf{b}_4 + \mathbf{b}_5}{2} = \left(v, \frac{1-v}{2}, \frac{1-v}{2}\right) \ , \\ \mathbf{m}_{23} &= \frac{\mathbf{b}_2 + \mathbf{b}_3}{2} = \left(\frac{1-u}{2}, \frac{1-u}{2}, u\right) \ , \quad \mathbf{m}_{56} = \frac{\mathbf{b}_5 + \mathbf{b}_6}{2} = \left(\frac{1-v}{2}, \frac{1-v}{2}, v\right) \ , \\ \mathbf{m}_{31} &= \frac{\mathbf{b}_3 + \mathbf{b}_1}{2} = \left(\frac{1-u}{2}, u, \frac{1-u}{2}\right) \ , \quad \mathbf{m}_{64} = \frac{\mathbf{b}_6 + \mathbf{b}_4}{2} = \left(\frac{1-v}{2}, v, \frac{1-v}{2}\right) \ . \end{aligned} \tag{8}$$

Given an attribute $\mathbf{A}_i = (\mathbf{a}_{i1}, \mathbf{a}_{i2}, \mathbf{a}_{i3})$ defined at the triangle vertices and a barycentric coordinate (q_1, q_2, q_3), we represent the barycentric interpolation as:

$$(q_1, q_2, q_3) \odot \mathbf{A}_i = q_1 \mathbf{a}_{i1} + q_2 \mathbf{a}_{i2} + q_3 \mathbf{a}_{i3} \ . \tag{9}$$

Then, for each midpoint m_{jk}, we sample a Gaussian point as:

$$\begin{aligned} \boldsymbol{\mu} &= \mathbf{m}_{jk} \odot \mathbf{P}_i \ , & \mathbf{n} &= \mathbf{m}_{jk} \odot \mathbf{N}_i \ , \\ \mathbf{S}_x &= \alpha_{jk} \| \mathbf{b}_k \odot \mathbf{P}_i - \mathbf{m}_{jk} \odot \mathbf{P}_i \|_2 \ , & \mathbf{R}_x &= \frac{\mathbf{b}_k \odot \mathbf{P}_i - \mathbf{m}_{jk} \odot \mathbf{P}_i}{\| \mathbf{b}_k \odot \mathbf{P}_i - \mathbf{m}_{jk} \odot \mathbf{P}_i \|_2} \ , \\ \mathbf{S}_y &= \frac{\text{Area}(F_i)}{\beta_{jk} \| \mathbf{b}_k \odot \mathbf{P}_i - \mathbf{m}_{jk} \odot \mathbf{P}_i \|_2} \ , & \mathbf{R}_y &= \mathbf{n} \times \mathbf{R}_x \ , \\ \mathbf{S}_z &= \delta_{jk} \ , & \mathbf{R}_z &= \mathbf{n} \ . \end{aligned} \tag{10}$$

Here, Eq. 10 provide the formulation of our heuristic function $\mathcal{T}$, with $u, v, \alpha_{jk}, \beta_{jk}, \delta_{jk}$ as hyperparameters. To achieve the generalized geometric alignment, we practically set these parameters as follows:

$$\begin{aligned} u &= 0.07 \ , \\ v &= 0.22 \ , \\ \alpha_{12} = \alpha_{23} = \alpha_{31} &= 0.80 \ , \\ \alpha_{45} = \alpha_{56} = \alpha_{64} &= 2.08 \ , \\ \beta_{12} = \beta_{23} = \beta_{31} &= 15.0 \ , \\ \beta_{45} = \beta_{56} = \beta_{64} &= 13.0 \ , \\ \delta_{12} = \delta_{23} = \delta_{31} = \delta_{45} = \delta_{56} = \delta_{64} &= 4.5 \times 10^{-5} \ . \end{aligned} \tag{11}$$

B.3. Analysis of Shape Consistency

Metrics We begin by providing a formal definition of shape consistency, as 3DGS points lack inherent geometry boundaries. In the context of inverse rendering, geometry quality is critical for accurate light transport modeling. Therefore, we measure shape consistency based on light transport accuracy, specifically in terms of errors in light reflection directions and light transfer distances. For computational convenience, we assess these errors using a ray-tracing approach. Specifically, given a reference mesh model and a set of viewpoints, we compute the intersections between per-pixel camera rays and the underlying mesh surface in screen space. By measuring the Mean Angular Error (MAE) of the reflected ray directions and the distance between intersection points and the camera position, we evaluate shape consistency, as shown in Table 5.

Metrics	Air balloons	Chair	Hotdog	Jugs
Reflecting Directions (MAE ↓)	1.23	0.81	0.74	0.89
Transfer Distance (L1 ↓)	0.016	0.010	0.012	0.013

Table 5. We measure shape consistency on the Synthetic4Relight dataset, including reflection directions (in degrees) and transfer distances (relative to the size of the scene's bounding box).

Discussion Numerous works have attempted to ground 3DGS points to the triangle mesh surface. While many focus on making Gaussian points deformable [6, 16] or enhancing rendering quality [3, 9], only a few have addressed the enhancement of geometry quality in 3DGS through mesh-Gaussian binding.

MeshSplats [15] converts Gaussian points into triangle slices, enabling ray tracing techniques for advanced lighting effects such as shadows and reflections. However, this method operates on pretrained Gaussian points, aligning them with the mesh in a post-hoc manner. This approach is not compatible with inverse rendering methods, which require a fully differentiable pipeline to propagate gradients from photometric loss functions to the underlying geometry. Another method [8] proposes a simple technique for differentiable mesh-Gaussian alignment, enabling end-to-end training. However, this technique only considers mesh-Gaussian alignment along the normal direction, leaving tangent-space alignment uncontrolled—similar to the ablated Gaussian sampling approach discussed in Sec. 4.4. As demonstrated in Table 3 of Sec. 4.4, simply applying this method to inverse rendering tasks results in shape disparity between the 3DGS and the mesh, leading to inaccurate light transport modeling and degraded decomposition results.

In contrast, we have carefully designed our MGadapter to ensure shape consistency between the Gaussian points and the mesh guidance. The inherently shape-consistent nature of MGadapter allows for end-to-end optimization of geometry guidance during training, enabling precise normal estimation and accurate light transport modeling for superior inverse rendering performance.

B.4. Vertex-Sampling Stage

As outlined in Sec. B.2, we typically sample six Gaussian points from each triangle surface. However, at the start of the training stage, the isovalues of FlexiCubes are randomly initialized, leading to an excessive number of triangle slices, as shown in Fig. 11. Directly sampling six Gaussian points per face in this context can incur substantial memory costs and reduce training efficiency.

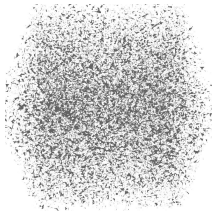


Figure 11. Initial mesh slices of FlexiCubes.

To address this issue, we implement an vertex-sampling stage for MGadapter at the beginning of training (covering the first 5%). During this stage, MGadapter outputs a significantly reduced number of Gaussian points by performing vertex sampling. Specifically, given a mesh vertex $\mathbf{v}$ with its normal $\mathbf{n_v}$, MGadapter samples a single Gaussian point with $\boldsymbol{\mu} = \mathbf{v}, \mathbf{n} = \mathbf{n_v}$, and:

$$\begin{aligned} \mathbf{S}_x = \mathbf{S}_y = \sqrt{\frac{k}{3} \sum_{F_i \in \text{Star}(\mathbf{v})} \text{Area}(F_i)} \ , \ \mathbf{S}_z = \delta_{jk} \ , \\ \mathbf{R}_x = (0,0,1) \times \mathbf{n_v} \ , \ \mathbf{R}_y = \mathbf{n_v} \times \mathbf{R}_x \ , \ \mathbf{R}_z = \mathbf{n_v} \ . \end{aligned} \tag{12}$$

Once the vertex-sampling stage concludes, MGadapter switches to face-sampling, as described in Sec. B.2.

C. Explanation of PBR Attribute Modeling

Since Gaussian points are generated in real time from the underlying mesh, directly modeling these attributes as learnable parameters is impractical due to the varying number of Gaussian points during training. Instead, as described in Eq. 4 (Sec. 3.2), we employ multi-resolution hash grids $\mathcal{E}_{\mathrm{d}}, \mathcal{E}_{\mathrm{s}}$ to construct attribute fields. Specifically, the hash grids incorporate the multi-resolution hash encoders introduced in [11], followed by small MLP headers. We implement them using tiny-cuda-nn[10]. Detailed parameters can be found in Table 6.

Module	Parameter	Value
$\mathcal{E}_{\mathrm{d}}$/$\mathcal{E}_{\mathrm{s}}$ HashEnc	Number of levels	16
	Max.entries per level (hash table size)	2^{19}
	Number of feature dimensions per entry	2
	Coarsest resolution	32
	Finest resolution	4096
$\mathcal{E}_{\mathrm{d}}$ MLP	MLP layers	$32 \times 32 \times 32 \times 6$
	Initialization	Kaiming-uniform
	Final activation	Sigmoid
$\mathcal{E}_{\mathrm{s}}$ MLP	MLP layers	$32 \times 32 \times 3$
	Initialization	Kaiming-uniform
	Final activation	None

Table 6. Parameters of Spatial MLP

D. Details of Loss Functions

D.1. Photometric Term

During the training stage, for the i-th view , GeoSplatting differentiably renders a RGB image $I_{\mathrm{pred}}^{(i)} \in \mathbb{R}^{H\times W\times 3}$ and takes the alpha channel as the mask $M_{\mathrm{pred}}^{(i)} \in \mathbb{R}^{H\times W\times 1}$. Given the ground truth $I_{\mathrm{gt}}^{(i)}$ and $M_{\mathrm{gt}}^{(i)}$ for view i, the photometric loss is computed as:

$$\begin{aligned}\mathcal{L}_{\mathrm{photo}} =& \mathcal{L}_1 + \lambda_{\mathrm{ssim}}\mathcal{L}_{\mathrm{SSIM}} + \lambda_{\mathrm{mask}}\mathcal{L}_{\mathrm{mask}} \\ =& \|I_{\mathrm{gt}}^{(i)} - I_{\mathrm{pred}}^{(i)}\|_1 + \lambda_{\mathrm{ssim}}\mathrm{SSIM}(I_{\mathrm{gt}}^{(i)}, I_{\mathrm{pred}}^{(i)}) + \lambda_{\mathrm{mask}}\|M_{\mathrm{gt}}^{(i)} - M_{\mathrm{pred}}^{(i)}\|_2^2 \ .\end{aligned} \tag{13}$$

Here, $\lambda_{\mathrm{ssim}} = 0.2$ and $\lambda_{\mathrm{mask}} = 5.0$ for all the cases.

D.2. Entropy Regularization Term

Following DMTet and FlexiCubes [13, 14], we add an entropy loss to constrain the shape. Specifically, we employ FlexiCubes as the underlying geometric representation, which defines a scalar function $\boldsymbol{\zeta} : \mathbb{R}^3 \to \mathbb{R}$ on the underlying cube grids $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and then extracts isosurfaces via the differential Dual Marching Cubes introduced by [14]. Given an edge (v_i, v_j) from edge set $\mathcal{E}$, the SDF values defined on the endpoints v_i, v_j are respectively $\boldsymbol{\zeta}(v_i)$ and $\boldsymbol{\zeta}(v_j)$.

Then, we can compute the regularization term as:

$$\mathcal{L}_{\text{sdf}} = \sum_{(v_i,v_j)\in\mathcal{E},\text{sgn}(\boldsymbol{\zeta}(v_i))\neq\text{sgn}(\boldsymbol{\zeta}(v_j))} \mathcal{H}(\boldsymbol{\zeta}(v_i), \text{sgn}(\boldsymbol{\zeta}(v_j))) + \mathcal{H}(\boldsymbol{\zeta}(v_j), \text{sgn}(\boldsymbol{\zeta}(v_i))) \tag{14}$$

Here, $\mathcal{H}$ denotes the binary cross entropy. By encouraging the same sign of $\boldsymbol{\zeta}$, such a regularization term penalize internal geometry and floaters.

D.3. Smoothness Regularization Term

Following prior works [5, 7, 12], we apply smoothness regularization on albedo, roughness, and metallic to prevent dramatic high-frequency variations. Given the positions $\boldsymbol{\mu}$ of Gaussian points, the albedo, roughness and metallic attributes are generated by the hash grids:

$$\mathbf{a} = \mathcal{E}_{\text{d}}(\boldsymbol{\mu}) \ \ , (\rho, m) = \mathcal{E}_{\text{s}}(\boldsymbol{\mu}) \ \ . \tag{15}$$

While applying a small perturbation $\Delta\boldsymbol{\mu} \sim \mathcal{N}(0, \sigma^2)$ on $\boldsymbol{\mu}$ can yield a different set of attributes:

$$\mathbf{a}' = \mathcal{E}_{\text{d}}(\boldsymbol{\mu} + \Delta\boldsymbol{\mu}) \ \ , (\rho', m') = \mathcal{E}_{\text{s}}(\boldsymbol{\mu} + \Delta\boldsymbol{\mu}) \ \ . \tag{16}$$

The smoothness is then computed as:

$$\mathcal{L}_{smooth} = \|\mathbf{a} - \mathbf{a}'\|_1 + \|\rho - \rho'\|_1 + \|m - m'\|_1 \tag{17}$$

Here, $\sigma = 0.01$.

D.4. Light Regularization Term

Following NVdiffrecmc [7], we add a light regularization that is based on monochrome image loss between the demodulated lighting terms and the reference image. Given the demodulated diffuse lighting L_{d}, the specular lighting L_{s}, and the reference image I_{gt}, the regularization is computed as follows:

$$\mathcal{L}_{\text{light}} = \|Y(L_{\text{d}} + L_{\text{s}}) - V(I_{\text{gt}})\| \tag{18}$$

Here, $Y(\mathbf{x}) = (\mathbf{x}_r + \mathbf{x}_g + \mathbf{x}_b)/3$ is the luminance operator, and $V(\mathbf{x}) = \max(\mathbf{x}_r, \mathbf{x}_g, \mathbf{x}_b)$ is the HSV value component. As discussed in the original paper [7], this regularization is based on the assumption that the demodulated lighting is mostly monochrome, *i.e.*, $Y(\mathbf{x}) \sim V(\mathbf{x})$, and it is proven to be effective for shadow disentanglement.

D.5. Final Loss

The final loss $\mathcal{L}$ is computed as:

$$\mathcal{L} = \mathcal{L}_{\text{photo}} + \lambda_{\text{sdf}}\mathcal{L}_{\text{sdf}} + \lambda_{\text{smooth}}\mathcal{L}_{\text{smooth}} + \lambda_{\text{light}}\mathcal{L}_{\text{light}} \tag{19}$$

Here, λ_{sdf} is initially set to 0.2 at the start of the training stage and is linearly decreased to 0.01 by the midpoint of the training, with $\lambda_{\text{smooth}} = 0.03$ and $\lambda_{\text{light}} = 0.15$.

E. Explanation of Appearance Refinement

In this section, we provide a detailed explanation of the implementation of our appearance refinement technique. Specifically, we enable GeoSplatting to transition to deferred shading and optimize 3DGS attributes to adjust their displacement, including positions, scales, rotations, and opacities.

Deferred Shading As described in Sec. 3.2, we extract per-Gaussian PBR attributes and conduct Monte Carlo sampling to obtain Gaussian-wise colors. Then, we utilize alpha-blending to rasterize Gaussian-wise colors into screen-space pixels, which constitute the forward shading:

$$\mathrm{c}_{\mathrm{GS}} = \mathrm{PBR}(\boldsymbol{\mu}, \mathbf{n}, \mathbf{a}, \rho, m) \ , \qquad I_{\mathrm{RGB}} = \mathrm{Rasterize}(\mathrm{c}_{\mathrm{GS}}) \ . \tag{20}$$

During the appearance Refinement, we conduct deferred shading instead. Specifically, we render per-Gaussian attributes into screen-space attribute map.

$$\begin{aligned} I_{\mu} &= \mathrm{Rasterize}(\boldsymbol{\mu}) \ , \quad I_{\mathrm{n}} = \mathrm{Rasterize}(\mathbf{n}) \ , \\ I_{\mathrm{a}} &= \mathrm{Rasterize}(\mathbf{a}) \ , \quad I_{\rho} = \mathrm{Rasterize}(\rho) \ , \quad I_{\mathrm{m}} = \mathrm{Rasterize}(m) \ . \end{aligned} \tag{21}$$

Then, PBR is conducted in screen-space:

$$I_{\mathrm{RGB}} = \mathrm{PBR}(I_{\mu}, I_{\mathrm{n}}, I_{\mathrm{a}}, I_{\rho}, I_{\mathrm{m}}) \ . \tag{22}$$

Optimization Details Since the transition from forward shading to deferred shading is not seamless and may introduce artifacts, we perform a slight optimization using a learning rate of 0.001 for 100 steps (approximately 1 minute). This optimization adjusts the displacement of the 3DGS. During this process, we unlock the geometry constraints, allowing the 3DGS to modify its positions, scales, rotations, normals, and opacities. As shown in Table 7, this adjustment does not significantly alter these attributes, ensuring that the Gaussian-mesh consistency remains well-preserved.

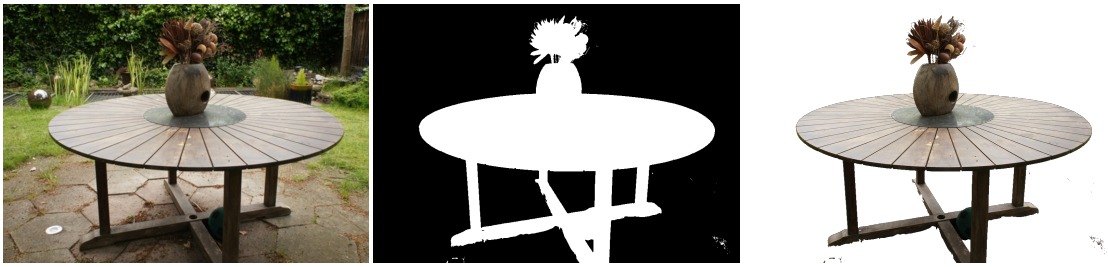
Dataset	positions $\Delta\boldsymbol{\mu}$	scales $\Delta\mathbf{S}$	rotations $\Delta\mathbf{R}$	opacities Δo	normals $\Delta\mathbf{n}$
TensoIR Synthetic Dataset	0.0014	0.0003	0.0148	0.0131	0.0146
Shiny Blender Dataset	0.0011	0.0002	0.0118	0.0168	0.0087

Table 7. We measure the average L1 difference between the unoptimized and optimized attributes to validate that this technique does not significantly degrade geometry quality.

F. Discussion on Mask Requirements

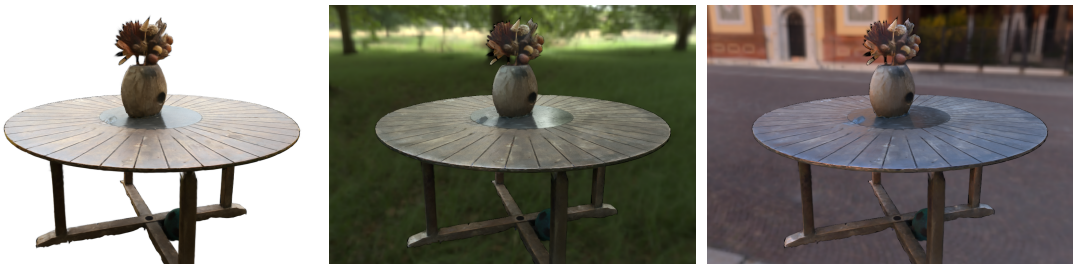
Benefiting from the explicit mesh normals and the opaque mesh surface, our GeoSplatting achieves improved light transport modeling compared to previous 3DGS-based inverse rendering approaches. However, as mentioned in Sec. 3.4, optimizing explicit surfaces is quite challenging, and existing isosurface-based approaches [7, 12] typically require an object mask loss for background removal. Similar to these methods, our GeoSplatting also relies on input masks. In this section, we discuss how to mitigate this limitation and adapt our approach for real-world applications in the absence of ground truth masks.

As analyzed in prior work [12], isosurface techniques are not highly sensitive to mask quality, and a coarse input mask is sufficient. Therefore, we use vanilla 3DGS to develop a simple solution that effectively approximates object masks for wild captures. Specifically, given a set of multi-view captures, we first train a standard 3DGS for only 5000 steps (2-4 minutes) to obtain Gaussian points. Then, we simply choose a bounding box and clip the Gaussian points outside the bounding box. As illustrated in Fig. 12, the rendering results of the remaining Gaussian points naturally form a coarse mask, which can be used for background removal. Based on this mask estimation trick, our GeoSplatting can be applied to object decomposition for wild captures. As shown in Fig. 13, we demonstrate the decomposition and relighting results for the *Garden* scene.



(a) Original Capture (b) Binary Mask (c) Masked Capture

Figure 12. Coarse mask estimated on *Garden* from the Mip-NeRF 360 Dataset [1].



(a) NVS (b) Relit 1 (c) Relit 2

(d) Normal (e) Albedo (f) Roughness

Figure 13. Decomposition and relighting results of *Garden* from the Mip-NeRF 360 Dataset [1].

G. Analysis of Resolution

As discussed in Sec. 5, one of the most significant limitations of our GeoSplatting is the resolution constraint. Due to the limited grid resolution of the isosurface technique (FlexiCubes), GeoSplatting struggles with thin structures and surfaces featuring complicated geometry. This issue can be mitigated by simply increasing the FlexiCubes resolution. We present a comparison in Fig. 14 to demonstrate the impact of different resolutions on the recovery of thin geometry ($R = 96$ and $R = 128$, respectively).

However, due to the space and time complexity of $O(R^3)$, the optimization process with $R = 96$ can be easily conducted on a single NVIDIA RTX 4090 (24GB CUDA memory) within 15-20 minutes. In contrast, the $R = 128$ setting typically takes more than 1 hour to train and requires more than 60GB of CUDA memory. Therefore, in Sec. 4, when presenting experimental comparisons, we use $R = 72$ for simple shapes and $R = 96$ for others. Furthermore, this resolution limitation also restricts our method from adapting to scene-level decomposition. A promising direction for future work is to explore how adaptive resolution techniques could be applied to accommodate detailed geometry, enabling the extension of our approach to scene-level tasks.

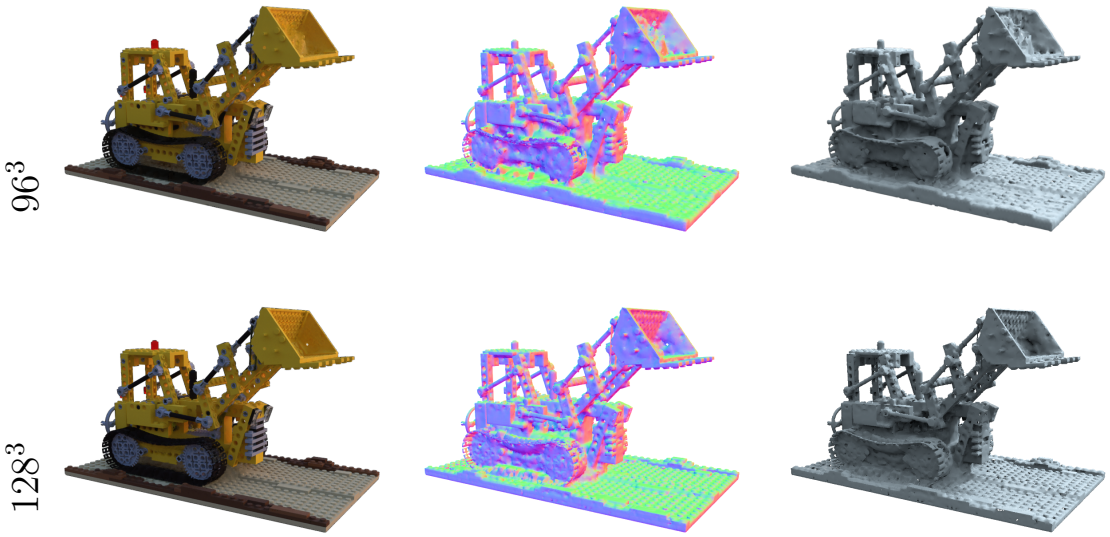


Figure 14. Resolution comparisons of *Lego* from the TensoIR Synthetic Dataset.

H. Optimization with An Existing Mesh

While the underlying geometry is modeled from scratch using isosurfaces in the standard pipeline, GeoSplatting also supports initializing the optimization process with an existing mesh. For instance, given multi-view images, one can first extract a triangle mesh using methods like NeuS [17] or GOF [18]. Subsequently, GeoSplatting can leverage this initial mesh by converting it into Gaussian points via MGadapter. The remaining steps then follow the standard GeoSplatting pipeline.

Figure 15. Extending GeoSplatting to scene levels by providing initial mesh.

The differentiable nature of MGadapter enables the refinement of the initial mesh through gradients derived from 3DGS rendering. A significant advantage of this approach is the ability to circumvent the limitations associated with isosurfaces, such as restricted resolution and mask dependencies. This capability inherently extends GeoSplatting's applicability to scene-level inverse rendering, with illustrative results presented in Figure 15.

It is important to note, however, that this two-stage extension is not applicable to specular objects. This limitation arises because surface reconstruction methods typically lack effective PBR modeling, struggling to extract accurate geometry for specular objects and leading to noisy meshes. As a result, GeoSplatting is unable to rectify geometric inaccuracies from the previous stage. Actually, the superior performance of GeoSplatting in reflective scenarios, as demonstrated in Figure 7 of the main paper, is primarily attributable to its joint optimization of geometry and BRDF material.

I. Path Tracing Adaption

In the standard GeoSplatting pipeline, we follow NVDiffrecmc [7] to use one-bounce ray tracing to evaluate the rendering equation, leaving indirect lighting terms represented as learnable SH coefficents. While this is effective, it can be also improved by incorporating path tracing for physically correct indirect lighting. Following MIRRES [4], we incorporate ReSTIR [2] to conduct efficient path tracing. As shown in Table 8 and Figure 16, this improves performance at the cost of 3x slower training speed.

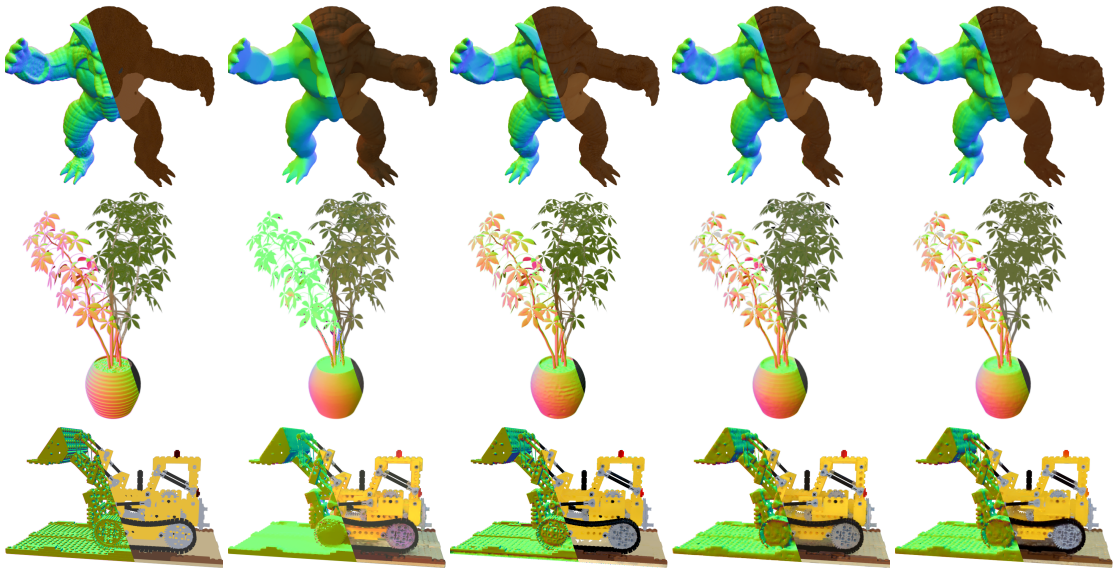


Figure 16. **Decomposition Results.** Left: Normal; Right: Albedo.

PSNR ↑	Ours	Ours+PT	R3DG [5]	MIRRES [4]
TensoIR Synthetic Albedo	29.45	**29.97**	28.74	29.24
TensoIR Synthetic Relit	29.59	30.04	28.55	**31.44**
Shiny Blender PBR	31.46	**31.60**	28.83	26.53
Training Time	∼**14min**	∼40min	∼110min	∼240min

Table 8. Quantitative Comparisons.

References

[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 7

[2] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (TOG)*, 39(4):148–1, 2020. 9

[3] Jaehoon Choi, Yonghan Lee, Hyungtae Lee, Heesung Kwon, and Dinesh Manocha. Meshgs: Adaptive mesh-aligned gaussian splatting for high-quality rendering. In *Computer Vision – ACCV 2024: 17th Asian Conference on Computer Vision, Hanoi, Vietnam, December 8–12, 2024, Proceedings, Part IX*, page 262–279, Berlin, Heidelberg, 2024. Springer-Verlag. 3

[4] Yuxin Dai, Qi Wang, Jingsen Zhu, Dianbing Xi, Yuchi Huo, Chen Qian, and Ying He. Inverse rendering for shape, light, and material decomposition using multi-bounce path tracing and reservoir sampling. In *The Thirteenth International Conference on Learning Representations*, 2025. 9

[5] Jian Gao, Chun Gu, Youtian Lin, Hao Zhu, Xun Cao, Li Zhang, and Yao Yao. Relightable 3d gaussian: Real-time point cloud relighting with brdf decomposition and ray tracing. *arXiv:2311.16043*, 2023. 5, 9

[6] Lin Gao, Jie Yang, Botao Zhang, Jiamu Sun, Yujie Yuan, Hongbo Fu, and Yu-Kun Lai. Real-time large-scale deformation of gaussian splatting. *ACM Transactions on Graphics (SIGGRAPH Asia 2024)*, 2024. 3

[7] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, Light, and Material Decomposition from Images using Monte Carlo Rendering and Denoising. *arXiv:2206.03380*, 2022. 5, 6, 9

[8] Ancheng Lin and Jun Li. Direct learning of mesh and appearance via 3d gaussian splatting, 2024. 3

[9] Jiaming Liu, Linghe Kong, Jiajie Yan, and Guihai Chen. Mesh-aligned 3d gaussian splatting for multi-resolution anti-aliasing rendering. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2025. 3

[10] Thomas Müller. tiny-cuda-nn, 2021. 4

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 4

[12] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8280–8290, 2022. 5, 6

[13] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 4

[14] Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Trans. Graph.*, 42(4), 2023. 4

[15] Rafał Tobiasz, Grzegorz Wilczyński, Marcin Mazur, Sławomir Tadeja, and Przemysław Spurek. Meshsplats: Mesh-based rendering with gaussian splatting initialization, 2025. 3

[16] Joanna Waczyńska, Piotr Borycki, Sławomir Tadeja, Jacek Tabor, and Przemysław Spurek. Games: Mesh-based adapting and modification of gaussian splatting, 2024. 3

[17] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021. 8

[18] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient adaptive surface reconstruction in unbounded scenes. *ACM Transactions on Graphics*, 2024. 8