DeferredGS: Decoupled and Relightable Gaussian Splatting with Deferred Shading

Tong Wu, Jia-Mu Sun, Yu-Kun Lai, Yuewen Ma, Leif Kobbelt and Lin Gao*

Abstract—Reconstructing and editing 3D objects and scenes both play crucial roles in computer graphics and computer vision. Neural radiance fields (NeRFs) can achieve realistic reconstruction and editing results but suffer from inefficiency in rendering. Gaussian splatting significantly accelerates rendering by rasterizing Gaussian ellipsoids. However, Gaussian splatting utilizes a single Spherical Harmonic (SH) function to model both texture and lighting, limiting independent editing capabilities of these components. Recently, attempts have been made to decouple texture and lighting with the Gaussian splatting representation but may fail to produce plausible geometry and decomposition results on reflective scenes. Additionally, the *forward shading* technique they employ introduces noticeable blending artifacts during relighting, as the geometry attributes of Gaussians are optimized under the original illumination and may not be suitable for novel lighting conditions. To address these issues, we introduce DeferredGS, a method for decoupling and relighting the Gaussian splatting representation using *deferred shading*. To achieve successful decoupling, we model the illumination with a learnable environment map and define additional attributes such as texture parameters and normal direction on Gaussians, where the normal is distilled from a jointly trained signed distance function. More importantly, we apply *deferred shading*, resulting in more realistic relighting effects compared to previous methods. Both qualitative and quantitative experiments demonstrate the superior performance of DeferredGS in novel view synthesis and relighting tasks.

Index Terms—Gaussian Splatting, Inverse Rendering, Editing.

1 Introduction

Reconstructing and editing 3D scenes from multi-view images is an important task in computer graphics and computer vision. Recently, Neural Radiance Fields (NeRFs) [1] have shown promising results in novel view synthesis, based on which a number of editing methods [2], [3], [4], [5] are proposed. However, NeRF-based methods require dense sampling of camera rays, making rendering less efficient. Gaussian splatting [6] instead approximates a scene with a set of 3D Gaussians and visualizes it with a rasterization-based renderer to achieve real-time performance. However, despite high-quality and efficient rendering, Gaussian splatting still lacks the ability of editing.

In the original Gaussian splatting representation, the appearance is approximated by a view-dependent spherical harmonic function which entangles both texture and lighting information, making separate texture and lighting editing impossible. Independent editing requires texture and lighting decomposition so that one part can be manipulated without influencing the other. However, different from 3D representations like textured meshes or NeRFs that have an explicit surface or a continuous implicit field, which can provide a good geometry basis like normal direction for the texture and lighting decomposition, Gaussian splatting models a scene's geometry with a discrete set of Gaussians, and this discrete representation makes the geometry less tractable, posing challenges for later texture and lighting decoupling.

Recently, in the context of Gaussian splatting, a few methods have made attempts to edit it. Two independent works both named GaussianEditor [7], [8] utilize the 2D segmentation model [9] and text-to-image diffusion models [10] to enable scene editing like object removal/insertion or text-guided modification similar to Instruct-NeRF2NeRF [11]. To enable better-controlled editing on texture and lighting, the works [12], [13], [14], [15] try to disentangle texture and lighting to enable individual editing for both diffuse scenes and reflective scenes. Although obtaining plausible decomposition results, they use *forward shading* that first calculates shading for each Gaussian and then blends shaded Gaussians, which can cause blending artifacts when the scene is relighted. This is because the geometry attributes of Gaussians like opacity are optimized under the input lighting condition while under novel illuminations shaded Gaussians might be blended in a wrong way (see 3rd to

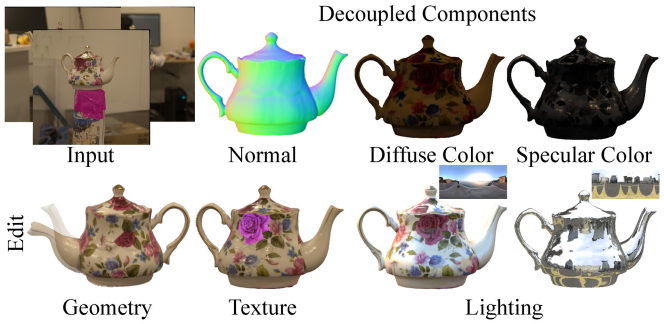


Figure 1: Given multi-view images, DeferredGS optimizes a Gaussian splatting representation with decoupled geometry, texture, and lighting. With this decoupled representation, DeferredGS not only allows geometry and texture editing, but can also render the input scene (the 3rd column, bottom row) or the edited scene (the last column, bottom row) under a novel illumination.

* *Corresponding Author is Lin Gao (gaolin@ict.ac.cn).*

6th rows in Fig. 5 for some examples). In addition, they learn geometry from the Gaussian splatting representation with the normal direction derived from the rendered depth map so their geometry may contain noise and degenerate on scenes with reflective surfaces (see the "Normal" rows in Fig. 4).

To resolve the aforementioned issues, we propose DeferredGS that decouples a scene's texture and its surrounding illumination of the Gaussian splatting representation with the *deferred shading* technique. Given multi-view captures of an input scene, we define a set of Gaussians with extra learnable texture attributes and normal directions attached to model texture and geometry respectively. To enable faithful geometry reconstruction, we distill the normal field from a signed distance function (SDF) onto the Gaussians. We also optimize a learnable environment map to approximate the surrounding lighting conditions. In the rendering process, geometry and texture buffers are first rasterized and shading is calculated at the pixel level with the *deferred shading* technique to enable faithful decoupling and editing. Despite involving more complex shading computation, DeferredGS still allows real-time rendering ($\sim$30FPS at 800×800 resolution on a single 3090 GPU).

Our contributions are as follows:

- We introduce DeferredGS, a decoupled and relightable Gaussian splatting representation, where its geometry is enhanced by a normal distillation module.
- To the best of our knowledge, DeferredGS is the first to apply the *deferred shading* technique to Gaussian splatting, which alleviates blending artifacts of previous methods.
- Experiments show that our DeferredGS produces more faithful decomposition and relighting results compared to previous methods.

2 Related Work

2.1 Neural Rendering

With the development of recent implicit representations [16], [17], [18] and volume rendering [19], neural rendering has become popular in 3D scene reconstruction. The pioneering work Neural Radiance Field (NeRF) [1] models a sample point's density and view-dependent color with two neural networks and renders the scene by integrating sample points' colors along the rays. NeRF can synthesize realistic novel view results but dense sampling and network query make both training and rendering slow. To improve the efficiency, a few *hybrid* rendering methods bake view-independent features onto regular or irregular geometry proxies. NSVF [20] models 3D scenes with an octree structure and prunes voxels with low-density values so that sample points outside the octree can be ignored in the rendering process. DVGO [21] stores density values and appearance features on two voxel grids and directly optimizes these two voxel grids to reconstruct 3D scenes. TensoRF [22] models a scene with a set of matrices and vector pairs that depict the scene's geometry and appearance on corresponding planes, which not only accelerates the training speed but also reduces the storage size. Instant-NGP [23] instead utilizes multi-resolution hash encoding as its scene representation and enables training convergence in as little as 5 seconds. MobileNeRF [24] and BakedSDF [25] share a similar idea to first reconstruct the mesh of a scene and attach learnable appearance information onto mesh vertices, which enables real-time rendering on consumer devices. Moving a step forward, Gaussian splatting [6] abandons the neural network and models a 3D scene as a set of 3D Gaussians on which attributes like position, rotation, scaling, opacity, and spherical harmonic coefficients can be tuned. Gaussian splatting uses a rasterization-based renderer which can produce rendered results from a viewpoint in real time. To better reconstruct reflective scenes, GaussianShader [14] proposes to model view-dependent effects separately similar to RefNeRF [26] and Ye *et al.* [27] further introduce deferred rendering for reflection modeling. Despite the high-quality novel view results, these methods still lack the capability of editing, making appearance change or relighting impossible.

2.2 Neural Surface Reconstruction

Although NeRF and Gaussian splatting can synthesize realistic novel view results, they do not have an explicit surface representation, limiting their application in the traditional graphics pipeline. Therefore, a few methods propose to build the connection between the distance field and the density field in volume rendering. UniSurf [28] models the geometry as an occupancy function and replaces alpha values in volume rendering with occupancy values. NeuS [29] instead treats the geometry as a signed distance field (SDF) and proposes an unbiased and occlusion-aware transformation from SDF to density field. VOLSDF [30] also uses SDF as its geometry representation but is based on a Laplace distribution's Cumulative Distribution Function. To reconstruct open surfaces, several concurrent works [31], [32], [33] introduce the unsigned distance field (UDF) into the training of NeRF. However, for shiny scenes, these methods may produce unsatisfactory results like concave surfaces. To address this, the works [34], [5], [35] decompose the appearance of a scene into view-independent appearance and view-dependent appearance based on RefNeRF [26] and better reconstruct shiny scenes. In the field of Gaussian splatting, NeuSG [36] introduces a NeuS [29] branch to align Gaussians' normals with corresponding normal directions in NeuS but requires a long training time (16 hours) compared to the original Gaussian splatting. SuGaR [37] instead introduces a depth self-regularization loss term to constrain Gaussians to lie on the surface. 2DGS [38] further improves the geometric reconstruction quality by utilizing flattened 2D Gaussian ellipsoids as the basic primitive. GaussianSurfel [39] shares a similar basic representation but further introduces monocular normal estimation as an extra prior. GOF [40] builds up an implicit opacity field based on the Gaussian Splatting representation and uses Marching Tetrahedra algorithm for surface extraction.

2.3 Inverse Rendering and Relighting

The goal of inverse rendering is to recover geometry, material and lighting information from a set of images of a single scene under the same or different illuminations.

NeRV [41] is the pioneering work that decomposes geometry and BRDF (Bidirectional Reflectance Distribution) material under a given lighting condition. To reduce dependence on light input, later methods work under unknown light conditions. NeRD [42] and PhySG [43] use Spherical Gaussian (SG) as the light representation and utilize a density field and a signed distance field as their geometry representations. Based on PhySG, InvRender [44] further takes visibility into consideration to better recover material information in occluded regions. NeROIC [45] approximates the illumination with Spherical Harmonic functions and enables training on Internet images under different lighting conditions. NeRFactor [46] represents the lighting with a low-resolution image. To better estimate material from images, it also trains a BRDF autoencoder. NDR [47] and NDRMC [48] use a hybrid geometry representation DMTet [49] to enable efficient inverse rendering. NMF [50] and TensoIR [51] introduce the importance sampling strategy for effective shading calculation. To model reflective objects, NeRO [52] and DE-NeRF [5] express high-frequency lighting as MLP networks. Apart from works concentrating on object-level scenes, there are methods [53], [54], [55] targeting large-scale outdoor scenes. Different from those methods learning decomposition from just multi-view images to enable the relighting task, recent methods introduce data priors from pre-trained models. Some of them [56], [57] use score-based distillation [58], [10] optimization to assist the inverse rendering process. Instead of optimization, DiLightNet [59] inserts outgoing radiance map as control signals in the ControlNet [60] to directly relight an image. IC-Light [61] instead takes light maps as the condition in the ControlNet [60] to enable realistic single-view relighting. Later, NeuralGaffer [62] and IllumiNeRF [63] extend this idea to multi-view settings by resolving the inconsistency between relighting results from different viewpoints. Instead of operating on multi-view images, FlashTex [64] takes a mesh as input and trains a multi-view light control net to relight it. There are also concurrent inverse rendering works based on the Gaussian splatting representation like [12], [13], [15], [14]. Different from them, our method introduces a normal distillation module to enable more accurate normal estimation and alleviate blending artifacts with the *deferred shading* technique when relighting Gaussians.

3 METHOD

We propose DeferredGS, a decoupled Gaussian splatting representation that decomposes geometry, texture, and lighting from multi-view images, which enables downstream applications like scene relighting and texture editing. The overview of our method is shown in Fig. 2. First, we briefly introduce the rendering formulation in Gaussian splatting and define the texture parameters we use in rendering (Sec. 3.1). Since directly optimizing everything from scratch can lead to suboptimal results, especially on the geometry side, we introduce a normal field distillation module (Sec. 3.2) that utilizes an SDF network to guide Gaussians to a better surface representation. With geometry set up properly, we bind texture parameters onto the Gaussian splatting representation and perform *deferred shading* to obtain plausible decomposition (Sec. 3.3). Finally, we demonstrate how this decomposition enables scene editing applications like relighting and texture editing (Sec. 3.4).

3.1 Gaussian Splatting and Texture Definition

Gaussian splatting models a 3D scene by a set of Gaussians with which attributes like position P, covariance Σ, scale S, opacity α and Spherical Harmonic coefficients (SH) representing view-dependent appearance are associated. It uses a point-based α-blending rendering formulation similar to NeRF [1].

$$C_* = \sum_{i=1}^{N} *_i \alpha_i T_i \tag{1}$$

where N is the number of sample points on a ray. $T_i = \prod_{j=1}^{i-1}(1-\alpha_j)$ is accumulated transmittance and $*$ can be any attribute of the sample point to be blended such as depth, color and normal direction. Gaussian splatting avoids densely sampling on the ray and efficiently projects 3D Gaussians onto a 2D image plane and approximates 3D scenes by blending view-dependent color represented by SH. Instead of modeling the appearance as a whole with SH, our DeferredGS replaces SH with optimizable texture parameters including diffuse albedo k_d, roughness r, and specular albedo k_s as shown in Fig. 2 to enable texture and lighting decoupling.

3.2 Normal Field Distillation

Normal estimation is important for extracting correct texture and lighting information. However, it is challenging to obtain plausible surface normals in Gaussian splatting as mentioned in Sec. 1 since it is a discrete representation and only promotes rendering quality without constraining the geometry. To resolve this issue, we propose a normal field distillation module that integrates current multi-view geometry reconstruction techniques with Gaussian splatting. Specifically, we jointly train a NeRF-like network equipped with multi-level hash encoding [23] namely *Instant-RefNeuS* and a Gaussian splatting representation. For the *Instant-RefNeuS*, we utilize the signed distance field (SDF) as the geometry representation which can be parameterized as a neural network $f_s(x)$ where x is a sample point in 3D space. For the color branch, instead of modeling all the appearances with a single view-dependent network like NeRF [1], we separate the appearance into a view-independent branch and a view-dependent branch to avoid geometric artifacts like concave surfaces on reflective scenes [35], [34]. The view-independent branch predicts the diffuse color c_d and the specular tint p for a sample point x. The view-dependent branch takes the view direction as input and outputs the specular lighting c_l. The final color can be composed by $c' = c_d + pc_l$. For better understanding, all terms with superscript $'$ refer to the NeRF network while terms without refer to the Gaussian splatting representation. To render a pixel $C'_c(v)$, we integrate colors of sample points on a ray v with Eqn. (1). Adopting the occlusion-aware and unbiased volume rendering technique from NeuS [29], the SDF value can be transformed to volume density by $\sigma(t) = \max\left(-\frac{\frac{d\Phi_s}{dt}(f(x(t)))}{\Phi_s(f(x(t)))}, 0\right)$, where $\Phi_s(x) = (1+e^{-sx})^{-1}$ and s is a trainable deviation parameter. The opaque value

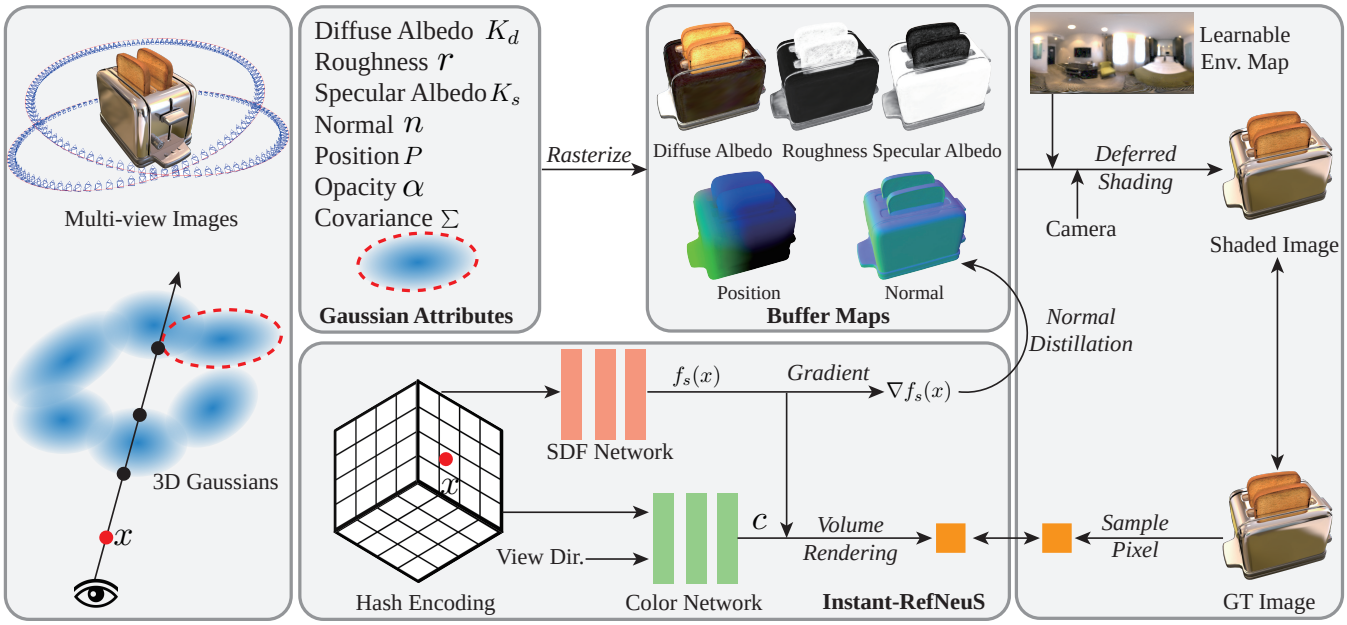


Figure 2: Overview of DeferredGS. Given multi-view images, DeferredGS builds a decoupled Gaussian splatting representation where auxiliary texture attributes are defined on each Gaussian. To enable successful geometry and appearance separation, we attach an extra normal direction to each Gaussian and optimize it by distilling the normal field from a jointly trained signed distance function. For texture and lighting decomposition, DeferredGS rasterizes geometry and texture attributes into buffer maps and computes shading at the pixel level under the illumination of a learnable environment map with *deferred shading*.

for point x_i is derived from the density function $\sigma(t)$ by $\alpha_i = 1 - exp\left(-\int_{t_i}^{t_{i+1}} \sigma(t)dt\right)$. The loss function for *Instant-RefNeuS* is as follows:

$$L_{nerf} = \sum_{v \in V} \|C'_c(v) - C^t(v)\| + \lambda \sum_{v \in V} \sum_{i=1}^{M} \left\| \|\nabla_{x_{v,i}}\| - 1 \right\|_2^2, \tag{2}$$

where V is the number of rays in a batch. M is the number of sample points on a single ray and $C^t(v)$ is the corresponding ground truth color. The second term is the Eikonal loss [65], [66], where $\|\nabla f_s(x_{v,i})\|$ is the spatial norm of the SDF network $f_s(x)$'s gradient at the ith sample point $x_{v,i}$ on the ray v. In order to distill the normal information from *Instant-RefNeuS* to Gaussian splatting, we define an extra normal direction n on each Gaussian as shown in Fig. 2 and render the normal C_n of Gaussian splatting for a pixel using Eqn. (1). Then we distill the normal field Gaussian splatting by minimizing the difference between C_n and the corresponding rendered normal in *Instant-RefNeuS*:

$$L_{nd} = \sum_{v \in V} \|1 - C_n(v) \cdot C'_n(v)\| = \sum_{v \in V} \|1 - C_n(v) \cdot C_{\nabla x_{v,i}}\| \tag{3}$$

where $\cdot$ is the dot product operator.

3.3 Deferred Shading in Gaussian Splatting

For shading calculation, we follow the rendering equation [67]:

$$L_o(\omega_o) = \int_\Omega L_i(\omega_i) f(\omega_i, \omega_o)(\omega_i \cdot n) d\omega_i \tag{4}$$

where $L_o(\omega_o)$ and $L_i(\omega_i)$ represent outgoing lighting in direction ω_o and incident lighting from direction ω_i respectively. $f(\omega_i, \omega_o)$ is the BRDF at a point which can be parameterized by the Disney shading model [68]:

$$f(\omega_i, \omega_o) = \frac{k_d}{\pi} + \frac{DFG}{4(\omega_i \cdot n)(\omega_o \cdot n)} \tag{5}$$

where D, F, G correspond to normal distribution function, Fresnel term, and geometry term. Their detailed calculations can be found in [69].

We model the scene's environment lighting with a $6 \times 512 \times 512$ High Dynamic Range (HDR) cube map and utilize the SplitSum [70] approximation which separates the integrals of lighting and BRDF to enable efficient shading calculation. T he diffuse color c_{diff}:

$$c_{diff} = \frac{k_d}{\pi} \int_\Omega L_i(\omega_i)(\omega_i \cdot n) d\omega_i \tag{6}$$

which is only dependent on the normal direction n and can be pre-computed and stored in a 2D texture.

For the specular color c_{spec}:

$$\begin{aligned} c_{spec} &\approx Int_{light} \cdot Int_{BRDF} \\ &= \int_\Omega L_i(\omega_i) D(\omega_i, \omega_o)(\omega_i \cdot n) d\omega_i \cdot \int_\Omega f(\omega_i, \omega_o)(\omega_i \cdot n) d\omega_i \end{aligned} \tag{7}$$

where Int_{light} represents the integral of incident light with the normal distribution function D. Given a fixed environment map, Int_{light} is only related to the roughness value r and therefore can be pre-computed and stored in a mipmap where each mip level corresponds to a fixed roughness value. Int_{BRDF} is the integral of specular BRDF under a

uniform white environment lighting. It is determined by the roughness value in the BRDF and the dot product between incident light direction and normal direction $\omega_i \cdot n$. Again it can be pre-computed and stored in a 2D texture and efficiently looked up in rendering. The final shaded color is $c_g = c_{diff} + c_{spec}$.

A straightforward way to obtain a decoupled Gaussian splatting representation is performing *forward shading* for each Gaussian to get its color c_i and rasterizing Gaussians with Eqn. (1) similar to previous works [44], [46], [15], [14]. Though obtaining plausible decoupled results, we observe notable artifacts when rendering the decoupled Gaussians under a novel lighting condition (see "F.S." columns in Fig. 8 and "RGS" and "GShader" columns in Fig. 5). This is because even though Eqn. (3) constrains the rasterized normal to be close to that of *Instant-RefNeuS* at pixel level, the rasterized normal is still blended from multiple Gaussians' normals. The geometry attributes like position, covariance and opacity may overfit the given lighting condition to produce proper blended normal and shaded color and it does not guarantee that each Gaussian's individual normal or shaded color is correct. Thus when changing the lighting condition, the trained geometry attributes may not work for the shaded colors under a new lighting condition and blend them in a wrong way.

To resolve this issue, we introduce the *deferred shading* technique into the rendering of Gaussians. The core difference between *forward shading* and *deferred shading* is how shading is computed for a single pixel during rendering. In *forward shading*, shading is applied to all Gaussian ellipsoids along a ray, and the pixel's appearance results from blending the shaded outputs of multiple Gaussian ellipsoids. To achieve accurate shading for a pixel, it is essential that all shaded colors from the Gaussian ellipsoids are correct, which requires that their intrinsic properties, including geometry and texture, are accurate. In contrast, *deferred shading* calculates shading only once per pixel. As long as the blended properties at the pixel level are accurate, it can yield reliable results without needing to ensure that the geometry and texture properties of all Gaussian ellipsoids are correct. In addition, in deferred shading, the blending operator is applied to view-independent quantities (like normals) while in forward shading, the blending operator is applied to view-dependent quantities (shaded colors). As a consequence, deferred shading is more robust against view-dependent artifacts than forward shading. In some sense this is comparable to Gouraud Shading and Phong Shading where the former interpolates colors while the later interpolates normals. On specular surfaces (with significant view dependence) Gouraud Shading shows strong artifacts which Phong Shading avoids. Specifically, we first rasterize components like position P, normal n, diffuse albedo k_d, roughness r, specular albedo k_s, and opacity α into 2D pixels $C_P, C_n, C_{k_d}, C_r, C_{k_s}$, and C_α with Eqn. (1). By aggregating all pixels, we get corresponding 2D maps I_P, I_n, I_{k_d}, I_r, I_{k_s}, and I_α. Then we calculate the shaded color C for a pixel with Eqn. (4). Note that since computing shading for every single pixel in the image is time-consuming, we only perform shading on pixels with C_α bigger than 0.5. Overall, we optimize the following loss:

$$\begin{aligned} L =& L_{nerf} + \lambda_{nd} L_{nd} + \lambda_{L1} L_{L1} + \lambda_{ssim} L_{ssim} + \\ & \lambda_{mask} L_{mask} + \lambda_{TV} L_{TV} \end{aligned} \tag{8}$$

where L_{L1} and L_{ssim} are the L1 loss and D-SSIM loss between the rasterized image and the ground truth image same as those in [6]. $L_{mask} = \|I_\alpha - I_m\|_1$ is the mask loss that encourages Gaussians to lie in the foreground and I_m is the ground truth mask image. L_{TV} is the total variation loss applied on images $I_{k_d}, I_{k_s}, I_{k_s}$ to encourage smooth texture estimation.

3.4 Editing Decoupled Gaussians

After optimizing the attributes on Gaussians, we obtain a decoupled Gaussian splatting representation and can render the original scene with edited texture or geometry under a novel illumination.

3.4.1 Relighting

As we extract the environment map of the original scene, we can replace it by a novel environment map with little effort and render the scene with novel illumination using *deferred shading*.

3.4.2 Geometry Editing

For geometry editing, instead of directly operating on Gaussians, we first extract a mesh with *Instant-RefNeuS* and use it as a geometry proxy to guide the deformation of Gaussians. Specifically, we first deform the mesh using the As-Rigid-As-Possible (ARAP) deformation algorithm [71] which solves for rotations and translations of mesh vertices. Then we find the closest neighbor for each Gaussian on the original mesh surface, whose rotation and translation can be obtained via barycentric mapping (based on the mesh triangle that contains the neighboring point). Finally, we deform each Gaussian by applying the rotation and the translation of its corresponding closest neighbor to Gaussian's position, covariance matrix, and normal direction. By rasterizing deformed Gaussians, we can render the deformed scene.

3.4.3 Texture Editing

For texture editing, we follow the editing paradigm in traditional 3D modeling software [72] to paint the scene from a given viewpoint. Our texture editing supports changes on all texture components. We first determine which Gaussians need to be adjusted by considering both whether they lie in the editing mask and the difference between the Gaussian depth and the corresponding depth in the depth map rendered with Eqn. (1). Then we can optimize these Gaussians' texture attributes $*$ by:

$$\arg\min_{*} \|I^i_* - I^i_e\|, * \in \{k_d, r, k_s\} \tag{9}$$

where I^i_* denotes a rendered texture map, e.g. diffuse albedo map, from the input editing viewpoint i. I^i_e is the edited texture map from viewpoint i by putting edited content onto I^i_*.

However, this naive solution only considers the input editing viewpoint and may produce blending artifacts when we view the optimized Gaussians from other viewpoints.

Therefore, we propose to augment the optimization with random view inputs. That is to say, we generate edited rendered results from multiple random viewpoints by first unprojecting the edited content onto the mesh extracted by *Instant-RefNeuS* to form a textured mesh and rendering the textured mesh from random viewpoints. So the final editing loss is:

$$\arg\min_{*} \sum_{j \in L} \|I_*^j - I_e^j\|, * \in \{k_d, r, k_s\} \tag{10}$$

where set L includes both the input viewpoint and randomly sampled viewpoints.

4 Results and Evaluations

4.1 Datasets and Metrics

We conduct qualitative and quantitative experiments on two synthetic datasets, NeRF Synthetic [1] and Shiny Blender [26] and one real dataset, the Stanford ORB dataset [73]. For all datasets, we evaluate DeferredGS and baseline methods on novel view synthesis, decomposition and relighting tasks. For quantitative comparisons, we use PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity) [74] and LPIPS (Learned Perceptual Image Patch Similarity) [75] metrics to compare the similarity between images. To evaluate the quality of the estimated normals, we compare the estimated normal images with the ground truth with Mean Squared Error (MSE).

4.2 Implementation Details

We train the decoupled Gaussian splatting representation on a single 3090 GPU with 24GB memory. The training process takes 3-4 hours depending on the scenes' complexity and the input images' resolution. DeferredGS allows $\sim$30FPS novel view synthesis at 800×800 resolution on a single 3090 GPU. For hyperparameter settings, in Eqn. (2), λ_1 is set to 0.1. In Eqn. (8), $\lambda_{nd}, \lambda_{L1}, \lambda_{ssim}, \lambda_{mask}, \lambda_{TV}$ are set to $0.5, 0.8, 0.2, 0.5, 0.001$ in all our examples.

4.3 Novel View Synthesis

We present novel view synthesis results on different cases' test splits in Fig. 3 and compare our method with NeRFactor [46], TensoIR [51], NDR [47], NeRO [52], Gaussian Splatting [6], RelightableGaussian [12], GuassianShader [14], Ye at al. [27], and a 2DGS [38] variation with physically based rendering namely 2DGS_{pbr}. NeRFactor models the geometry with a density function and utilizes a low-resolution environment map as its lighting representation, which leads to less convincing geometry (first row in Fig. 3) and blurry reconstruction results. TensoIR is unable to synthesize appearance details with its smooth lighting representation approximated by Spherical Gaussian functions. NDR uses deformable tetrahedra to approximate the geometry which may cause irregular geometry and noisy novel view synthesis results. NeRO produces blurry appearance (second and fourth rows) and may reconstruct wrong geometry (first row). Gaussian Splatting models the texture and lighting with a single spherical harmonic function which may fail to generalize to novel views and produce wrong reflections (last row in Fig. 3). RelightableGaussian and GaussianShader separately model texture and lighting but learn geometry from scratch, which causes blurry reconstruction. As for quantitative comparisons, we compare novel view reconstruction results by different methods with the ground truth image using PSNR, SSIM and LPIPS metrics in Table 1. Our method achieves comparable results with baseline methods on the NeRF synthetic dataset and outperforms other baselines on the shiny blender dataset.

4.4 Decomposition

To evaluate the decoupling ability of our method, we render optimized diffuse albedo and normal images and compare them with the corresponding ground truth images. Baseline methods include NeRFactor [46], TensoIR [51], NDR [47], NeRO [52], RelightableGaussian [12], GuassianShader [14], and 2DGS_{pbr} that model texture and lighting separately. As shown in Fig. 4, the smooth or low-resolution lighting representation in NeRFactor and TensoIR may cause incorrect geometry or texture estimation. NDR still suffers from its tetrahedral representation and produces an irregular normal estimation and wrong texture estimation. NeRO produces faithful geometry but less correct texture estimation (first to fourth rows). For RelightableGaussian and GaussianShader, the normals are directly learned from the Gaussian splatting representation and can be rough on shiny scenes, leading to wrong textures. Note that GaussianShader decomposes diffuse color instead of diffuse albedo. Overall our method produces smooth normal estimation without losing geometric details and the extracted textures have less lighting baked in. We also report the PSNR, SSIM and LPIPS metrics for the extracted diffuse albedo components and the MSE value for the normal image in Table 2 and our method comes on top. Note that before evaluating the albedo component, we compute the scales between estimated normals and the ground truth normals following NeRFactor [46] except for glossy scenes ('materials' and 'mic' in the NeRF Synthetic dataset and 'ball' and 'toaster' in the Shiny Blender dataset) since the scales would be all 0 and result in infinite PSNRs for all methods.

4.5 Editing

We show relighting results and compare with NeRFactor [46], TensoIR [51], NDR [47], NeRO [52], RelightableGaussian [12], GuassianShader [14], 2DGS_{pbr}, and NeuralGaffer (NG) in Fig. 5. Limited by the lighting representations, NeRFactor and TensoIR are unable to render high-frequency reflections. The irregular geometry of NDR makes the scene shaded in a wrong way. NeRO can produce realistic reflection but its material estimation is less accurate, leading to less faithful relighting (third to fourth rows). In addition, it directly uses the mesh extracted by an SDF network for relighting, which may produce level-set stripe-like artifacts (better viewed in video) or miss geometry details like the metal frame on the side window in the car example. GaussianShader and RelightableGaussian produce less faithful normal and texture estimation in the decomposition process and the *forward rendering* technique

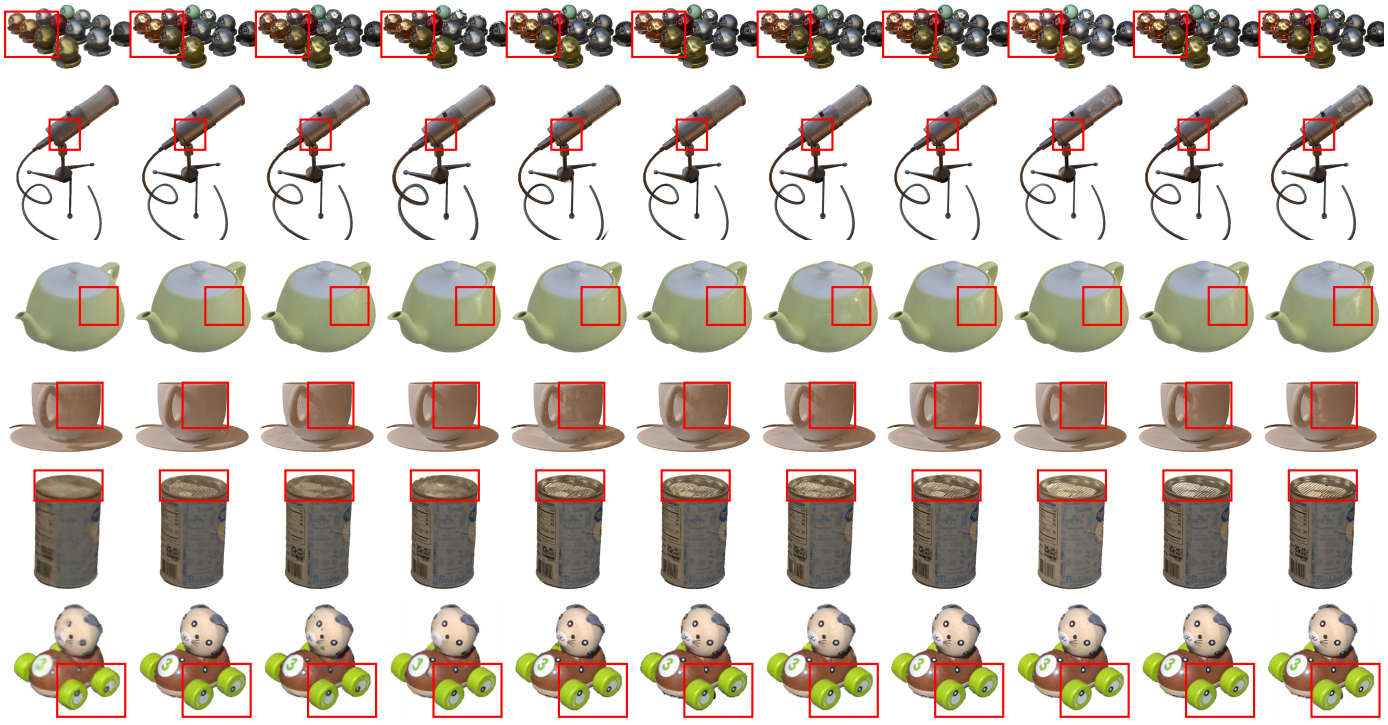


Figure 3: Novel view synthesis comparisons with NeRFactor [46], TensoIR [51], NDR [47], NeRO [52], Gaussian Splatting (GS) [6], RelightableGaussian (RGS) [12], GaussianShader (GShader) [14], Ye at al. [27], and a 2DGS [38] variation 2DGS$_{pbr}$. The bottom two rows are from the real Stanford ORB dataset [73].

Table 1: Quantitative comparison of novel view synthesis results using SSIM, PSNR and LPIPS metrics on NeRF Synthetic, Shiny Blender and Stanford ORB datasets. Cells are colored as best, second best and third best.

Methods	NeRF Synthetic			Shiny Blender			Stanford ORB		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
NeRFactor	27.86	0.924	0.044	27.04	0.913	0.123	30.01	0.934	0.079
TensoIR	29.52	0.944	0.050	28.01	0.905	0.131	34.16	0.980	0.020
NDR	29.05	0.939	0.081	28.11	0.935	0.076	32.84	0.985	0.014
NeRO	29.03	0.942	0.051	30.96	0.953	0.045	29.25	0.970	0.019
GS	33.30	0.969	0.030	30.35	0.944	0.088	35.77	0.991	0.007
RGS	29.25	0.941	0.042	28.69	0.930	0.081	36.65	0.992	0.006
GShader	30.84	0.933	0.067	30.72	0.951	0.069	35.43	0.990	0.008
Ye et al.	32.03	0.967	0.045	32.58	0.961	0.057	37.88	0.993	0.007
2DGS$_{pbr}$	29.07	0.943	0.046	29.75	0.947	0.073	34.32	0.989	0.011
Ours	32.29	0.971	0.039	32.61	0.968	0.041	37.93	0.995	0.006

Table 2: Quantitative comparison of decomposed diffuse albedos and normals by different methods. Diffuse albedo is evaluated with PSNR, SSIM and LPIPS metrics while estimated normals are evaluated with MSE. Note that GaussianShader (GShader) [14] only decomposes diffuse color instead of diffuse albedo so its albedo results are unavailable. Cells are colored as best, second best and third best.

Methods	NeRF Synthetic				Shiny Blender				Stanford ORB			
	Diffuse Albedo			**Normal**	**Diffuse Albedo**			**Normal**	**Diffuse Albedo**			**Normal**
	PSNR↑	SSIM↑	LPIPS↓	MSE↓	PSNR↑	SSIM↑	LPIPS↓	MSE↓	PSNR↑	SSIM↑	LPIPS↓	MSE↓
NeRFactor	19.01	0.871	0.086	0.098	18.28	0.854	0.106	0.132	28.62	0.958	0.046	0.067
TensoIR	20.29	0.871	0.103	0.056	20.43	0.879	0.194	0.068	29.11	0.969	0.026	0.029
NDR	18.34	0.859	0.116	0.064	19.66	0.865	0.192	0.058	30.67	0.971	0.023	0.043
NeRO	19.69	0.866	0.137	0.135	18.31	0.867	0.187	0.037	24.70	0.948	0.045	0.144
RGS	18.13	0.862	0.115	0.072	18.10	0.847	0.198	0.065	29.33	0.979	0.025	0.049
GShader	-	-	-	0.115	-	-	-	0.093	-	-	-	0.037
2DGS$_{pbr}$	18.04	0.860	0.115	0.061	19.07	0.867	0.202	0.086	30.88	0.975	0.028	0.033
w/o NFD	16.25	0.821	0.138	0.143	16.58	0.839	0.189	0.102	29.04	0.964	0.039	0.081
Ours	19.14	0.878	0.109	0.053	20.65	0.883	0.188	0.037	31.99	0.977	0.021	0.024

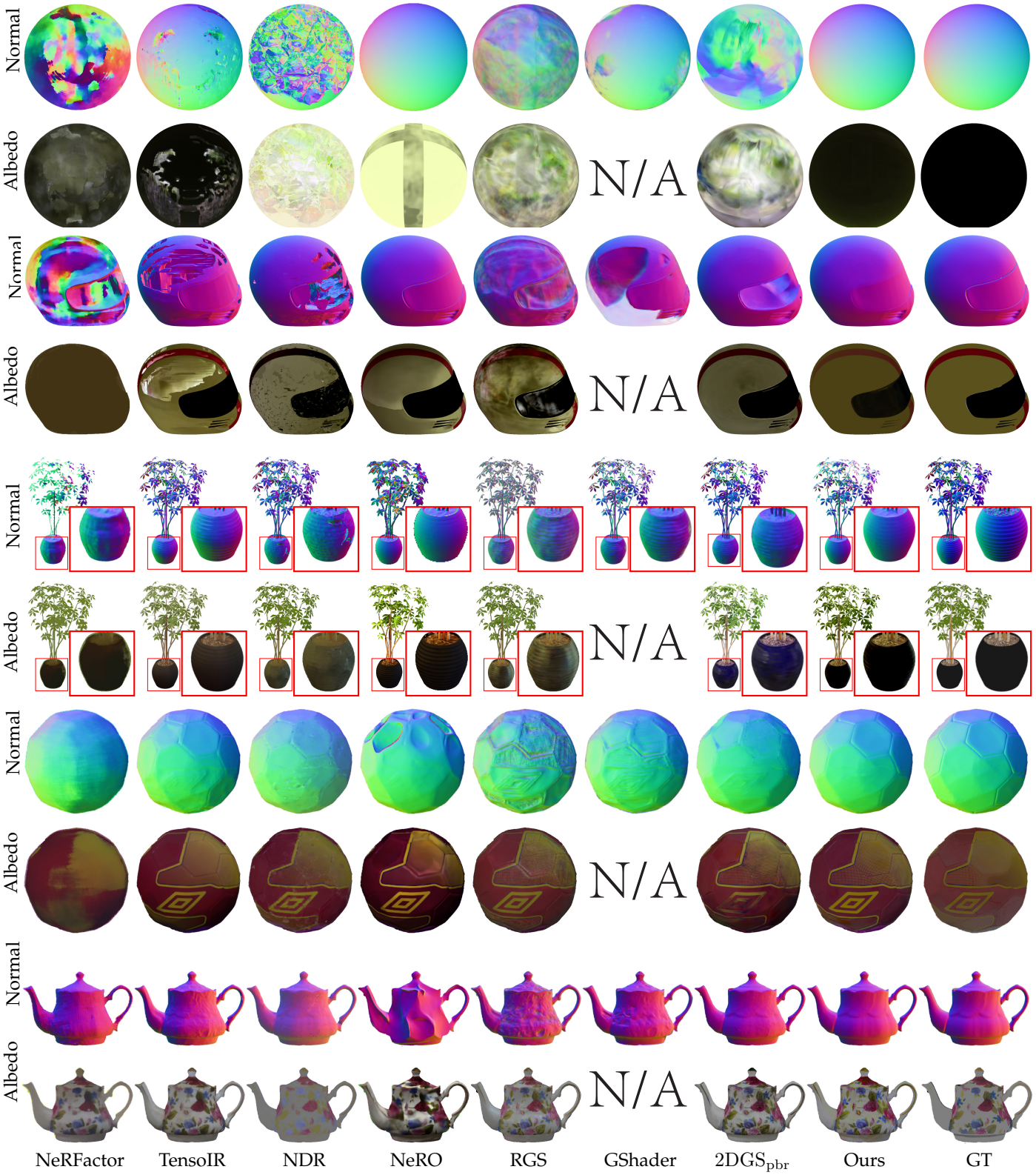


Figure 4: Decomposed result comparisons with NeRFactor [46], TensoIR [51], NDR [47], NeRO [52], RelightableGaussian (RGS) [12], GaussianShader (GShader) [14], and a 2DGS [38] variation 2DGS$_{\text{pbr}}$. In every two rows, we show decomposed normal and diffuse albedo components by different methods and compare them with the ground truth. Note that GaussianShader only decomposes diffuse color instead of diffuse albedo so its diffuse albedo results are unavailable.

they apply causes blending artifacts even if the normal estimation is plausible (zoom-in views of the car example in Fig. 5). NeuralGaffer (NG) relights 3D objects by relighting images from different viewpoints. However, in

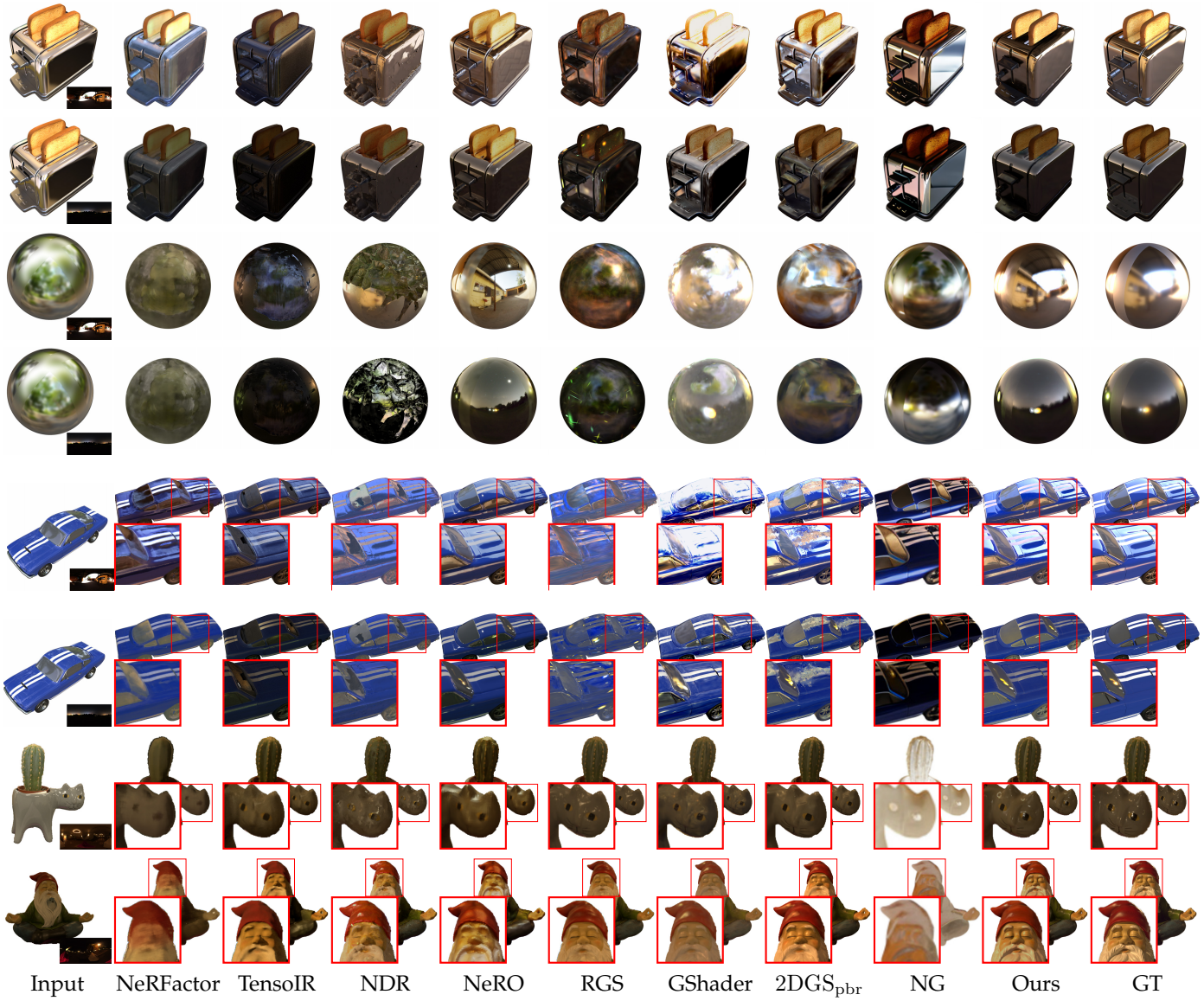


Figure 5: Relighting comparisons with NeRFactor [46], TensoIR [51], NDR [47], NeRO [52], RelightableGaussian (RGS) [12], GaussianShader (GShader) [14], a 2DGS [38] variation 2DGS$_{\text{pbr}}$, and NeuralGaffer (NG) [62]. In the first column, we show the input scene and target environment map. Relighting results by different methods and the ground truth are in other columns.

Table 3: Quantitative comparison of relighting results using SSIM, PSNR, and LPIPS metrics. Results are averaged over ten different viewpoints with eight different environment maps on NeRF Synthetic and Shiny Blender datasets. For the Stanford ORB dataset, relighting results are evaluated on the provided 20 image-envmap pairs. Cells are colored as best, second best and third best.

Methods	NeRF Synthetic			Shiny Blender			Stanford ORB		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
NeRFactor	18.40	0.868	0.094	19.79	0.901	0.086	30.03	0.970	0.028
TensoIR	20.25	0.916	0.071	19.17	0.887	0.150	29.60	0.969	0.026
NDR	19.89	0.899	0.066	20.70	0.921	0.063	30.41	0.968	0.023
NeRO	20.09	0.915	0.067	21.86	0.929	0.059	29.67	0.966	0.028
RGS	18.27	0.859	0.073	18.03	0.832	0.155	29.86	0.975	0.018
GShader	18.12	0.838	0.145	20.35	0.917	0.042	30.72	0.976	0.019
2DGS$_{\text{pbr}}$	18.14	0.841	0.088	21.07	0.916	0.067	30.65	0.977	0.019
NeuralGaffer	17.75	0.794	0.129	19.52	0.861	0.144	29.74	0.969	0.025
Forward Shading	18.21	0.833	0.070	20.05	0.914	0.053	29.94	0.970	0.024
w/o NFD	18.08	0.836	0.093	19.97	0.905	0.094	29.08	0.963	0.023
Ours	19.24	0.904	0.057	22.24	0.939	0.049	31.11	0.978	0.017

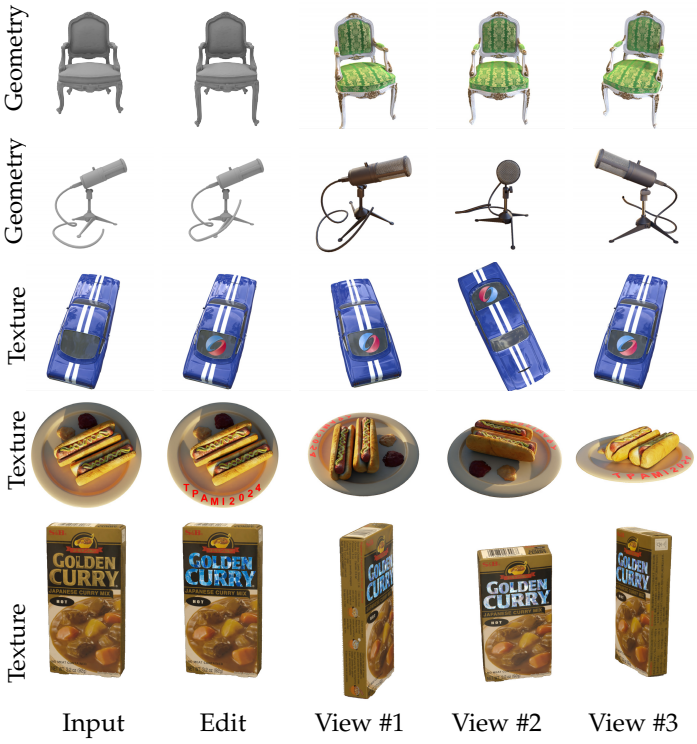


Figure 6: Geometry editing and texture editing results. For each row, we show input and edited geometry/image in the first two columns, and in the remaining three columns we show edited Gaussians rendered from three different viewpoints.

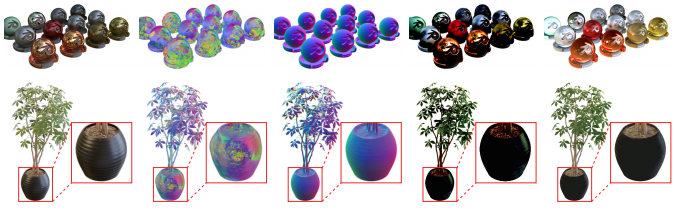
some cases, the relighting network cannot relight the input images properly and leaves reflections baked in as shown in the 3rd and 4th rows in Fig. 5. In contrast, the relighting results produced by our method are more realistic. This is because the geometry of DeferredGS distilled from a signed distance function, which works as a good basis for the geometry, texture and lighting decomposition. In addition, DeferredGS uses the *deferred shading*, which avoids blending artifacts when rendering Gaussians in a new illumination. Quantitative results on the relighting task are reported in Table 3 and our method achieves comparable results with NeRO [52]. We show geometry and texture editing results in Fig. 6. For texture editing results in the last two rows, the editing is performed on the diffuse component.

4.6 Ablation

In this subsection, we conduct several ablation studies to prove the effectiveness of our design choices.

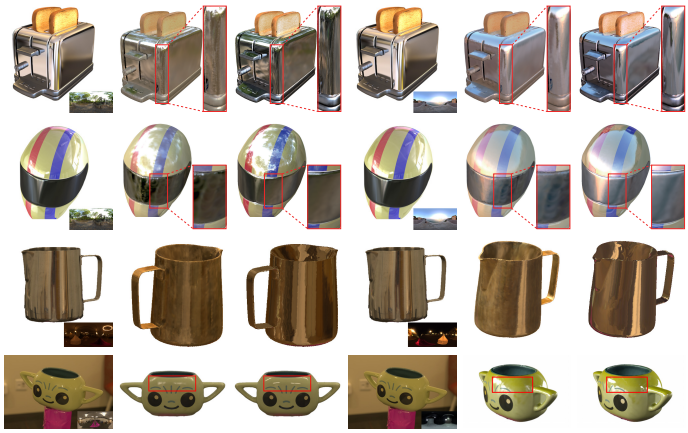
4.6.1 Normal Field Distillation

To produce a plausible geometry for the decomposition problem, we distill the normal direction from a signed distance function by encouraging the rendered normal of Gaussians to be consistent with the normal of a signed distance function. Here, we generate the decomposed results without normal field distillation (w/o NFD) and compare them with the full pipeline qualitatively in Fig. 7. We also report numerical results of decomposed results in Table 2



(a) Input (b) w/o NFD (c) w/ NFD (d) w/o NFD (e) w/ NFD

Figure 7: Qualitative comparisons between decomposed normals and diffuse albedos with normal field distillation (w/ NFD) and without normal field distillation (w/o NFD).



(a) Input (b) F.S. (c) D.S. (d) Input (e) F.S. (f) D.S.

Figure 8: Qualitative comparisons between relighting results with *forward shading* (F.S.) and with *deferred shading* (D.S.). The relit scene contains blending artifacts when we use *forward shading* while *deferred shading* can avoid them. The bottom two rows are from the real Stanford ORB dataset [73].

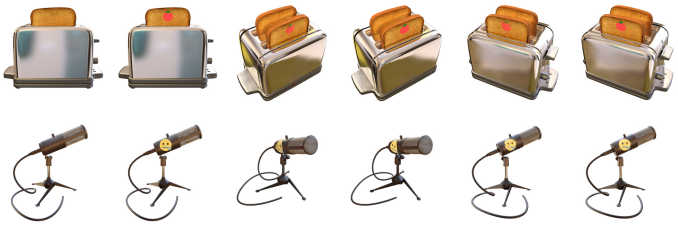
and the decomposition quality sees a drop without normal field distillation.

4.6.2 Deferred Shading

In the decomposition and relighting process, we utilize the *deferred shading* technique instead of *forward shading* commonly used in previous methods. Here we ablate its influence on the relighting results in Fig. 8. Even though the geometry is enhanced by the normal field distillation, we can still observe notable Gaussian shape—like blending artifacts when we apply the *forward shading* technique. By contrast, the *deferred shading* technique we use can successfully resolve this issue and produce smooth and realistic relighting results. We also evaluate it quantitatively in Table 3 and the relighting quality drops when it is not applied.

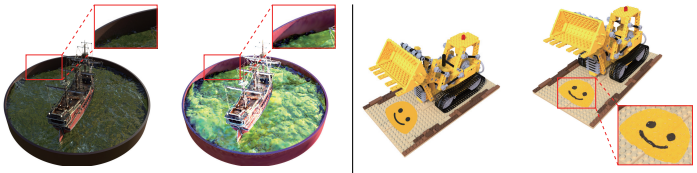
4.6.3 Random View Inputs in Texture Editing

As mentioned in Sec. 3.4.3, we perform texture editing on the Gaussian splatting representation by optimizing Gaussians' attributes to fit the rendered results to both the input edited image and randomly rendered images from different viewpoints. Here we ablate how randomly generated edited results influence the final rendered results of edited Gaussians in Fig. 9. It shows that optimizing Gaussians' attributes only from the input edited viewpoint can result in overfitting and exhibit inconsistent novel views.



(a) Input (b) Edit (c) w/o R.V. (d) w/ R.V. (e) w/o R.V. (f) w/ R.V.

Figure 9: Qualitative comparisons between texture editing results with rendered results from random viewpoints (w/ R.V.) and without (w/o R.V.). The rendered results of edited gaussians splatting from novel viewpoints can exhibit inconsistency without inputting rendered results from random viewpoints.



(a) Input (b) Diffuse Albedo (c) Edit (d) Novel View

Figure 10: Limitations. DeferredGS bakes shadow into texture on scenes containing shadow ((a)-(b)) and may result in noises after texture editing ((c)-(d)).

5 Conclusion

In this paper, we present DeferredGS, a decoupled and relightable Gaussian splatting representation, which also supports geometry and texture editing. To make this happen, we define optimizable texture attributes on Gaussians and an environment map for the physically based rendering process. To ensure proper geometry reconstruction, DeferredGS distills the normal field from previous multi-view reconstruction methods onto Gaussians' learnable normal attributes. Notably, we propose to use the *deferred rendering* technique to enable realistic relighting. However, as shown in Fig. 10 our method may produce wrong decoupled results on scenes with shadow. For future directions, it is promising to resolve the ambiguity by capturing scenes under multiple lighting conditions and introducing visibility into the rendering process like [54], [55]. On the editing side, the optimized Gaussians after texture editing may contain noise since Gaussian splatting is a global representation where a pixel is influenced by multiple Gaussians. Adaptive cloning and splitting in the edited region might produce better editing results.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020, pp. 405–421.

[2] C. Bao, B. Yang, Z. Junyi, B. Hujun, Z. Yinda, C. Zhaopeng, and Z. Guofeng, "NeuMesh: Learning disentangled neural mesh-based implicit field for geometry and texture editing," in *ECCV*, 2022, pp. 597–614.

[3] F. Xiang, Z. Xu, M. Hasan, Y. Hold-Geoffroy, K. Sunkavalli, and H. Su, "NeuTex: Neural texture mapping for volumetric neural rendering," in *CVPR*, 2021, pp. 7119–7128.

[4] S. Liu, X. Zhang, Z. Zhang, R. Zhang, J.-Y. Zhu, and B. Russell, "Editing conditional radiance fields," in *ICCV*, 2021, pp. 5773–5783.

[5] T. Wu, J. Sun, Y. Lai, and L. Gao, "DE-NeRF: Decoupled neural radiance fields for view-consistent appearance editing and high-frequency environmental relighting," in *ACM SIGGRAPH 2023 Conference Proceedings*. ACM, 2023, pp. 74:1–74:11.

[6] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, 2023.

[7] Y. Chen, Z. Chen, C. Zhang, F. Wang, X. Yang, Y. Wang, Z. Cai, L. Yang, H. Liu, and G. Lin, "GaussianEditor: Swift and controllable 3D editing with Gaussian splatting," 2023.

[8] J. Fang, J. Wang, X. Zhang, L. Xie, and Q. Tian, "GaussianEditor: Editing 3D gaussians delicately with text instructions," 2023.

[9] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *arXiv:2304.02643*, 2023.

[10] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *CVPR*, 2022, pp. 10 674–10 685.

[11] A. Haque, M. Tancik, A. Efros, A. Holynski, and A. Kanazawa, "Instruct-NeRF2NeRF: Editing 3D scenes with instructions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[12] J. Gao, C. Gu, Y. Lin, H. Zhu, X. Cao, L. Zhang, and Y. Yao, "Relightable 3D gaussian: Real-time point cloud relighting with BRDF decomposition and ray tracing," *CoRR*, vol. abs/2311.16043, 2023.

[13] Z. Liang, Q. Zhang, Y. Feng, Y. Shan, and K. Jia, "GS-IR: 3D Gaussian splatting for inverse rendering," *CoRR*, vol. abs/2311.16473, 2023.

[14] Y. Jiang, J. Tu, Y. Liu, X. Gao, X. Long, W. Wang, and Y. Ma, "GaussianShader: 3D Gaussian splatting with shading functions for reflective surfaces," *CoRR*, vol. abs/2311.17977, 2023.

[15] Y. Shi, Y. Wu, C. Wu, X. Liu, C. Zhao, H. Feng, J. Liu, L. Zhang, J. Zhang, B. Zhou, E. Ding, and J. Wang, "GIR: 3D Gaussian inverse rendering for relightable scene factorization," *Arxiv*, vol. abs/2312.05133, 2023.

[16] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3D reconstruction in function space," in *CVPR*, 2019, pp. 4460–4470.

[17] Z. Chen and H. Zhang, "Learning implicit fields for generative shape modeling," in *CVPR*, 2019, pp. 5939–5948.

[18] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *CVPR*, 2019, pp. 165–174.

[19] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, "Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision," in *CVPR*, 2020, pp. 3501–3512.

[20] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," *Advances in Neural Information Processing Systems*, pp. 15 651–15 663, 2020.

[21] C. Sun, M. Sun, and H. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," in *CVPR*, 2022, pp. 5449–5459.

[22] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "TensoRF: Tensorial radiance fields," in *ECCV*, vol. 13692, 2022, pp. 333–350.

[23] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, 2022.

[24] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi, "MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures," in *CVPR*, 2023.

[25] L. Yariv, P. Hedman, C. Reiser, D. Verbin, P. P. Srinivasan, R. Szeliski, J. T. Barron, and B. Mildenhall, "BakedSDF: Meshing neural sdfs for real-time view synthesis," in *ACM SIGGRAPH 2023 Conference Proceedings*, 2023, pp. 46:1–46:9.

[26] D. Verbin, P. Hedman, B. Mildenhall, T. E. Zickler, J. T. Barron, and P. P. Srinivasan, "Ref-NeRF: Structured view-dependent appearance for neural radiance fields," in *CVPR*, 2022, pp. 5481–5490.

[27] K. Ye, Q. Hou, and K. Zhou, "3D Gaussian splatting with deferred reflection," in *ACM SIGGRAPH 2024 Conference Proceedings*. ACM, 2024.

[28] M. Oechsle, S. Peng, and A. Geiger, "UNISURF: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction," in *ICCV*, 2021, pp. 5589–5599.

[29] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," in *Advances in Neural Information Processing Systems*, 2021, pp. 27 171–27 183.

[30] L. Yariv, J. Gu, Y. Kasten, and Y. Lipman, "Volume rendering of neural implicit surfaces," in *Advances in Neural Information Processing Systems*, 2021, pp. 4805–4815.

[31] Y. Liu, L. Wang, J. Yang, W. Chen, X. Meng, B. Yang, and L. Gao, "NeUDF: Leaning neural unsigned distance fields with volume rendering," in *CVPR*, 2023, pp. 237–247.

[32] X. Long, C. Lin, L. Liu, Y. Liu, P. Wang, C. Theobalt, T. Komura, and W. Wang, "NeuralUDF: Learning unsigned distance fields for multi-view reconstruction of surfaces with arbitrary topologies," in *CVPR*, 2023, pp. 20 834–20 843.

[33] X. Meng, W. Chen, and B. Yang, "NeAT: Learning neural implicit surfaces with arbitrary topologies from multi-view images," in *CVPR*, 2023, pp. 248–258.

[34] Y. Fan, I. Skorokhodov, O. Voynov, S. Ignatyev, E. Burnaev, P. Wonka, and Y. Wang, "Factored-NeuS: Reconstructing surfaces, illumination, and materials of possibly glossy objects," *CoRR*, vol. abs/2305.17929, 2023.

[35] W. Ge, T. Hu, H. Zhao, S. Liu, and Y.-C. Chen, "Ref-NeuS: Ambiguity-reduced neural implicit surface learning for multi-view reconstruction with reflection," *arXiv preprint arXiv:2303.10840*, 2023.

[36] H. Chen, C. Li, and G. H. Lee, "NeuSG: Neural implicit surface reconstruction with 3D Gaussian splatting guidance," *CoRR*, vol. abs/2312.00846, 2023.

[37] A. Guédon and V. Lepetit, "SuGaR: Surface-aligned Gaussian splatting for efficient 3D mesh reconstruction and high-quality mesh rendering," *CoRR*, vol. abs/2311.12775, 2023.

[38] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, "2D Gaussian splatting for geometrically accurate radiance fields," in *ACM SIGGRAPH 2024 Conference Papers, SIGGRAPH*. ACM, 2024, p. 32.

[39] P. Dai, J. Xu, W. Xie, X. Liu, H. Wang, and W. Xu, "High-quality surface reconstruction using Gaussian surfels," in *ACM SIGGRAPH 2024 Conference Papers, SIGGRAPH*. ACM, 2024, p. 22.

[40] Z. Yu, T. Sattler, and A. Geiger, "Gaussian opacity fields: Efficient adaptive surface reconstruction in unbounded scenes," *ACM Transactions on Graphics*, 2024.

[41] P. P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron, "NeRV: Neural reflectance and visibility fields for relighting and view synthesis," in *CVPR*, 2021, pp. 7495–7504.

[42] M. Boss, R. Braun, V. Jampani, J. T. Barron, C. Liu, and H. Lensch, "NeRD: Neural reflectance decomposition from image collections," in *ICCV*, 2021, pp. 12 684–12 694.

[43] K. Zhang, F. Luan, Q. Wang, K. Bala, and N. Snavely, "PhySG: Inverse rendering with spherical gaussians for physics-based material editing and relighting," in *CVPR*, 2021, pp. 5453–5462.

[44] Y. Zhang, J. Sun, X. He, H. Fu, R. Jia, and X. Zhou, "Modeling indirect illumination for inverse rendering," in *CVPR*, 2022, pp. 18 622–18 631.

[45] Z. Kuang, K. Olszewski, M. Chai, Z. Huang, P. Achlioptas, and S. Tulyakov, "NeROIC: Neural object capture and rendering from online image collections," *Computing Research Repository (CoRR)*, vol. abs/2201.02533, 2022.

[46] X. Zhang, P. P. Srinivasan, B. Deng, P. Debevec, W. T. Freeman, and J. T. Barron, "NeRFactor: Neural factorization of shape and reflectance under an unknown illumination," *ACM Trans. Graph.*, vol. 40, no. 6, pp. 1–18, 2021.

[47] J. Munkberg, W. Chen, J. Hasselgren, A. Evans, T. Shen, T. Müller, J. Gao, and S. Fidler, "Extracting triangular 3D models, materials, and lighting from images," in *CVPR*, 2022, pp. 8270–8280.

[48] J. Hasselgren, N. Hofmann, and J. Munkberg, "Shape, light, and material decomposition from images using Monte Carlo rendering and denoising," in *Advances in Neural Information Processing Systems*, 2022.

[49] T. Shen, J. Gao, K. Yin, M.-Y. Liu, and S. Fidler, "Deep marching tetrahedra: a hybrid representation for high-resolution 3D shape synthesis," in *Advances in Neural Information Processing Systems*, 2021, pp. 6087–6101.

[50] A. Mai, D. Verbin, F. Kuester, and S. Fridovich-Keil, "Neural microfacet fields for inverse rendering," 2023.

[51] H. Jin, I. Liu, P. Xu, X. Zhang, S. Han, S. Bi, X. Zhou, Z. Xu, and H. Su, "TensoIR: Tensorial inverse rendering," in *CVPR*, 2023.

[52] Y. Liu, P. Wang, C. Lin, X. Long, J. Wang, L. Liu, T. Komura, and W. Wang, "NeRO: Neural geometry and BRDF reconstruction of reflective objects from multiview images," *ACM Trans. Graph.*, vol. 42, no. 4, pp. 114:1–114:22, 2023.

[53] V. Rudnev, M. Elgharib, W. Smith, L. Liu, V. Golyanik, and C. Theobalt, "NeRF for outdoor scene relighting," in *ECCV*, 2022.

[54] Z. Wang, T. Shen, J. Gao, S. Huang, J. Munkberg, J. Hasselgren, Z. Gojcic, W. Chen, and S. Fidler, "Neural fields meet explicit geometric representations for inverse rendering of urban scenes," in *CVPR*, 2023, pp. 8370–8380.

[55] J. Sun, T. Wu, Y. Yang, Y. Lai, and L. Gao, "SOL-NeRF: Sunlight modeling for outdoor scene decomposition and relighting," in *SIGGRAPH Asia 2023 Conference Papers*. ACM, 2023, pp. 31:1–31:11.

[56] C. Xi, P. Sida, Y. Dongchen, L. Yuan, P. Bowen, L. Chengfei, and Z. Xiaowei, "IntrinsicAnything: Learning diffusion priors for inverse rendering under unknown illumination," 2024.

[57] Y. Litman, O. Patashnik, K. Deng, A. Agrawal, R. Zawar, F. D. la Torre, and S. Tulsiani, "MaterialFusion: Enhancing inverse rendering with material diffusion priors," 2024.

[58] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall, "DreamFusion: Text-to-3D using 2D diffusion," *arXiv*, 2022.

[59] C. Zeng, Y. Dong, P. Peers, Y. Kong, H. Wu, and X. Tong, "DiLightNet: Fine-grained lighting control for diffusion-based image generation," in *ACM SIGGRAPH 2024 Conference Papers*, 2024.

[60] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," 2023.

[61] ——, "Scaling in-the-wild training for diffusion-based illumination harmonization and editing by imposing consistent light transport," in *International Conference on Learning Representations*, 2024.

[62] H. Jin, Y. Li, F. Luan, Y. Xiangli, S. Bi, K. Zhang, Z. Xu, J. Sun, and N. Snavely, "Neural gaffer: Relighting any object via diffusion," *CoRR*, vol. abs/2406.07520, 2024.

[63] X. Zhao, P. P. Srinivasan, D. Verbin, K. Park, R. M. Brualla, and P. Henzler, "IllumiNeRF: 3D Relighting without Inverse Rendering," in *NeruIPS*, 2024.

[64] K. Deng, T. Omernick, A. Weiss, D. Ramanan, J.-Y. Zhu, T. Zhou, and M. Agrawala, "Flashtex: Fast relightable mesh texturing with lightcontrolnet," in *European Conference on Computer Vision (ECCV)*, 2024.

[65] L. Yariv, Y. Kasten, D. Moran, M. Galun, M. Atzmon, R. Basri, and Y. Lipman, "Multiview neural surface reconstruction by disentangling geometry and appearance," in *Advances in Neural Information Processing Systems*, 2020.

[66] M. G. Crandall and P.-L. Lions, "Viscosity solutions of hamilton-jacobi equations," *Transactions of the American mathematical society*, vol. 277, no. 1, pp. 1–42, 1983.

[67] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986, pp. 143–150.

[68] B. Burley and W. D. A. Studios, "Physically-based shading at Disney," in *ACM SIGGRAPH*, 2012, pp. 1–7.

[69] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, "Microfacet models for refraction through rough surfaces," in *Eurographics conference on Rendering Techniques*, 2007, pp. 195–206.

[70] B. Karis, "Real shading in Unreal engine 4," 2013.

[71] O. Sorkine-Hornung and M. Alexa, "As-rigid-as-possible surface modeling," in *Symposium on Geometry Processing*, 2007.

[72] B. O. Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, 2018. [Online]. Available: http://www.blender.org

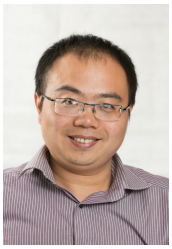
[73] Z. Kuang, Y. Zhang, H. Yu, S. Agarwala, S. Wu, and J. Wu, "Stanford-ORB: A real-world 3d object inverse rendering benchmark," in *NeurIPS 2023*, 2023.

[74] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[75] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018, pp. 586–595.

Tong Wu received his bachelor's degree in computer science from Huazhong University of Science and Technology in 2019. He is currently a PhD candidate at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include computer graphics and computer vision.

Jia-Mu Sun is a Master student of Computer Science at Institute of Computing Technology, Chinese Academy of Sciences. He received his bachelor's degree from Huazhong University of Science and Technology. His research interests include geometry learning and rendering.



Yu-Kun Lai received his bachelor's degree and PhD degree in computer science from Tsinghua University in 2003 and 2008, respectively. He is currently a Professor in the School of Computer Science & Informatics, Cardiff University. His research interests include computer graphics, geometry processing, image processing and computer vision. He is on the editorial boards of *IEEE Transactions on Visualization and Computer Graphics* and *The Visual Computer*.

Yuewen Ma received his PhD degree from Nanyang Technological University, Singapore, in 2013. He has been engaged in the research and product of computer graphics and 3D vision for a long time. He is currently the leader of 3D reconstruction at ByteDance Pico.

Leif Kobbelt received his master's and PhD degrees from the University of Karlsruhe, Germany, in 1992 and 1994, respectively. He is a full professor and the head of the Computer Graphics Group with the RWTH Aachen University, Germany. His research interests include all areas of computer graphics and geometry processing with a focus on multi-resolution and free-form modeling as well as the efficient handling of polygonal mesh data.

Lin Gao received his bachelor's degree in mathematics from Sichuan University and his PhD degree in computer science from Tsinghua University. He is currently an Associate Professor at the Institute of Computing Technology, Chinese Academy of Sciences. He has been awarded the Newton Advanced Fellowship from the Royal Society and the Asia Graphics Association Young Researcher Award. His research interests include computer graphics and geometric processing.