Extracting Triangular 3D Models, Materials, and Lighting From Images

Jacob Munkberg[1] Jon Hasselgren[1] Tianchang Shen[1,2,3] Jun Gao[1,2,3]
Wenzheng Chen[1,2,3] Alex Evans[1] Thomas Müller[1] Sanja Fidler[1,2,3]

[1]NVIDIA [2]University of Toronto [3]Vector Institute

Abstract

We present an efficient method for joint optimization of topology, materials and lighting from multi-view image observations. Unlike recent multi-view reconstruction approaches, which typically produce entangled 3D representations encoded in neural networks, we output triangle meshes with spatially-varying materials and environment lighting that can be deployed in any traditional graphics engine unmodified. We leverage recent work in differentiable rendering, coordinate-based networks to compactly represent volumetric texturing, alongside differentiable marching tetrahedrons to enable gradient-based optimization directly on the surface mesh. Finally, we introduce a differentiable formulation of the split sum approximation of environment lighting to efficiently recover all-frequency lighting. Experiments show our extracted models used in advanced scene editing, material decomposition, and high quality view interpolation, all running at interactive rates in triangle-based renderers (rasterizers and path tracers).

1. Introduction

3D content creation is a challenging, mostly manual task which requires both artistic modeling skills and technical knowledge. Efforts to automate 3D modeling can save substantial production costs or allow for faster and more diverse content creation. Photogrammetry [54,62] is a popular technique to assist in this process, where multiple photos of an object are converted into a 3D model. Game studios leverage photogrammetry to quickly build highly detailed virtual landscapes [24]. However, this is a multi-stage process, including multi-view stereo [58] to align cameras and find correspondences, geometric simplification, texture parameterization, material baking and delighting. This complex pipeline has many steps with conflicting optimization goals and errors that propagate between stages. Artists rely on a plethora of software tools and significant manual adjustments to reach the desired quality of the final 3D model.

Project page: https://nvlabs.github.io/nvdiffrec/

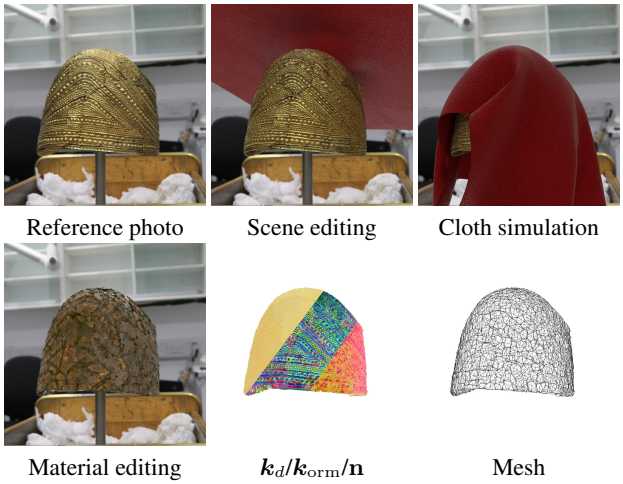


Figure 1. We reconstruct a triangular mesh with unknown topology, spatially-varying materials, and lighting from a set of multi-view images. We show examples of scene manipulation using off-the-shelf modeling tools, enabled by our reconstructed 3D model.

Our goal is to frame this process as an *inverse rendering* task, and optimize as many steps as possible jointly, driven by the quality of the rendered images of the reconstructed model, compared to the captured input imagery. Recent work approaches 3D reconstruction with neural rendering, and provides high quality novel view synthesis [45]. However, these methods typically produce representations that entangle geometry, materials and lighting into neural networks, and thus cannot easily support scene editing operations. Furthermore, to use them in traditional graphics engines, one needs to extract geometry from the network using methods like Marching Cubes which may lead to poor surface quality, particularly at low triangle counts. Recent neural methods can disentangle shape, materials, and lighting [5, 77, 79], but sacrifice reconstruction quality. Also, the materials encoded in neural networks cannot easily be edited or extracted in a form compatible with traditional game engines. In contrast, we reconstruct 3D content compatible with traditional graphics engines, supporting relighting and scene editing.

In this paper, we present a highly efficient inverse render-

ing method capable of extracting triangular meshes of unknown topology, with spatially-varying materials and lighting from multi-view images. We assume that the object is illuminated under one unknown environment lighting condition, and that we have corresponding camera poses and masks indicating the object in these images, as in past work [5]. Our approach learns topology and vertex positions for a surface mesh without requiring any initial guess for the 3D geometry. The heart of our method is a differentiable surface model based on a deformable tetrahedral mesh [60], which we extend to support spatially-varying materials and high dynamic range (HDR) environment lighting, through a novel differentiable split sum approximation. We optimize geometry, materials and lighting (50M+ parameters) jointly using a highly optimized differentiable rasterizer with deferred shading [25, 36]. The resulting 3D model can be deployed without conversion on any device supporting triangle rendering, including phones and web browsers, and renders at interactive rates.

Experiments show our extracted models used in scene editing (e.g., Figure 1), material decomposition, and high quality view interpolation, all running at interactive rates in triangle-based renderers (rasterizers and path tracers).

2. Related Work

2.1. Multi-view 3D Reconstruction

Classical methods for multi-view 3D reconstruction either exploit inter-image correspondences [2, 14, 15, 58] to estimate depth maps or use voxel grids to represent shapes [12, 59]. The former methods typically fuse depth maps into point clouds, optionally generating meshes [32]. They rely heavily on the quality of matching, and errors are hard to rectify during post-processing. The latter methods estimate occupancy and color for each voxel and are often limited by the cubic memory requirement.

Neural implicit representations leverage differentiable rendering to reconstruct 3D geometry with appearance from image collections [29, 45, 49]. NeRF [45] and follow-ups [18, 44, 47, 48, 55, 56, 68, 70, 75, 78], use volumetric representations and compute radiance by ray marching through a neurally encoded 5D light field. While achieving impressive results on novel view synthesis, geometric quality suffers from the ambiguity of volume rendering [78]. Surface-based rendering methods [49, 73] use implicit differentiation to obtain gradients, optimizing the underlying surface directly. Unisurf [53] is a hybrid approach that gradually reduces the sampling region, encouraging a volumetric representation to converge to a surface, and NeuS [67] provides an unbiased conversion from SDF into density for volume rendering. Common for all methods is that they rely on ray marching for rendering, which is computationally expensive both during training and inference. While implicit surfaces can be converted to meshes for fast inference, this introduces additional error not accounted for during optimization [60]. We optimize the end-to-end image loss of an explicit mesh representation, supporting intrinsic decomposition of shape, materials and lighting by design, and utilizing efficient differentiable rasterization [36].

Explicit surface representations are proposed to estimate explicit 3D mesh from images [9, 10, 17, 27, 40, 41, 60]. Most approaches assume a given, fixed mesh topology [9, 10, 27, 41], but this is improved in recent work [17, 40, 60]. In particular, DMTet [60] directly optimizes the surface mesh using a differentiable marching tetrahedral layer. However, it focuses on training with 3D supervision. In this work, we extend DMTet to 2D supervision, using differentiable rendering to jointly optimize topology, materials, and lighting.

2.2. BRDF and Lighting Estimation

Beyond geometry, several techniques estimate surface radiometric properties from images. Previous work on BTF and SVBRDF estimation rely on special viewing configurations, lighting patterns or complex capturing setups [4, 7, 19–21, 23, 37, 57, 69]. Recent methods exploit neural networks to predict BRDF from images [16, 22, 38, 39, 43, 50]. Differentiable rendering methods [9, 10, 25, 41, 80] learn to predict geometry, SVBRDF and, in some cases, lighting via 2D image loss. Still, their shape is generally deformed from a sphere and cannot represent arbitrary topology.

Neural implicit representations successfully estimate lighting and BRDF from image collections. Bi et al. [4] and NeRV [64] model light transport to support advanced lighting effects, e.g., shadows, but have very high computational cost. Most related to our work are neural 3D reconstruction methods for jointly estimating shape, BRDFs and lighting from images [5, 6, 77, 79], while providing an *intrinsic decomposition* of these terms. Illumination is represented using mixtures of spherical Gaussians (NeRD [5], PhySG [77]), or low resolution envmaps (NeRFactor [79]), in both cases limited to low frequency illumination. In contrast, we propose a differentiable split sum lighting model, also adopted by the concurrent work Neural-PIL [6]. These neural implicit methods use multiple MLPs to factorize

Method	Geometry	Factorize	Training	Inference
NeRF [45]	NV		day	seconds
NGP-NeRF [47]	NV		minutes	ms
NeRD [5]	NV	✓	days	seconds
NerFactor [79]	NV	✓	days	seconds
PhySG [77]	NS	✓	day	seconds
NeuS [67]	NS		day	seconds
Our	Mesh	✓	hour	ms

Table 1. Taxonomy of methods. NV: Neural volume, NS: Neural surface. Factorize indicates if the method supports some decomposition of geometry, materials, and lighting.

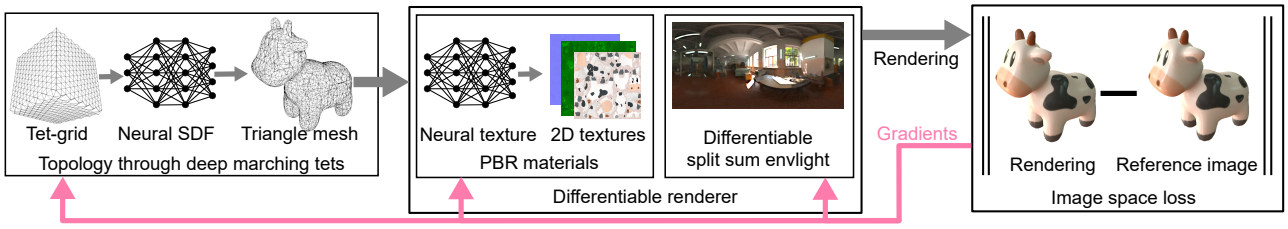


Figure 2. **Overview of our approach.** We learn topology, materials, and environment map lighting jointly from 2D supervision. We leverage differentiable marching tetrahedrons to directly optimize topology of a triangle mesh. While the topology is drastically changing, we learn materials through volumetric texturing, efficiently encoded using an MLP with positional encoding. Finally, we introduce a differentiable version of the split sum approximation for environment lighting. Our output representation is a triangle mesh with spatially varying 2D textures and a high dynamic range environment map, which can be used unmodified in standard game engines. The system is trained end-to-end, supervised by loss in image space, with gradient-based optimization of all stages. Spot model by Keenan Crane.

the representation, resulting in long training and inference times. Furthermore, these methods forgo the vast ecosystem of available 3D modeling and rendering tools, "reinventing the wheel" for tasks such as rendering, scene editing [72] and simulation. In contrast, our output is directly compatible with existing renderers and modeling tools. We optimize an explicit surface mesh, BRDF parameters, and lighting stored in an HDR probe, achieving faster training speed and better decomposition results. Table 1 shows a high level comparison of the methods.

3. Our Approach

We present a method for 3D reconstruction supervised by multi-view images of an object illuminated under one unknown environment lighting condition, together with known camera poses and background segmentation masks. The target representation consists of triangle meshes, spatially-varying materials (stored in 2D textures) and lighting (a high dynamic range environment probe). We carefully design the optimization task to *explicitly* render triangle meshes, while robustly handling arbitrary topology. Hence, unlike most recent work using neural implicit surface or volumetric representations, we directly optimize the target shape representation.

Concretely, we adapt Deep Marching Tetrahedra [60] (DMTet) to work in the setting of 2D supervision, and jointly optimize shape, materials, and lighting. At each optimization step, the shape representation – parameters of a signed distance field (SDF) defined on a grid with corresponding per-vertex offsets – is converted to a triangular surface mesh using a marching tetrahedra layer. Next, we render the extracted surface mesh in a differentiable rasterizer with deferred shading, and compute loss in image space on the rendered image compared to a reference image. Finally, the loss gradients are back-propagated to update the shape, textures and lighting parameters. Our approach is summarized in Figure 2 and each step is described in detail below; Section 3.1 outlines our topology optimization, Section 3.2 presents the spatially-varying shading model, and our approach for reconstructing all-frequency environment lighting is described in Section 3.3.

Optimization task Let ϕ denote our optimization parameters (i.e., SDF values and vertex offsets representing the shape, spatially varying material and light probe parameters). For a given camera pose, c, the differentiable renderer produces an image $I_\phi(c)$. The reference image $I_{\text{ref}}(c)$ is a view from the same camera. Given a loss function L, we minimize the empirical risk

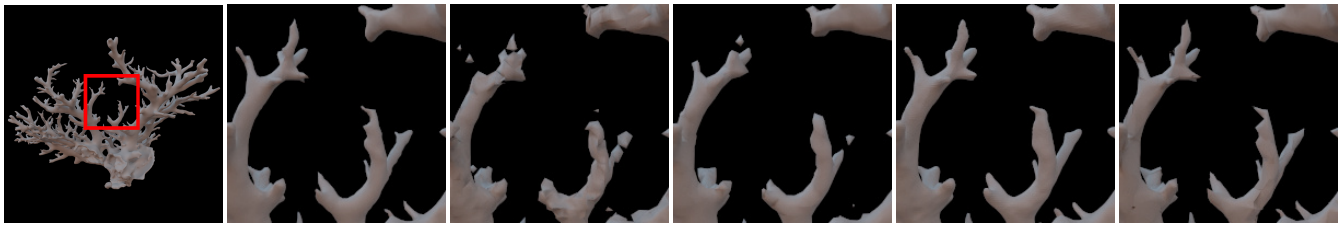
$$\underset{\phi}{\operatorname{argmin}}\, \mathbb{E}_c\big[L\big(I_\phi(c), I_{\text{ref}}(c)\big)\big] \tag{1}$$

using Adam [33] based on gradients w.r.t. the optimization parameters, $\partial L/\partial\phi$, which are obtained through differentiable rendering. Our renderer uses physically based shading and produces images with high dynamic range. Therefore, the objective function must be robust to the full range of floating-point values. Following recent work in differentiable rendering [25], our loss function is $L = L_{\text{image}} + L_{\text{mask}} + \lambda L_{\text{reg}}$, an image space loss, L_{image} (L_1 norm on tone mapped colors), a mask loss, L_{mask} (squared L_2) and a regularizer L_{reg} (Equation 11) to improve geometry. Please refer to the supplemental material for details.

Assumptions For performance reasons we use a differentiable rasterizer with deferred shading [25], hence reflections, refractions (e.g., glass), and translucency are not supported. During optimization, we only renderer direct lighting without shadows. Our shading model uses a diffuse Lambertian lobe and a specular, isotropic microfacet GGX lobe, which is commonly used in modern game engines [31, 35]. Both dielectrics and metal materials are supported. We note that our approach directly generalizes to a differentiable path tracer [51, 52], but at a significantly increased computational cost.

3.1. Learning Topology

Volumetric and implicit shape representations (e.g., SDFs) can be converted to meshes through Marching



	Reference	NeRF (57k tris)	NeuS (53k tris)	NeuS (900k tris)	Our (53k tris)
Chamfer L_1 $\times 10^{-4}$		33.4	9.19	5.84	4.65

Figure 3. Triangle mesh extraction from a set of 256 rendered images w/ masks. Damicornis model from the Smithsonian 3D repository [61], We extracted meshes from NeRF and NeuS using Marching Cubes for a target triangle count of roughly 50k triangles and optimized the example in our pipeline for a similar count. We show renderings of the extracted meshes in a path tracer and report the Chamfer loss. We note that NeuS, which optimizes a surface representation, significantly improves on the volumetric representation used by NeRF for this example. Furthermore, our end-to-end optimization of a triangle mesh improves both the visual quality and the Chamfer loss at a fixed triangle count. When drastically increasing the triangle count in the NeuS mesh extraction (from 53k to 900k triangles), the quality improves significantly, indicating that NeuS has a high quality internal surface representation. Still, our mesh with 53k triangles is on par with the high resolution NeuS output, indicating the benefit of directly optimizing the mesh representation.

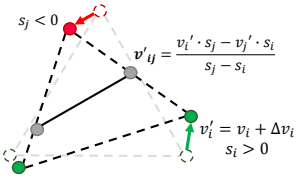


Figure 4. Marching Tetrahedra extracts faces from a tetrahedral grid with grid vertices, $v_i' = v_i + \Delta v_i$ and scalar SDF values, s_i. For tets with $\text{sign}(s_i) \neq \text{sign}(s_j)$, faces are extracted, and the face vertices, v_{ij}, are determined by by linear interpolation.

Cubes [42] (MC) in a post-processing step. However, MC inevitably imposes discretization errors. As a result, the output mesh quality, particularly at the moderate triangle counts typically used in real-time rendering, is often not sufficient. Similarly, simplifying dense extracted meshes using decimation tools may introduce errors in rendered appearance. To avoid these issues, we explicitly render triangle meshes during optimization. We leverage Deep Marching Tetrahedra [60] (DMTet) in a 2D supervision setting through differentiable rendering. DMTet is a hybrid 3D representation that represents a shape with a discrete SDF defined on vertices of a deformable tetrahedral grid. The SDF is converted to triangular mesh using a differentiable marching tetrahedra layer (MT), as illustrated in Figure 4. The loss, in our case computed on renderings of the 3D model, is back-propagated to the implicit field to update the surface topology. This allows us to directly optimize the surface mesh and rendered appearance *end-to-end*.

We illustrate the advantage of end-to-end learning in Figure 3, where we compare our meshes to those generated by competing methods. While NeRF [45] (volumetric rep.) and NeuS [67] (implicit surface rep.) provide high quality view interpolation, the quality loss introduced in the MC step is significant at low triangle counts.

Given a tetrahedral grid with vertex positions v, DMTet learns SDF values, s, and deformation vectors Δv. The SDF values and deformations can either be stored explicitly as values per grid vertex, or implicitly [49, 53] by a neural network. At each optimization step, the SDF is first converted to a triangular surface mesh using MT, which is shown to be differentiable w.r.t. SDF and can change surface topology in DMTet [60]. Next, the extracted mesh is rendered using a differentiable rasterizer to produce an output image, and image-space loss gradients are back-propagated to the SDF values and offsets (or network weights). A neural SDF representation can act as a smoothness prior, which can be beneficial in producing well-formed shapes. Directly optimizing per-vertex attributes, on the other hand, can capture higher frequency detail and is faster to train. In practice, the optimal choice of parametrization depends on the ambiguity of geometry in multi-view images. We provide detailed analysis in the supplemental materials.

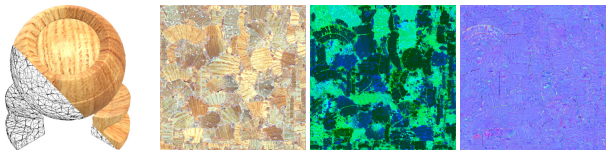
To reduce floaters and internal geometry, we regularize the SDF values of DMTet similar to Liao et al. [40]. Given the binary cross-entropy H, sigmoid function σ, and the sign function $\text{sign}(x)$, we define the regularizer as

$$L_{\text{reg}} = \sum_{i,j \in \mathbb{S}_e} H\left(\sigma(s_i), \text{sign}\left(s_j\right)\right) + H\left(\sigma(s_j), \text{sign}\left(s_i\right)\right), \quad (2)$$

where we sum over the set of unique edges, $\mathbb{S}_e$, in the tetrahedral grid, for which $\text{sign}(s_i) \neq \text{sign}(s_j)$. Intuitively, this reduces the number of sign flips and simplifies the surface, penalizing internal geometry or floaters. We ablate the choice of regularization loss in the supplemental material.

3.2. Shading Model

Material Model We follow previous work in differentiable rendering [25] and use the physically-based (PBR) material model from Disney [8]. This lets us easily import game assets and render our optimized models directly in existing engines without modifications. This material model combines a diffuse term with an isotropic, specular GGX lobe [66]. Referring to Figure 5, the diffuse lobe param-



wire/shaded $\boldsymbol{k}_d$ $\boldsymbol{k}_{\text{orm}}$ normals

Figure 5. We represent 3D models as a triangular mesh and a set of spatially varying materials following a standard PBR model.

eters $\boldsymbol{k}_d$ are provided as a four-component texture, where the optional fourth channel α represents transparency. The specular lobe is described by a roughness value, r, for the GGX normal distribution function and a metalness factor, m, which interpolates between plastic and metallic appearance by computing a specular highlight color according to $\boldsymbol{k}_s = (1-m)\cdot 0.04 + m\cdot \boldsymbol{k}_d$ [31]. Following a standard convention, we store these values in a texture $\boldsymbol{k}_{\text{orm}} = (o, r, m)$, where o is left unused. Finally, we include a tangent space normal map, $\mathbf{n}$, to capture high frequency shading detail. We regularize material parameters using a smoothness loss [79], please refer to our supplemental material for details.

Texturing Automatic texture parametrization for surface meshes is an active research area in computer graphics. We optimize topology, which requires continually updating the parametrization, potentially introducing discontinuities into the training process. To robustly handle texturing during topology optimization, we leverage volumetric texturing, and use world space position to index into our texture. This ensures that the mapping varies smoothly with both vertex translations and changing topology.

The memory footprint of volumetric textures grows cubically, which is unmanageable for our target resolution. We therefore extend the approach of PhySG [77], using a multilayer perceptron (MLP) to encode all material parameters in a compact representation. This representation can adaptively allocate detail near the 2D manifold representing the surface mesh, which is a small subset of the dense 3D volume. More formally, we let a positional encoding + MLP represent a mapping $\mathbf{x} \rightarrow (\boldsymbol{k}_d, \boldsymbol{k}_{\text{orm}}, \mathbf{n})$, e.g., given a world space position $\mathbf{x}$, compute the base color, $\boldsymbol{k}_d$, specular parameters, $\boldsymbol{k}_{\text{orm}}$ (roughness, metalness), and a tangent space normal perturbation, $\mathbf{n}$. We leverage the tiny-cuda-nn framework [46], which provides efficient kernels for hashgrid positional encoding [47] and MLP evaluations.

Once the topology and MLP texture representation have converged, we *re-parametrize* the model: we generate unique texture coordinates using xatlas [74] and sample the MLP on the surface mesh to initialize 2D textures, then continue the optimization with fixed topology. Referring to Figure 6, this effectively removes texture seams introduced by the (u, v)-parametrization, and may also increase texture detail as we can use high resolution 2D textures for each of

Figure 6. Sampling out the volumetric representation to create 2D textures results in texture seams (left). However, further optimization (right), quickly removes the seams automatically.

$\boldsymbol{k}_d$, $\boldsymbol{k}_{\text{orm}}$, and $\mathbf{n}$. This results in 2D textures compatible with standard 3D tools and game engines.

3.3. Image Based Lighting

We adopt an image based lighting model, where the scene environment light is given by a high-resolution cube map. Following the rendering equation [30], we compute the outgoing radiance $L(\omega_o)$ in direction ω_o by:

$$L(\omega_o) = \int_{\Omega} L_i(\omega_i) f(\omega_i, \omega_o)(\omega_i \cdot \mathbf{n}) d\omega_i. \tag{3}$$

This is an integral of the product of the incident radiance, $L_i(\omega_i)$ from direction ω_i and the BSDF $f(\omega_i, \omega_o)$. The integration domain is the hemisphere Ω around the surface intersection normal, $\mathbf{n}$.

Below, we focus on the specular part of the outgoing radiance, where, in our case, the BSDF is a Cook-Torrance microfacet specular shading model [11] according to:

$$f(\omega_i, \omega_o) = \frac{D\,G\,F}{4(\omega_o \cdot \mathbf{n})(\omega_i \cdot \mathbf{n})}, \tag{4}$$

where D, G and F are functions representing the GGX [66] normal distribution (NDF), geometric attenuation and Fresnel term, respectively.

High quality estimates of image based lighting can be obtained by Monte Carlo integration. For low noise levels, large sample counts are required, which is typically too expensive for interactive applications. Thus, spherical Gaussians (SG) and spherical harmonics (SH) are common approximations for image based lighting [5, 9, 77]. They allow for control over the lighting frequency through varying the number of SG lobes (or SH coefficients), and are efficient representations for low to medium frequency lighting. However, representing high frequency and highly specular materials requires many SG lobes, which comes at a high runtime cost and hurts training stability.

We instead draw inspiration from real-time rendering, where the *split sum* approximation [31] is a popular, efficient method for all-frequency image based lighting. Here, the lighting integral from Equation 3 is approximated as:

$$L(\omega_o) \approx \int_{\Omega} f(\omega_i, \omega_o)(\omega_i \cdot \mathbf{n}) d\omega_i \int_{\Omega} L_i(\omega_i) D(\omega_i, \omega_o)(\omega_i \cdot \mathbf{n}) d\omega_i. \tag{5}$$

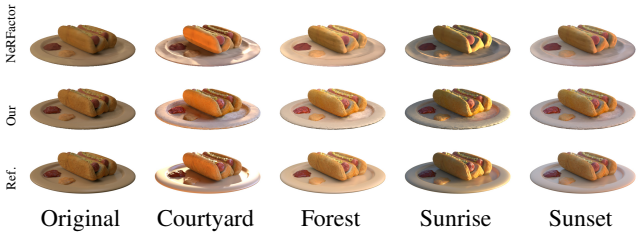


Figure 7. Relighting quality for a scene from the NeRFactor dataset, with our examples relit using Blender, and NeRFactor results generated using the public code.

The first term represents the integral of the specular BSDF with a solid white environment light. It only depends on the parameters $\cos\theta = \omega_i \cdot \mathbf{n}$ and the roughness r of the BSDF, and can be precomputed and stored in a 2D lookup texture. The second term represents the integral of the incoming radiance with the specular NDF, D. Following Karis [31], this term is also pre-integrated and represented by a filtered cubemap. In each mip-level, the environment map is integrated against D for a fixed roughness value (increased roughness at coarser mips).

The split sum approach is popular for its modest runtime cost, using only two texture lookups: query the 2D lookup texture representing the first term based on $(r, \cos\theta)$ and the mip pyramid at level r, in direction ω_o. This should be compared to evaluating SG products with hundreds of lobes for each shading point. Furthermore, it uses the standard GGX parametrization, which means that we can relight our extracted models with different kinds of light sources (point, area lights etc.) and use our reconstructed materials with no modifications in most game engines and modeling tools.

We introduce a *differentiable* version of the split sum shading model to learn environment lighting from image observations through differentiable rendering. We let the texels of a cube map (typical resolution $6 \times 512 \times 512$) be the trainable parameters. The base level represents the pre-integrated lighting for the lowest supported roughness value, and each smaller mip-level is constructed from the base level using the pre-filtering approach from Karis [31].

To obtain texel gradients, we express the lighting computations using PyTorch's auto-differentiation. However, the pre-filtering of the second term in Equation 5 must be updated in each training iteration, and therefore warrant a specialized CUDA implementation to reduce the training cost. This term can either be estimated through Monte-Carlo integration (BSDF importance sampling), or by pre-filtering the environment map in a solid-angle footprint derived from the NDF. To reduce noise, at the cost of introducing some bias, we chose the latter approach. Please refer to our supplemental material for implementation details.

We additionally create a single filtered low-resolution ($6 \times 16 \times 16$) cube map representing the *diffuse* lighting. The process is identical to the filtered specular probe, sharing the same trainable parameters, average-pooled to the mip level with roughness $r = 1$. The pre-filtering of the diffuse term only uses the cosine term, $\omega_i \cdot \mathbf{n}$. The two filtering steps are fully differentiable and are performed at each training step.

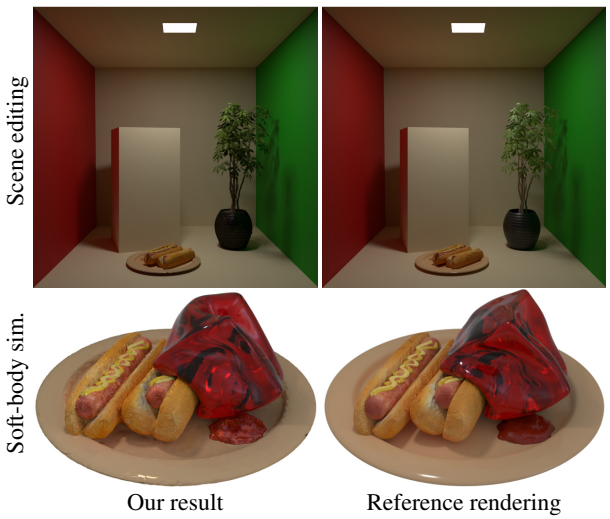


Figure 8. To highlight the benefits of our explicit representation, we insert two reconstructed models into the Cornell box. Note that the objects accurately interact with the scene lighting, and cast shadows (e.g., the green wall). Next, we use our reconstructed hotdog model in a soft-body physics simulation, dropping red jelly on the plate. We run the entire simulation (21 frames) on both the reference 3D mesh and our reconstructed mesh, and display the last frame. Note that these applications are not currently feasible for neural volumetric representations.

4. Experiments

In the following, we evaluate our system for a variety of applications. To emphasize our approach's explicit decomposition into a triangle mesh and materials, we show re-lighting, editing, and simulation using off-the shelf tools. We also compare to recent neural methods supporting factorization: NeRD [5] and NeRFactor [79]. While not our main focus, we include view interpolation results to establish a baseline comparison to state-of-the-art methods. Finally, we compare our split-sum approximation against spherical Gaussians for image-based lighting.

4.1. Scene Editing and Simulation

Our factorized scene representation enables advanced scene editing. Previous work using density-based neural representations only supports rudimentary relighting and simple forms of scene edits [5, 77, 79].

In Figure 7 we compare the relighting quality of our reconstructed model, rendered using the Blender Cycles path tracer, with the results of NeRFactor (rendered by evaluat-

	Relighting			k_d		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
NeRFactor	23.78	0.907	0.112	23.11	0.917	0.094
Our	24.53	0.914	0.085	24.75	0.924	0.092

Table 2. Relighting quality for NeRFactor's synthetic dataset. The reported image metrics are the arithmetic mean over 8 validation views and 8 light probes for all 4 test scenes. We also show metrics for the k_d (albedo) textures. Following NeRFactor, we note that the scale factor between material and lighting is indeterminable and therefore normalize albedo images by the average intensity of the reference before measuring errors.

ing a neural network). A quantitative summary is provided in Table 2, where we also measure the quality of the reconstructed albedo textures. We note that our method produces more detailed results and outperforms NeRFactor in all metrics. Our artifacts come mainly from the mismatch between training (using a rasterizer), and inference (using full global illumination). In areas with strong shadowing or color bleeding, our material and geometry quality suffer. A differentiable path tracer [52] can likely improve the material separation in our pipeline, but would require significantly more computations.

Our representation can be directly deployed in the vast collection of 3D content generation tools available for triangle meshes. This greatly facilitates scene editing, which is still very challenging for neural volumetric representations [79]. We show advanced scene editing examples in Figure 8, where we add our reconstructed models from the NeRFactor dataset to the Cornell box and use them in a soft-body simulation. Note that our models receive scene illumination, cast accurate shadows, and robustly act as colliders for virtual objects. In Figure 1, and the supplemental video, we show another example, where an object is reconstructed from real-world photographs and then used as a collider for a virtual cloth object. The combined scene is then rendered using our extracted environment light. Note that shading of the virtual object looks plausible given the reference photo. We also show material editing on the same example.

4.2. View interpolation

Synthetic datasets We show results for the NeRF *realistic synthetic image* dataset in Table 3 and a visual example of the MATERIALS scene in Figure 9. Per-scene results and visual examples are included in our supplemental material, where we also include Chamfer loss on extracted meshes. Our method consistently performs on par with NeRF, with better quality in some scenes. The margins are smaller for perceptually based image metrics (SSIM and LPIPS). We speculate that density-based volume approaches can more efficiently minimize PSNR than our opaque meshes. However, the effect of slightly moving a silhouette edge will not be as detrimental to a perceptual metric.

The DRUMS and SHIP scenes are failure cases for our method. We assume mostly direct lighting, with no significant global illumination effects, and these scenes contain both significant intra-scene reflections, refractions, and caustics. Interestingly, while material reconstruction suffers, we still note high quality results for view interpolation.

Given that we factorize into explicit shape, materials and lighting, we have slightly lower quality on novel view synthesis than methods specialized for view-interpolation. To put this in context, in Table 4 we compare our aproach against NeRFactor, which performs a similar factorization, and our approach. We observe a 4.21 dB PSNR image quality reduction for NeRFactor compared to the NeRF baseline. This is consistent with NeRD [5] which do not provide source code but report a 4.17 dB quality drop for their factorized representation on a subset of the NeRF synthetic dataset. In contrast, our quality is significantly higher, while still providing the flexibility of a factorized representation.

Real-world datasets NeRD [5] provides a small dataset of real-world photo scans with auto-generated (inaccurate) coverage masks and diverse camera placement.

Method	PSNR↑	SSIM↑	LPIPS↓
PhySG	18.91	0.847	0.182
NeRF	31.00	0.947	0.081
Mip-NeRF	33.05	0.961	0.067
Our	29.05	0.939	0.081

Table 3. Average results for the eight scenes in the NeRF realistic synthetic dataset. Each scene consists of 100 training images, and 200 test images, with masks and known camera poses. Results from NeRF are taken from Table 4 of the NeRF paper [45]. PhySG and Mip-NeRF were retrained using public source code.

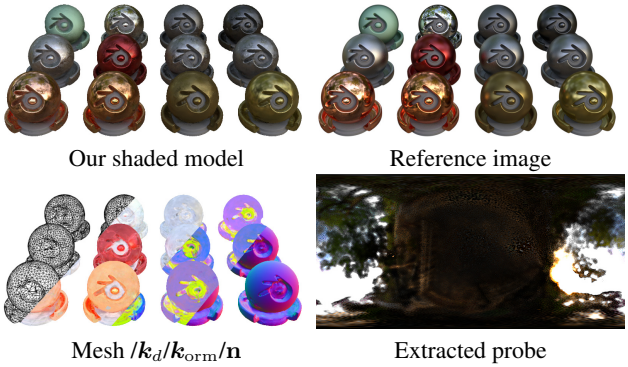


Figure 9. Our result on the MATERIALS scene, reconstructed from 100 images from the NeRF synthetic dataset.

Method	PSNR↑	SSIM↑	LPIPS↓
NeRF	31.08	0.956	0.064
NeRFactor	26.87	0.930	0.099
Our	31.65	0.967	0.054

Table 4. View interpolation error metrics on NeRFactor's variant of the NeRF synthetic dataset. The reported image metrics are the arithmetic mean over the eight validation images of all four scenes.

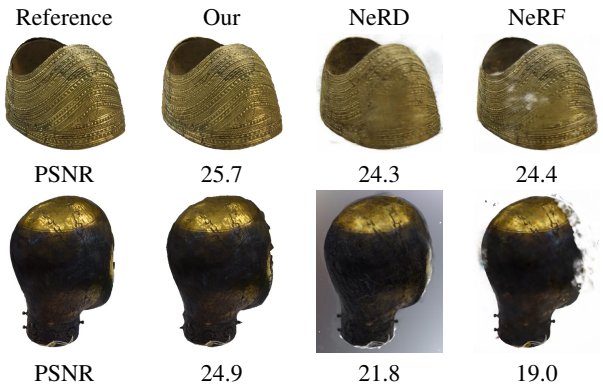


Figure 10. Reconstruction from photographs (datasets from NeRD), comparing our results with NeRD and NeRF. Images in the two rightmost colums were provided by the NeRD authors. We score higher in terms of image metrics, most likely due to our mesh representation enforcing opaque geometry, where competing algorithms rely on volumetric opacity. Despite inconsistencies in camera poses and masks, our results remain sharp while NeRF and NeRD suffer from floating or missing geometry.

Visual and quantitative results are shown in Figure 10, where we have masked out the background for the reference object. Due to inconsistencies in the dataset, both NeRF and NeRD struggle to find sharp geometry borders with transparent "floaters" and holes. In contrast, we get sharp silhouettes and a significant boost in image quality. The results reported for NeRD are for their volumetric representation. Note that NeRD can generate output meshes as a post-processing step, but at a significant quality loss (visual comparison included in our supplemental material).

4.3. Comparing Spherical Gaussians and Split Sum

In Figure 11, we compare our differentiable split sum environment lighting approximation, from Section 3.3 against the commonly used spherical Gaussian (SG) model. Split sum captures the lighting much more faithfully across all frequencies, while still having a lower runtime cost. In our implementation, we observe a $5\times$ reduction of optimization time compared to SG with 128 lobes. At inference, evaluating the split sum approximation is extremely fast, requiring just two texture lookups.

5. Limitations and Conclusions

Our main limitation is the simplified shading model, not accounting for global illumination or shadows. This choice is intentional to accelerate optimization, but is a limiting factor for material extraction and relighting. With the current progress in differentiable path tracing, we look forward to this limitation being lifted in future work. We additionally rely on alpha masks to separate foreground from background. While our method seems quite robust to corrupted masks, it would be beneficial to further incorporate this step

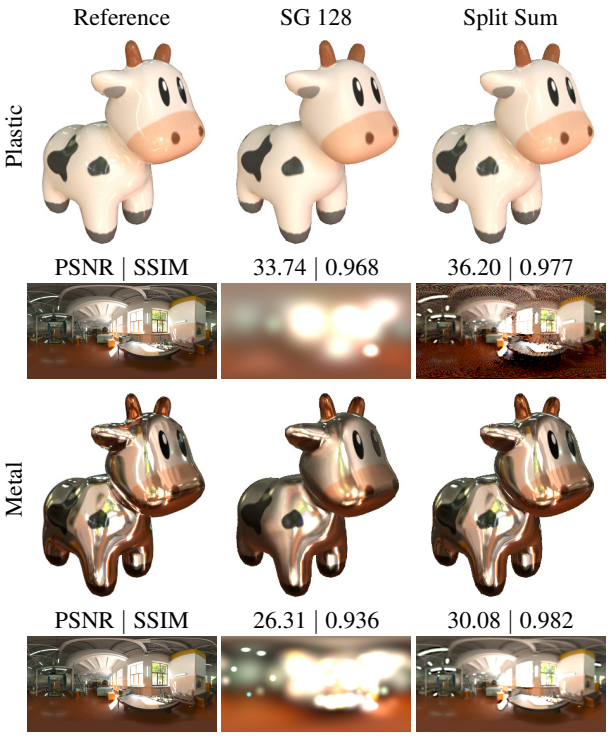


Figure 11. Environment lighting approximated with Spherical Gaussians using 128 lobes vs. Split Sum. The training set consists of 256 path traced images with Monte Carlo sampled environment lighting using a high resolution HDR probe. We assume known geometry and optimize materials and lighting using identical settings for both methods. Reported image metrics are the arithmetic mean of the 16 (novel) views in the test set. Note that the split sum approximation is able to capture high frequency lighting. Probe from Poly Haven [76].

into the system. Other limitations include the static lighting assumption, not optimizing camera poses, and high compute resource and memory consumption during training. Apart from deepfakes, which are common to all scene reconstruction methods, we are not aware of and do not foresee nefarious use cases of our method.

In summary, we show results on par with state-of-the-art for view synthesis and material factorization, while directly optimizing an *explicit* representation: triangle meshes with materials and environment lighting. Our representation is, by design, directly compatible with modern 3D engines and modeling tools, which enables a vast array of applications and simplifies artist workflows. We perform end-to-end optimization driven by appearance of the rendered model, while previous work often sidestep the error from mesh extraction through Marching Cubes. Our method can be applied as an appearance-aware converter from a (neural) volumetric or SDF representation to triangular 3D models with materials, complementing many recent techniques.

References

[1] Waleed Abdulla. Mask R-CNN for Object Detection and Instance Segmentation on Keras and TensorFlow. https://github.com/matterport/Mask_RCNN, 2017. 17

[2] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building Rome in a Day. *Communications of the ACM*, 54(10):105–112, 2011. 2

[3] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. FLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3(2):15:1–15:23, 2020. 14

[4] Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. Neural Reflectance Fields for Appearance Acquisition. *arXiv:2008.03824*, 2020. 2

[5] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P.A. Lensch. NeRD: Neural Reflectance Decomposition from Image Collections. In *IEEE International Conference on Computer Vision (ICCV)*, 2021. 1, 2, 5, 6, 7, 8, 16

[6] Mark Boss, Varun Jampani, Raphael Braun, Ce Liu, Jonathan T. Barron, and Hendrik P.A. Lensch. Neural-PIL: Neural Pre-Integrated Lighting for Reflectance Decomposition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 2

[7] Mark Boss, Varun Jampani, Kihwan Kim, Hendrik Lensch, and Jan Kautz. Two-shot Spatially-varying BRDF and Shape Estimation. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3982–3991, 2020. 2

[8] Brent Burley. Physically Based Shading at Disney. In *SIGGRAPH Courses: Practical Physically Based Shading in Film and Game Production*, 2012. 4

[9] Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer. In *Advances In Neural Information Processing Systems (NeurIPS)*, 2019. 2, 5

[10] Wenzheng Chen, Joey Litalien, Jun Gao, Zian Wang, Clement Fuji Tsang, Sameh Khalis, Or Litany, and Sanja Fidler. DIB-R++: Learning to Predict Lighting and Material with a Hybrid Differentiable Renderer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 2

[11] R. L. Cook and K. E. Torrance. A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics*, 1(1):7–24, 1982. 5

[12] Jeremy S De Bonet and Paul Viola. Poxels: Probabilistic Voxelized Volume Reconstruction. In *Proceedings of International Conference on Computer Vision (ICCV)*, 1999. 2

[13] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. JaxNeRF: an Efficient JAX Implementation of NeRF, 2020. 15

[14] Yasutaka Furukawa and Jean Ponce. Accurate, Dense, and Robust Multiview Stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–1376, 2009. 2

[15] Silvano Galliani, Katrin Lasinger, and Konrad Schindler. Gipuma: Massively Parallel Multi-view Stereo Reconstruction. *Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation e. V*, 25(361-369):2, 2016. 2

[16] Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. Deep Inverse Rendering for High-Resolution SVBRDF Estimation from an Arbitrary Number of Images. *ACM Transactions on Graphics*, 38(4), 2019. 2

[17] Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning Deformable Tetrahedral Meshes for 3D Reconstruction. In *Advances In Neural Information Processing Systems*, 2020. 2

[18] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-Fidelity Neural Rendering at 200FPS. *arXiv:2103.10380*, 2021. 2

[19] Andrew Gardner, Chris Tchou, Tim Hawkins, and Paul Debevec. Linear Light Source Reflectometry. *ACM Transactions on Graphics*, 22(3):749–758, 2003. 2

[20] Abhijeet Ghosh, Tongbo Chen, Pieter Peers, Cyrus A. Wilson, and Paul Debevec. Estimating Specular Roughness and Anisotropy from Second Order Spherical Gradient Illumination. *Computer Graphics Forum*, 28(4):1161–1170, 2009. 2

[21] D. Guarnera, G. C. Guarnera, A. Ghosh, C. Denk, and M. Glencross. BRDF Representation and Acquisition. In *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: State of the Art Reports*, pages 625–650, 2016. 2

[22] Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. MaterialGAN: Reflectance Capture Using a Generative SVBRDF Model. *ACM Transactions on Graphics*, 39(6), 2020. 2

[23] M. Haindl and J. Filip. *Visual Texture*. Springer-Verlag, 2013. 2

[24] Andrew Svanberg Hamilton. Photogrammetry at Embark, 2021. https://medium.com/embarkstudios/photogrammetry-at-embark-part-1-88142f2e036e. 1

[25] Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. Appearance-Driven Automatic 3D Model Simplification. In *Eurographics Symposium on Rendering*, 2021. 2, 3, 4, 12, 19

[26] Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. SAPE: Spatially-Adaptive Progressive Encoding for Neural Optimization. *arXiv preprint arXiv:2104.09125*, 2021. 18

[27] Krishna Murthy J., Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A PyTorch Library for Accelerating 3D Deep Learning Research. *arXiv:1911.05063*, 2019. 2

[28] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanæs. Large Scale Multi-view Stereopsis Evaluation. In *2014 IEEE Conference on Computer Vision and*

Pattern Recognition (CVPR), pages 406–413. IEEE, 2014. 17, 18, 20

[29] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. SDFDiff: Differentiable Rendering of Signed Distance Fields for 3D Shape Optimization. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[30] James T. Kajiya. The Rendering Equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH'86*. ACM Press, 1986. 5

[31] Brian Karis. Real Shading in Unreal Engine 4. *SIGGRAPH Course: Physically Based Shading in Theory and Practice*, 2013. 3, 5, 6

[32] Michael Kazhdan and Hugues Hoppe. Screened Poisson Surface Reconstruction. *ACM Transactions on Graphics*, 32(3):1–13, 2013. 2

[33] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, 2015. 3, 19

[34] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. PointRend: Image Segmentation as Rendering. *arXiv:1912.08193*, 2019. 17

[35] Sebastien Lagarde and Charles de Rousiers. Moving Frostbite to Physically Based Rendering 3.0. *SIGGRAPH Course: Physically Based Shading in Theory and Practice*, 2014. 3

[36] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular Primitives for High-Performance Differentiable Rendering. *ACM Transactions on Graphics*, 39(6), 2020. 2, 19

[37] Hendrik P.A. Lensch, Jochen Lang, Asla M. Sa, and Hans-Peter Seidel. Planned Sampling of Spatially Varying BRDFs. *Computer Graphics Forum*, 2003. 2

[38] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Inverse Rendering for Complex Indoor Scenes: Shape, Spatially-varying Lighting and SVBRDF from a Single Image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2475–2484, 2020. 2

[39] Zhengqin Li, Zexiang Xu, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Learning to Reconstruct Shape and Spatially-Varying Reflectance from a Single Image. *ACM Transactions on Graphics*, 37(6), 2018. 2

[40] Yiyi Liao, Simon Donné, and Andreas Geiger. Deep Marching Cubes: Learning Explicit Surface Representations. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 4, 20

[41] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, 2019. 2

[42] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987. 4

[43] Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. Unified Shape and SVBRDF Recovery using Differentiable Monte Carlo Rendering. In *Eurographics Symposium on Rendering*, 2021. 2

[44] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7210–7219, 2021. 2

[45] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020. 1, 2, 4, 7, 12, 14, 15, 18

[46] Thomas Müller. Tiny CUDA Neural Network Framework, 2021. https://github.com/nvlabs/tiny-cuda-nn. 5

[47] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *arXiv:2201.05989*, 2022. 2, 5, 16

[48] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11453–11464, 2021. 2

[49] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2, 4, 16, 17, 18

[50] Merlin Nimier-David, Zhao Dong, Wenzel Jakob, and Anton Kaplanyan. Material and Lighting Reconstruction for Complex Indoor Scenes with Texture-space Differentiable Rendering. In *Eurographics Symposium on Rendering*, 2021. 2

[51] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering. *ACM Transactions on Graphics*, 39(4), 2020. 3

[52] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Transactions on Graphics*, 38(6), 2019. 3, 7

[53] Michael Oechsle, Songyou Peng, and Andreas Geiger. UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 2, 4, 17

[54] Marc Pollefeys and Luc Van Gool. From Images to 3D Models. *Commun. ACM*, 45(7):50–55, 2002. 1

[55] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10318–10327, 2021. 2

[56] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. *arXiv:2103.13744*, 2021. 2

[57] Carolin Schmitt, Simon Donne, Gernot Riegler, Vladlen Koltun, and Andreas Geiger. On Joint Estimation of Pose, Geometry and svBRDF from a Handheld Scanner. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[58] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 1, 2

[59] Steven M Seitz and Charles R Dyer. Photorealistic Scene Reconstruction by Voxel Coloring. *International Journal of Computer Vision*, 35(2):151–173, 1999. 2

[60] 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. 2, 3, 4

[61] Smithsonian. Smithsonian 3D Digitization, 2018. https://3d.si.edu/. 4, 12, 13, 20

[62] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo Tourism: Exploring Photo Collections in 3D. *ACM Transactions on Graphics*, 25(3):835–846, 2006. 1

[63] Olga Sorkine. Laplacian Mesh Processing. In *Eurographics 2005 - State of the Art Reports*, 2005. 19

[64] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2

[65] Michael Stokes, Matthew Anderson, Srinivasan Chandrasekar, and Ricardo Motta. A Standard Default Color Space for the Internet - sRGB, 1996. 19

[66] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet Models for Refraction through Rough Surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, page 195–206, 2007. 4, 5

[67] 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. 2, 4, 15, 17

[68] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF−−: Neural Radiance Fields Without Known Camera Parameters. *arXiv:2102.07064*, 2021. 2

[69] Michael Weinmann and Reinhard Klein. Advances in Geometry and Reflectance Acquisition (Course Notes). In *SIGGRAPH Asia Courses*, 2015. 2

[70] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. NeX: Real-time View Synthesis with Neural Basis Expansion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2

[71] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 17

[72] Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Learning Object-Compositional Neural Radiance Field for Editable Scene Rendering. In *International Conference on Computer Vision (ICCV)*, 2021. 3

[73] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. *Advances in Neural Information Processing Systems*, 33, 2020. 2, 16, 17

[74] Jonathan Young. xatlas, 2021. https://github.com/jpcy/xatlas. 5

[75] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *International Conference on Computer Vision (ICCV)*, 2021. 2

[76] Greg Zaal. Poly Haven, 2021. https://polyhaven.com/hdris. 8, 20

[77] Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. PhySG: Inverse Rendering with Spherical Gaussians for Physics-based Material Editing and Relighting. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 1, 2, 5, 6

[78] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492*, 2020. 2

[79] Xiuming Zhang, Pratul P. Srinivasan, Boyang Deng, Paul Debevec, William T. Freeman, and Jonathan T. Barron. NeRFactor: Neural Factorization of Shape and Reflectance under an Unknown Illumination. *ACM Transactions on Graphics*, 40(6), 2021. 1, 2, 5, 6, 7, 13, 19

[80] Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. Image GANs meet Differentiable Rendering for Inverse Graphics and Interpretable 3D Neural Rendering. In *International Conference on Learning Representations*, 2021. 2

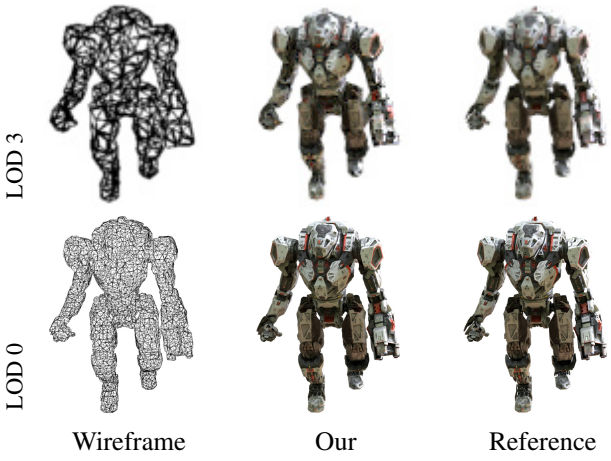


Figure 12. Automatic LOD example: We generated 256 views of an object (with masks & poses) from a path tracer in two resolutions: 1024×1024 and 128×128 pixels, then reconstructed the mesh, materials and lighting to approximate LOD creation. **Top**: LOD level optimized to look good at a resolution of 128×128 pixels with 3k triangles. **Bottom**: LOD level optimized to look good at a resolution of 1024×1024 pixels with 63k triangles.

6. Supplemental Material

In the following, we supplement the paper with additional results, ablations and implementation details. In Section 7 we present novel use cases for our method: automatic level of detail creation from images, and appearance aware model extraction. In Section 8, we present additional results, including an evaluation of geometric quality, additional per-scene statistics and visual examples. Finally, in Section 9 we provide implementation details, including efficient split-sum pre-integration, regularizer terms and losses.

7. Novel applications

7.1. Level-of-detail From Images

Inspired by a recent work in appearance-driven automatic 3D model simplification [25], we demonstrate level-of-detail (LOD) creation directly from rendered images of an object. The previous technique requires an initial guess with fixed topology and known lighting. We generalize this approach and showcase LOD creation directly from a set of images, i.e., we additionally learn both topology and lighting. To illustrate this, we generated 256 views (with masks & poses) from a path tracer, rendered in two resolutions: 1024×1024 pixels and 128×128 pixels, then reconstructed the mesh, materials and lighting in our pipeline to create two LOD levels (geometry and spatially-varying materials). We show visual results in Figure 12 and in the supplemental video.

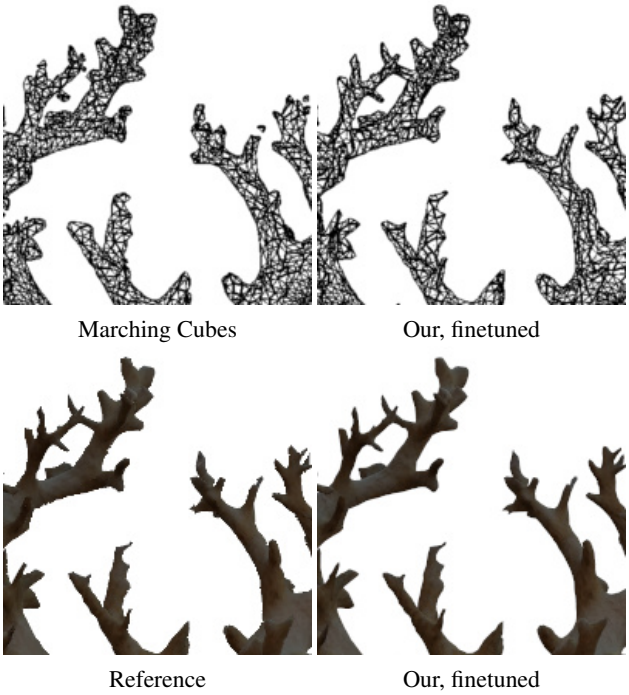


Figure 13. Appearance-aware NeRF 3D model extraction. We show insets of the silhouette quality before and after our optimization pass, alongside insets of the reference and our rendered result.

7.2. Appearance-Aware NeRF 3D Model Extractor

We devise a way to extract 3D models from neural radiance fields [45] (NeRF) in a format compatible with traditional 3D engines. Our pipeline for this task has three steps:

$$\text{NeRF} \rightarrow \text{Marching Cubes} \rightarrow \text{Differentiable renderer}.$$

The dataset consists of 256 images of the Damicornis model [61] (with masks and poses), rendered in a path tracer. We first train a NeRF model and extract the mesh with Marching Cubes. Next, we finetune the extracted mesh and learn materials parameters (2D textures) using our differentiable renderer (with DMTet topology optimization disabled), still only supervised by the images in the dataset. The output is a triangle mesh with textured PBR materials compatible with traditional engines. As a bonus, the silhouette quality improves over the Marching Cubes extraction, which is illustrated in Figure 13.

7.3. 3D Model Extraction with Known Lighting

We observe that the DMTet representation successfully learns challenging topology and materials jointly, even for highly specular models and when lit using high frequency lighting. We illustrate this in a joint shape and material optimization task with known environment lighting, optimized using a large number of views. In Figure 14 we show two

Figure 14. DMTet can accurately capture topology, even in challenging scenarios. To illustrate this, we show two examples from the Smithsonian 3D repository [61], where we jointly learn topology and materials under known environment lighting. The left column shows our approximation extracted from multiple 2D observations (5000 views) and the right side a rendering of the reference model. In both examples, we start from a tet grid of resolution 128^3 and optimize the grid SDF values, vertex offsets and material parameters.

examples from the Smithsonian 3D repository [61]. Note the quality in both the extracted materials and geometric detail.

8. Results

8.1. Scene Editing and Simulation

This section supplements Section 4.1 in the main paper. In Table 5 we present per-scene breakdowns of relighting results corresponding to Table 2 in the main paper. An additional visual relighting example is shown in Figure 15, where we relight the Ficus scene with four different light probes, comparing to the results of NeRFactor [79]. Figure 16 shows a visual example of material separation with albedo, $\boldsymbol{k}_d$, and normals, $\mathbf{n}$. In Figure 30 we show our lighting, material and shape separation for all scenes in the NeRF synthetic dataset. We note that we achieve significantly more detailed normals (thanks to the tangent space normal map included in our shading model) and albedo mostly decorrelated from lighting. Our remaining challenges are areas with strong shadows or global illumination effects, which are currently not rendered in our simplified shading model used during optimization.

PSNR↑					
Scene	Drums	Ficus	Hotdog	Lego	Avg
NeRFactor	21.94	22.35	25.59	25.25	23.82
Our	22.63	25.71	28.77	21.03	25.24
SSIM↑					
Scene	Drums	Ficus	Hotdog	Lego	Avg
NerFactor	0.912	0.930	0.917	0.870	0.907
Our	0.915	0.961	0.932	0.846	0.911
LPIPS↓					
Scene	Drums	Ficus	Hotdog	Lego	Avg
NeRFactor	0.092	0.095	0.129	0.132	0.112
Our	0.083	0.046	0.092	0.119	0.085
FLIP↓					
Scene	Drums	Ficus	Hotdog	Lego	Avg
NeRFactor	0.083	0.081	0.110	0.095	0.092
Our	0.087	0.063	0.074	0.163	0.097

Table 5. Relighting quality results for the four scenes in NeRFactor's synthetic dataset. The reported metrics are the arithmetic mean over eight validation views relit with eight different light probes.

Figure 15. Relighting quality for a scene from the NeRFactor dataset, with our examples relit using Blender, and NeRFactor results generated using the public code.

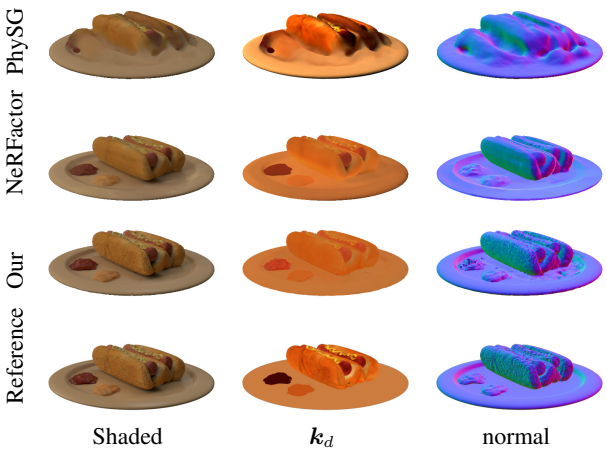


Figure 16. Extracted materials for a scene from the NeRFactor dataset. We directly compare albedo k_d and normals $\mathbf{n}$ to the results of NeRFactor. Specular parameters are omitted as we use different BSDF models. All k_d images have been renormalized using the reference albedo, as suggested in NeRFactor.

PSNR↑

Scene	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg
PhySG	21.87	16.45	17.40	21.57	18.81	18.02	19.16	18.06	18.91
NeRF	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65	31.00
MipNeRF	35.08	25.56	33.44	37.38	35.46	30.63	36.38	30.46	33.05
Our	31.60	24.10	30.88	33.04	29.14	26.74	30.78	26.12	29.05

SSIM↑

Scene	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg
PhySG	0.890	0.823	0.861	0.894	0.812	0.837	0.904	0.756	0.847
NeRF	0.967	0.925	0.964	0.974	0.961	0.949	0.980	0.856	0.947
MipNeRF	0.980	0.934	0.981	0.982	0.978	0.959	0.991	0.885	0.961
Our	0.969	0.916	0.970	0.973	0.949	0.923	0.977	0.833	0.939

LPIPS↓

Scene	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg
PhySG	0.122	0.188	0.144	0.163	0.208	0.182	0.108	0.343	0.182
NeRF	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206	0.081
MipNeRF	0.041	0.104	0.045	0.038	0.053	0.054	0.024	0.177	0.067
Our	0.045	0.101	0.048	0.060	0.061	0.100	0.040	0.191	0.081

FLIP↓

Scene	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg
PhySG	0.088	0.140	0.115	0.109	0.139	0.139	0.069	0.159	0.119
MipNeRF	0.028	0.073	0.035	0.026	0.036	0.043	0.016	0.061	0.040
Our	0.034	0.065	0.041	0.033	0.042	0.060	0.024	0.080	0.047

Table 6. Image quality metrics for the NeRF realistic synthetic dataset. Each training set consists of 100 images with masks and known camera poses, and the reported image metrics are the arithmetic mean over the 200 images in the test set. Results for NeRF are based on Table 4 of the original paper [45], with new measurements for PhySG and MipNeRF using their respective publicly available source code. We additionally report FLIP mean scores [3]. Note that the Hotdog outlier LPIPS score for NeRF is consistent with the original paper, but probably a bug.

8.2. View interpolation

This section supplements Section 4.2 in the main paper. In Table 6 we show per-scene breakdowns of the view-interpolation results corresponding to Table 3 in the main paper, evaluated on the NeRF Synthetic Dataset. Figure 17 shows a visual comparison to PhySG and MipNeRF for

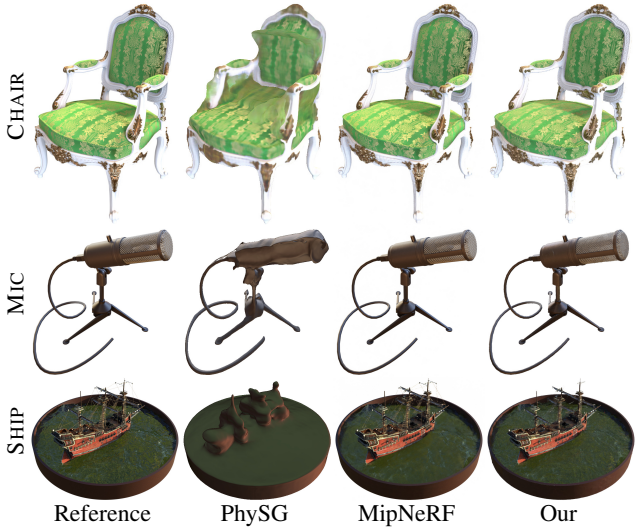


Figure 17. Visual quality examples from the NeRF realistic synthetic dataset comparing our method to PhySG and MipNeRF. PhySG struggles to accurately capture the complex geometry and spatially varying materials of the dataset.

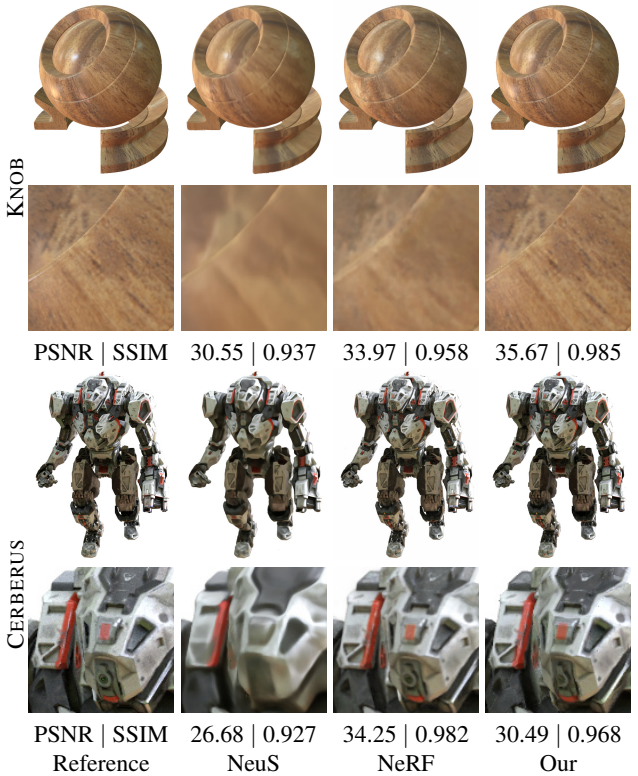


Figure 18. Visual quality examples on the synthetic Knob and Cerberus datasets. We observe slightly blurry results from NeuS.

PSNR↑

Scene	Drums	Ficus	Hotdog	Lego	Avg
PhySG	14.35	15.25	24.49	17.10	17.80
NeRF	27.67	28.05	36.71	31.89	31.08
NeRFactor	24.63	23.14	31.60	28.12	26.87
Our	28.45	31.20	36.26	30.70	31.65

SSIM↑

Scene	Drums	Ficus	Hotdog	Lego	Avg
PhySG	0.807	0.838	0.909	0.771	0.831
NeRF	0.951	0.957	0.971	0.944	0.956
NeRFactor	0.933	0.937	0.948	0.900	0.930
Our	0.959	0.978	0.981	0.951	0.967

LPIPS↓

Scene	Drums	Ficus	Hotdog	Lego	Avg
PhySG	0.215	0.176	0.153	0.278	0.206
NeRF	0.069	0.055	0.058	0.075	0.064
NeRFactor	0.082	0.087	0.101	0.124	0.099
Our	0.063	0.047	0.048	0.057	0.054

FLIP↓

Scene	Drums	Ficus	Hotdog	Lego	Avg
PhySG	0.163	0.133	0.076	0.168	0.135
NeRF	0.045	0.045	0.030	0.037	0.039
NeRFactor	0.058	0.071	0.050	0.058	0.059
Our	0.037	0.037	0.023	0.030	0.032

Table 7. View interpolation results for the four scenes of NeRFactor's synthetic dataset. The NeRF column shows the baseline NeRF trained as part of NeRFactor's setup, and is different from the NeRF in our other view interpolation results. Each training set consists of 100 images with masks and known camera poses, and the reported image metrics are the arithmetic mean over the eight images in the test set.

Chamfer Loss↓

Scene	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship
PhySG	0.1341	0.4236	0.0937	0.2420	0.2592	-	0.2712	0.7118
NeRF (w/o mask)	0.0185	0.0536	0.0115	4.6010	0.0184	0.0057	0.0124	2.0111
NeRF (w/ mask)	0.0435	0.0326	0.0145	0.0436	0.0201	0.0082	0.0122	0.2931
Our	0.0574	0.0325	0.0154	0.0272	0.0267	0.0180	0.0098	0.3930

Triangles↓ (Thousands)

Scene	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship
PhySG	353	439	489	725	498	-	386	557
NeRF (w/o mask)	192	261	585	869	2259	2411	261	1087
NeRF (w/ mask)	494	548	440	694	1106	594	307	3500
Our	102	65	39	57	111	58	22	190

Table 8. Chamfer loss and triangle counts for reconstructed meshes for the NeRF realistic synthetic dataset. We compare to the meshes produced by PhySG, and also generate meshes from the NeRF volume using density thresholding and marching cubes. Note that we primarily focus on opaque geometry, so the DRUMS, SHIP, and FICUS scenes with transparency are challenging cases.

	KNOB	CERBERUS	DAMICORNIS
NeRF [45]	2.77e-01	9.08e-03	3.34e-03
NeuS [67]	2.04e-01	2.84e-02	5.84e-04
Our	1.87e-01	1.03e-02	4.66e-04

Figure 19. Synthetic examples with increasing complexity. Each dataset consists of 256 rendered images at a resolution of 1024×1024 pixels. We report Chamfer L_1 scores on the extracted meshes for NeRF (neural volume), NeuS (neural implicit), and our explicit approach. Lower score is better.

the CHAIR, MICROPHONE and SHIP scenes. We note that PhySG struggles to capture the complex geometry of the NeRF dataset.

To study view interpolation quality for techniques which support material decomposition, we report per-scene breakdowns of view-interpolation result in Table 7. This corresponds to Table 4 in the main paper. We use the NeRFactor dataset (which is a subset of the NeRF dataset with simplified lighting conditions) and compare with NeRFactor and PhySG.

In Figure 18 we additionally compare view interpolation quality on a small synthetic dataset containing three scenes with increasing geometric complexity: KNOB, DAMICORNIS and CERBERUS, each dataset consists of 256 views with masks and known camera poses, and is validated on 200 novel views. We compare against NeRF (neural volumetric representation) and NeuS [67] (neural implicit representation). We provided masks at training for both approaches. We note that on this dataset, our method performs on par with NeRF, and consistently produces results with greater detail and sharpness than NeuS.

8.3. Geometry

Our primary targets are appearance-aware 3D reconstructions which render efficiently in real-time (e.g. for a game or interactive path tracer). As part of that goal, our shading model includes tangent space normal maps, which is a commonly used technique to capture the appearance of high frequency detail at modest triangle counts. For these reasons, we consider image quality our main evaluation metric, but additionally report Chamfer scores in Table 8 for completeness. When comparing with NeRF [45], we use pretrained checkpoints provided by JaxNeRF[1] [13], which we denote NeRF w/o mask. We note that the pretrained models suffer greatly from floater geometry in some scenes. To that end, we additionally show results for NeRF (w/

[1] https://github.com/google-research/google-research/tree/master/jaxnerf

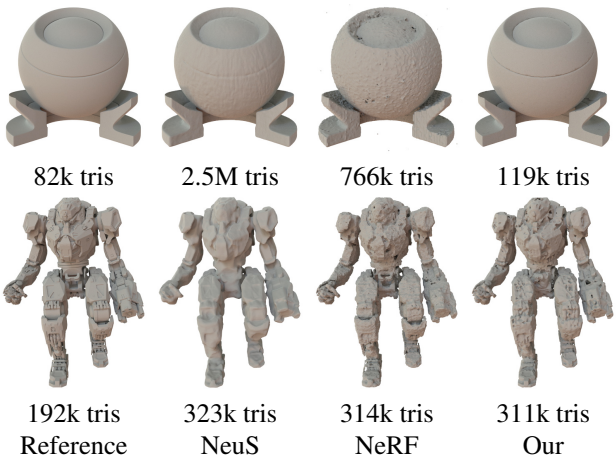


Figure 20. Extracted mesh quality visualization examples on the synthetic KNOB and CERBERUS datasets.

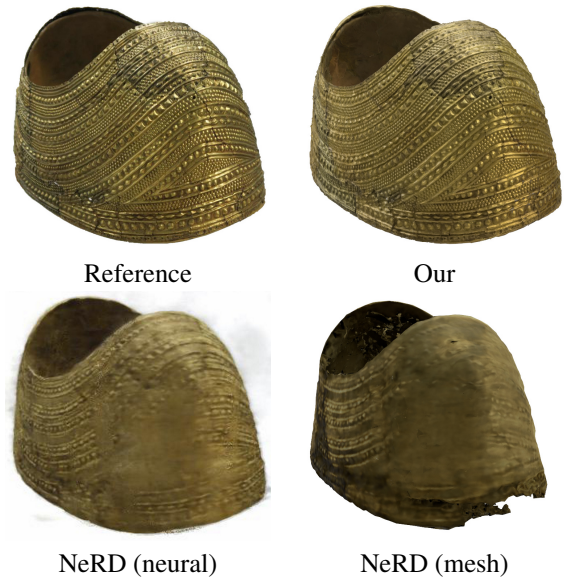


Figure 21. Example of the quality of the neural NeRD representation and their final generated mesh. Note the quality loss in both geometry and appearance (textures).

mask) which further utilizes coverage masks and regularizes density, trading some image quality for better geometric accuracy. We use the NGP-NeRF [47] code base to generate the NeRF (w/ mask) results. To calculate the Chamfer scores, we sample 2.5M points on both predicted mesh and ground mesh respectively, and calculate the Chamfer distance between the two point clouds.

While our meshes have considerably lower triangle count that the MC extractions, we are still competitive in terms of Chamfer loss. Note that we primarily focus on opaque geometry, hence, the DRUMS, SHIP, and FICUS scenes with transparency are challenging cases.

Figure 22. Examples of masking errors for the *Mold Gold Cape* dataset. Note the inconsistencies in classifying the plastic mount as both part of the object and background.

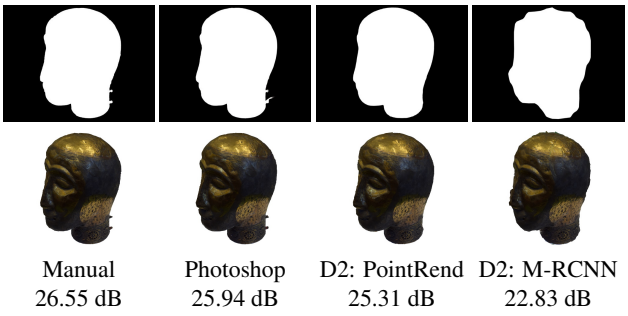


Figure 23. We compare four segmentation techniques for the *Ethiopian Head* dataset: The original (crowdsourced) masks, automatic segmentation in Photoshop (using the object selection tool), and two versions of Detectron2. For Detectron2, we run two pretrained instance segmentation models which predict accurate and coarse segmentations respectively. PSNR↑ scores are the arithmetic mean of all reconstructed frames.

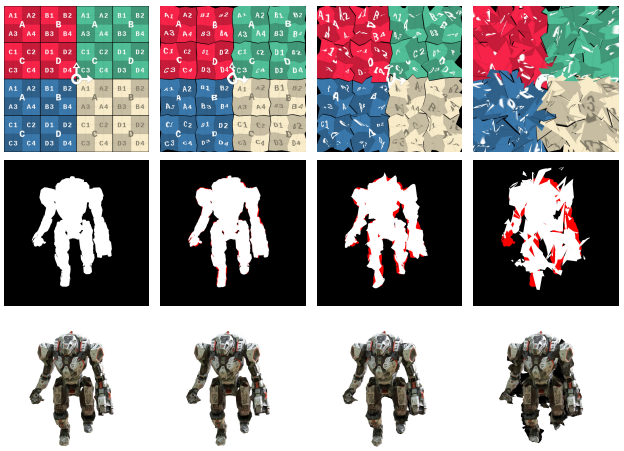
In Figure 19, we report Chamfer loss on three synthetic datasets of increasing geometric complexity. Interestingly, the neural implicit version performs very well on the organic shapes, but struggles on the CERBERUS robot model, where NeRF provide the lowest Chamfer loss. Visual comparisons of rendered reconstruction quality are included in Figure 18 and a visualization of the Lambertian shaded mesh is included in Figure 20.

We additionally present an example of an output mesh generated by NeRD [5] in Figure 21. The impact of the mesh extraction step is notable, both to geometry and material quality. As we only have this single data point, with no means of accurately aligning the meshes for measuring geometric loss (NeRD does not provide source code), we will not provide metrics.

8.4. Quality of Segmentation Masks

Like many related works (e.g. NeRD [5], DVR [49], and IDR [73]) our method relies on foreground segmentation masks. While this is a limitation we hope to see lifted in future work, we note that our method is robust to moderate levels of mask corruption, as can be expected from automated methods or crowdsourced annotation.

Both the DTU MVS dataset (Figure 25) and the NeRD dataset (Figure 21) rely on manually annotated masks, with some frames containing large errors and inconsistencies, as



	$\sigma = 0$	$\sigma = 5$ px	$\sigma = 16$ px	$\sigma = 51$ px
U	26.94 dB	25.60 dB	22.93 dB	17.85 dB
C	26.94 dB	24.36 dB	20.09 dB	16.08 dB

Figure 24. To evaluate the impact of corrupted masks, we warp perfect masks by texture-mapping them on a grid, displacing each of the 25×25 vertices by zero-mean Gaussian noise with increasing standard deviation, σ. From top to bottom, we show a warped texture (to give a sense of the magnitude of corruption), the corrupted masks with the reference mask shown in red, and our reconstruction. The training set consists of 200 images, and PSNR↑ scores are computed as the arithmetic mean of 50 validation images. The 'uncorrelated' series, **U**, are generated with unique random numbers for each frame, while in the "correlated" scores, **C**, we corrupt all masks using the *same* random seed, simulating a segmentation with systematic bias.

shown in Figure 22. In Figure 23 we automatically generate segmentation masks of varying quality for a real-world dataset. We generate masks using two versions of Detectron2 [71], namely PointRend [34] and Mask R-CNN [1]. Additionally, we generated another version of masks using the "object finder" tool in Adobe Photoshop 2022. As expected, reconstruction quality decreases gracefully with lower mask quality. Subjectively, the silhouette looks best using the automatic mask generated in Photoshop.

In Figure 24, we show a synthetic experiment where we corrupt perfect masks with increasing levels of noise. Reconstruction quality decreases gracefully as a function of corruption level, and while the quality reduction is significant, our system is stable even for large corruptions. Here, all masks in the dataset are corrupted, while segmentation algorithms typically produce good results with a few localized errors, so this experiment is a stress-test even for low noise levels. We speculate that *surface-based* representations more robustly handle mask corruptions, as inaccuracies in silhouettes are less objectionable than the "floater" geometry generated by density-based approaches.

	Chamfer loss↓				
Scene	NeRF	DVR	Our	IDR	NeuS
scan65	1.44	1.06	1.03	0.79	0.72
scan106	1.44	0.95	1.07	0.67	0.66
scan118	1.13	0.71	0.69	0.51	0.51

Table 9. Quantitative evaluation on the DTU dataset w/ mask. Chamfer distances are measured in the same way as NeuS [67], IDR [73], and DVR [49]. Results for NeRF, IDR and NeuS are taken from Table 1 in the NeuS paper [67], and the DVR results are taken from Table 8, 9 and 10 in the DVR supplemental material. We also reevaluated the DVR scores using the DTU MVS dataset evaluation scripts [28] to verify the evaluation pipeline. Our Chamfer distances are lower than NeRF, roughly on par with DVR, but higher than the current state-of-the-art (IDR/NeuS). Still, we find these results encouraging, considering that we provide an explicit mesh with factorized materials.

8.5. Multi-View Stereo Datasets

Our experiments with scans from a limited view angle, low number of views, and/or varying illumination, e.g., the DTU MVS datasets [28], shows that our approach work less well than the recent neural implicit versions, such as NeuS [67], Unisurf [53], and IDR [73], which we attribute to a more regularized, smoother shape representation for the neural implicit approaches, and our physically-bases shading model which assumes constant lighting. We provide quantitative results for three scans from DTU in Table 9, and visual examples of our results on three scans in Figure 25.

The sparse viewpoints and varying illumination (which breaks our shading model assumption of constant lighting) in the DTU datasets lead to strong ambiguity in the reconstructed geometry. In this case, we noticed that directly optimizing the per-vertex SDF values results in high-frequency noise in the surface mesh. Instead, we follow the approach of the neural implicit approaches and use an MLP to parametrize the SDF values, which implicitly regularize the SDF, and, as a consequence, the resulting surface geometry produced by DMTet. The smoothness of the reconstructed shape can be controlled by the frequency of the positional encoding applied to the inputs of the MLP, as shown in Figure 26. On the contrary, in case of densely sampled viewpoints and constant illumination, we observed that directly optimizing per-vertex attributes better captures high-frequency details, as shown in Figure 27, and is faster to train. We use direct optimization of per-vertex SDF values in all results presented in the paper, except for the DTU scans, and the NeRF hotdog example, where we obtained better geometry reconstruction using the MLP parameterization.

We use the same MLP as in DVR [49], which consists of five fully connected residual layers with 256 hidden fea-

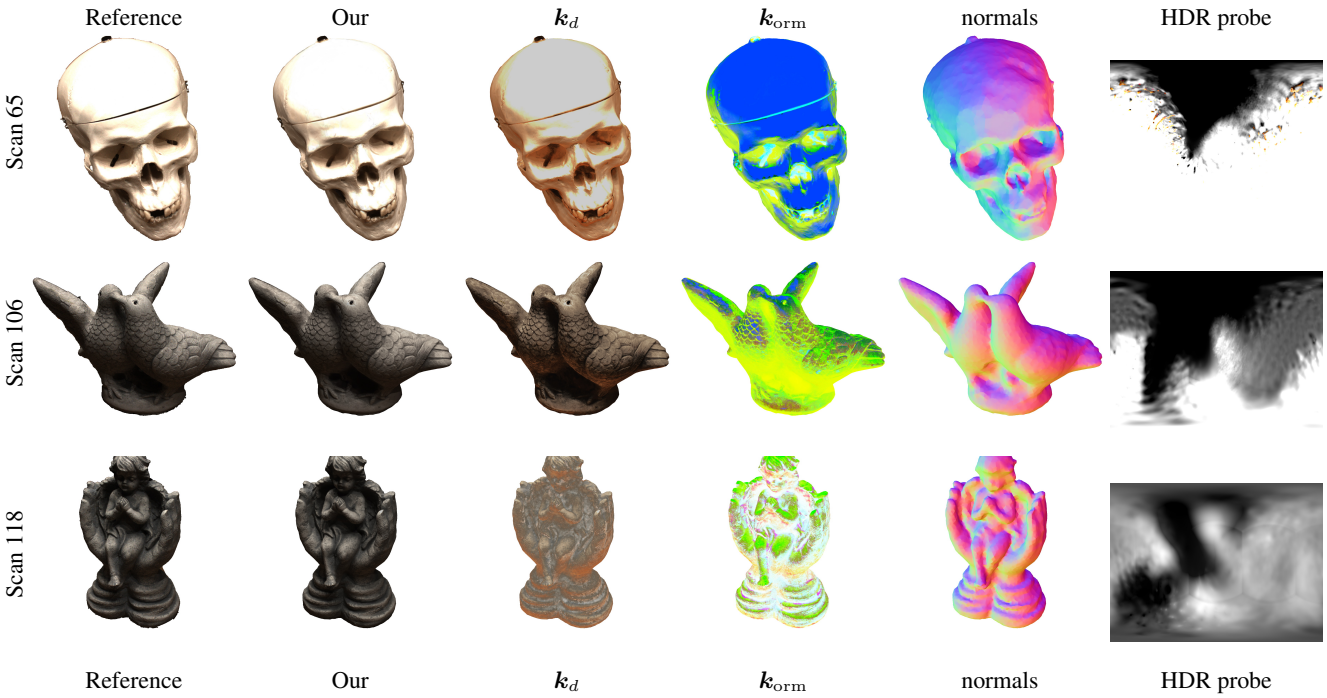


Figure 25. Our decomposition results on scan 65, 106, and 118 of the DTU MVS dataset [28]. Our model is trained on a reduced subset (49 of the 64 views) which has more consistent lighting across views, labelled by DVR [49]. However, we still penalize mask loss on the excluded views.

Figure 26. Comparing grid vs. MLP parametrizations of DMTet on scan 65 from the DTU MVS dataset [28]. Directly optimizing SDF values at grid vertices leads to a surface with high-frequency noise (left). In contrast, if we use an MLP to parametrize the SDF values, we can regularize the geometry, with smoothness controlled by the frequency of positional encoding. We use the positional encoding in NeRF [45] with frequency set to 4 (middle) and 6 (right) respectively.

tures. In addition, we adopt the positional encoding in NeRF [45] and progressively fit the frequencies similar to SAPE [26]. More specifically, for an input position p and a set of encoding functions $e_1, e_2, \ldots, e_n$ with increasing frequencies, we multiply each encoding $e_n(p)$ with a soft mask $\alpha_n(t)$ at training iteration t. The first n_{base} encodings are always exposed to the network, and we linearly reveal

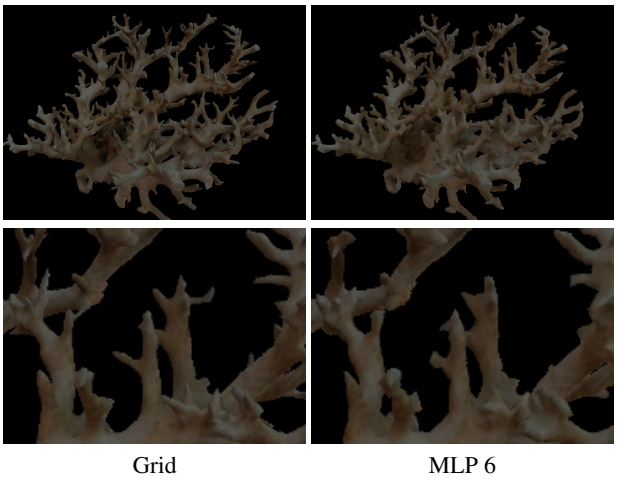


Figure 27. Comparing grid vs. MLP parametrization of DMTet on the synthetic DAMICORNIS dataset. Directly optimizing per-vertex SDF and offsets on a grid is faster to train, and better captures high-frequency geometric details than parametrizing DMTet with a network.

the rest during training such that:

$$\alpha_n(t) = \begin{cases} 1 & n \leq n_{base} \\ \min(1, \frac{t}{t_f}) & n > n_{base} \end{cases} \quad (6)$$

where t_f is the iteration when all encodings are fully revealed. In practice, we find that progressively fitting the frequencies produces less high-frequency artifacts on the reconstructed surface than the non-progressive scheme.

9. Implementation

9.1. Optimization

Unless otherwise noted, we start from a tetrahedral grid of resolution 128 (using 192k tetrahedra and 37k vertices). As part of the Marching Tetrahedral step, each tetrahedron can generate up to two triangles.

We initialize the per-vertex SDF values to random values in the range $[-0.1, 0.9]$, such that a random selection of approximately 10% of the SDF values will report "inside' status at the beginning of optimization. The per-vertex offsets are initialized to zero.

Textures are initialized to random values within the valid range. We also provide min/max values per texture channels, which are useful when optimizing from photographs, where we follow NeRFactor [79] and us a range on the albedo texture of $\boldsymbol{k}_d \in [0.03, 0.8]$. Similarly, we limit the minimal roughness value (green channel of the $\boldsymbol{k}_{\text{orm}}$ texture) to 0.08 (linearized roughness). The tangent space normal map is initialized to $(0, 0, 1)$, i.e., following the surface normal with no normal perturbation. The environment light texels are initialized to random values in the range $[0.25, 0.75]$, which we empirically found to be a reasonable starting point in our tests.

We use the Adam [33] optimizer with default settings combined with a learning rate scheduler with an exponential falloff from 1.0 to 0.1 over 5000 iterations. We typically train for 5000 iteration using a mini-batch of eight images, rendered at the native resolution of the images in the datasets (typically in the range from 512×512 pixels to 1024×1024 pixels). Next, after texture reparametrization, we finetune geometry and 2D textures with locked topology for another 5000 iterations. The entire process takes approximately an hour on a single NVIDIA V100 GPU, with indicative results after a few minutes. We include a training visualization in the supplemental video.

In DTU experiments, we set $n = 6$, $n_{base} = 4$ and $t_f = 2500$ for the progressive positional encoding. We disable the normal perturbation and second-stage optimization to get the best geometric quality, and train DMTet for 10k iterations.

9.2. Losses and Regularizers

Image Loss Our renderer uses physically based shading and produces images with high dynamic range. Therefore, the objective function must be robust to the full range of floating-point values. Following recent work in differentiable rendering [25], our image space loss, L_{image}, computes the L_1 norm on tone mapped colors. As tone map operator, we transform linear radiance values, x, according to $x' = \Gamma(\log(x+1))$, where $\Gamma(x)$ is the sRGB transfer function [65]:

$$\begin{aligned} \Gamma(x) &= \begin{cases} 12.92x & x \leq 0.0031308 \\ (1+a)x^{1/2.4} - a & x > 0.0031308 \end{cases} \\ a &= 0.055. \end{aligned} \tag{7}$$

Light Regularizer Real world datasets contain primarily neutral, white lighting. To that end, we use a regularizer for the environment light that penalizes color shifts. Given the per-channel average intensities $\overline{c_i}$, we define the regularizer as:

$$L_{\text{light}} = \frac{1}{3}\sum_{i=0}^{3}\left|\overline{c_i} - \frac{1}{3}\sum_{i=0}^{3}\overline{c_i}\right|. \tag{8}$$

Material Regularizer As mentioned in the paper, we regularize material parameters using a smoothness loss similar to NeRFactor [79]. Assuming that $\boldsymbol{k}_d(\mathbf{x})$ denotes the $\boldsymbol{k}_d$ parameter at world space position $\mathbf{x}$ and ϵ is a random displacement vector, we define the regularizer as:

$$L_{\text{mat}} = \sum_{\mathbf{x}_{\text{surf}}} \left|\boldsymbol{k}_d(\mathbf{x}_{\text{surf}}) - \boldsymbol{k}_d(\mathbf{x}_{\text{surf}} + \epsilon)\right|. \tag{9}$$

To account for the lack of global illumination and shadowing in our differentiable renderer, we use an additional, trainable visibility term which can be considered a regularizer. We store this term in the otherwise unused o-channel of the $\boldsymbol{k}_{\text{orm}}$ specular lobe parameter texture and use it to directly modulate the radiance estimated by our split sum shading model. Thus, it is similar to a simple ambient occlusion term and does not account for directional visibility.

Laplacian Regularizer In the second pass, when topology is locked, we use a Laplacian regularizer [63] on the triangle mesh to regularize the vertex movements. The uniformly-weighted differential $\boldsymbol{\delta}_i$ of vertex $\boldsymbol{v}_i$ is given by $\boldsymbol{\delta}_i = \boldsymbol{v}_i - \frac{1}{|N_i|}\sum_{j\in N_i}\boldsymbol{v}_j$, where N_i is the one-ring neighborhood of vertex $\boldsymbol{v}_i$. We follow Laine et al. [36] and use a Laplacian regularizer term given by

$$L_{\boldsymbol{\delta}} = \frac{1}{n}\sum_{i=1}^{n}\left\|\boldsymbol{\delta}_i - \boldsymbol{\delta}_i'\right\|^2, \tag{10}$$

where $\boldsymbol{\delta}_i'$ is the uniformly-weighted differential of the input mesh (i.e., the output mesh from the first pass).

SDF Regularizer If we only optimize for image loss, internal faces which are not visible from any viewpoint do not receive any gradient signal. This leads to random geometry

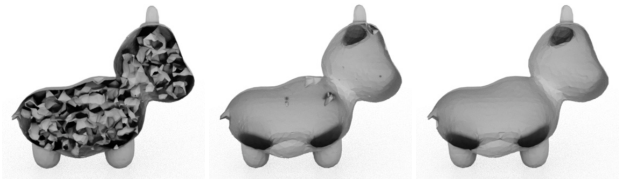


Figure 28. Cross sections of shapes optimized without regularization loss on SDF (left), with smoothness loss used by Liao et al. [40] (middle) and with our regularization loss (right). The random faces inside the object are removed by the regularization loss on SDF.

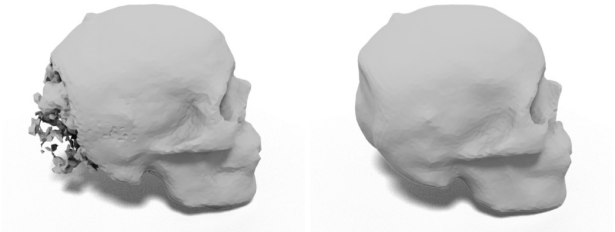


Figure 29. Reconstruction of scan 65 from the DTU MVS dataset [28] without (left) and with (right) the regularization loss based on visibility of faces. The regularization loss removes floaters behind the object that are not visible from the training views.

inside the object, as shown in Fig. 28, which is undesirable for extracting compact 2D textures. To remove the internal faces, we regularize the SDF values of DMTet similar to Liao et al. [40] as described in the main paper (Eqn. 2). The L_1 smoothness loss proposed by Liao et al., adapted from occupancy to SDF values, can be written as:

$$L_{\text{smooth}} = \sum_{i,j \in \mathbb{S}_e} |s_i - s_j|, \tag{11}$$

where $\mathbb{S}_e$ is the set of unique edges, and s_i represents the per-vertex SDF values. In contrast, our regularization loss explicitly penalizes the sign change of SDF values over edges in the tetrahedral grid. Empirically, our loss more efficiently removes internal structures, as shown in Fig. 28.

In our DTU experiments, we use an additional regularization loss to removes the floaters behind the visible surface, as illustrated in Fig 29. Specifically, for a triangular face f extracted from tetrahedron T, if f is not visible in current training views, we encourage the SDFs at vertices of T to be positive with BCE loss.

9.3. Split Sum Implementation Details

We represent the trainable parameters for incoming lighting as texels of a cube map (typical resolution $6 \times 512 \times 512$). The base level represents the pre-integrated lighting for the lowest supported roughness value, which then linearly increases per mip-level. Each filtered mip-map is computed by average-pooling the base level texels to the current resolution (for performance reasons, the quantization this process introduces is an acceptable approximation for our use case). Then, each level is convolved with the GGX normal distribution function. We pre-compute accurate filter bounds per mip-level (the filter bound is a function of the roughness, which is constant per mip level).

The loss gradients w.r.t. the inputs, $\frac{\partial L}{\partial X}$, for a convolution operation can be computed as a gather operation using products of the transposed weight tensor, W^T, and the output gradient, $\frac{\partial L}{\partial Y}$, within the filter footprint. However, in cube maps, the filter footprint may extend across cube edges or corners, which makes a gather operation non-trivial. We therefore express the gradient computation as a *scatter* operation, which can be efficiently implemented on the GPU using non-blocking `atomicadd` instructions. We illustrate the two approaches in Algorithm 1.

ALGORITHM 1: Computation of the loss gradient w.r.t, inputs, $\frac{\partial L}{\partial X}$, for a 2D convolution, expressed as a gather or scatter operation. We use the notation $x_{i,j}$ to denote element (i, j) of the tensor X.

Input: output gradient: $\frac{\partial L}{\partial Y}$, weight tensor: W
$\frac{\partial L}{\partial \mathbf{X}} = \mathbf{0}$;
for $i, j \in$ pixels **do**
  for $k, l \in$ footprint **do**
    $\frac{\partial L}{\partial x_{i,j}}$ += $W^T_{k,l} \cdot \frac{\partial L}{\partial y_{i+k,j+l}}$; `// gather`
    $\frac{\partial L}{\partial x_{i+k,j+l}}$ += $W_{k,l} \cdot \frac{\partial L}{\partial y_{i,j}}$; `// scatter`

10. Scene Credits

Mori Knob from Yasotoshi Mori (CC BY 3.0). Cerberus model used with permission from NVIDIA. Damicornis, Saxophone, and Jackson models courtesy of the Smithsonian 3D repository [61], (CC0). Spot model (public domain) by Keenan Crane. NeRD datasets (moldGoldCape, ethiopianHead) (Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International). The NeRF and NeRFactor datasets contain renders from modified blender models located on blendswap.com: chair by 1DInc (CC-0), drums by bryanajones (CC-BY), ficus by Herberhold (CC-0), hotdog by erickfree (CC-0), lego by Heinzelnisse (CC-BY-NC), materials by elbrujodelatribu (CC-0), mic by up3d.de (CC-0), ship by gregzaal (CC-BY-SA). Probes from Poly Haven [76] (CC0) and the probes provided in the NeRFactor dataset which are modified from the probes (CC0) shipped with Blender. DTU scans from the DTU MVS dataset [28].

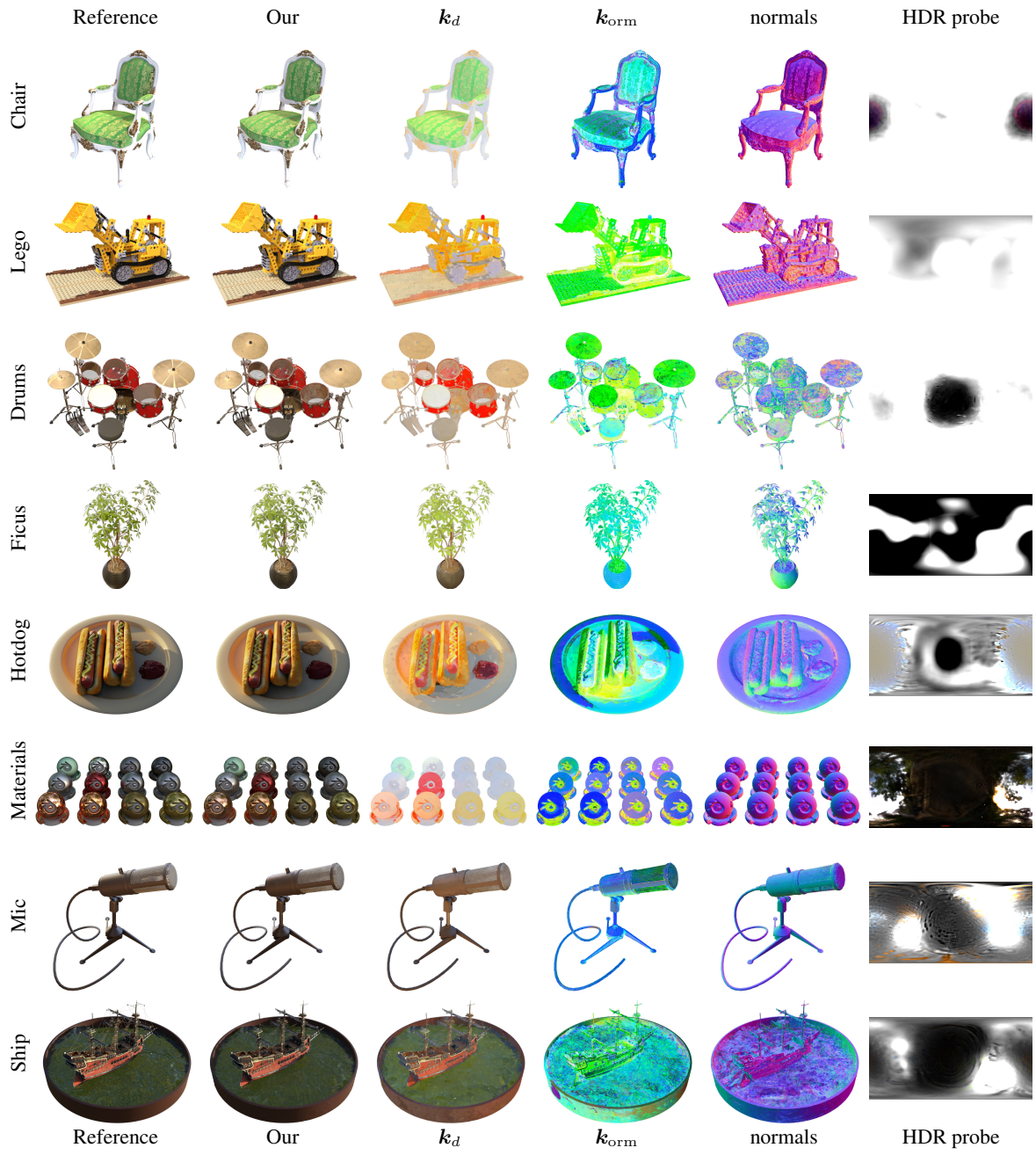


Figure 30. Our decomposition results on the NeRF Synthetic dataset. We show our rendered models alongside the material textures: diffuse ($\boldsymbol{k}_d$), roughness/metalness ($\boldsymbol{k}_{\text{orm}}$), the normals, and the extracted lighting.