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

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

1. Introduction

In the following, we supplement the paper with additional results, ablations and implementation details. In Section 2 we present novel use cases for our method: automatic level of detail creation from images, and appearance aware model extraction. In Section 3, we present additional results, including an evaluation of geometric quality, additional per-scene statistics and visual examples. Finally, in Section 4 we provide implementation details, including efficient split-sum pre-integration, regularizer terms and losses.

2. Novel applications

2.1. Level-of-detail From Images

Inspired by a recent work in appearance-driven automatic 3D model simplification [5], 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 1 and in the supplemental video.

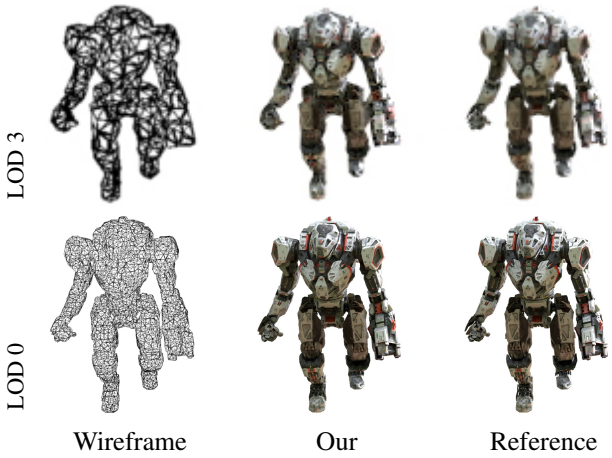


Figure 1. Automatic LOD example: We generated 256 views of an object (with masks & poses) from a path tracer in two resolutions: 1024×1024 and 128×128 pixels, then reconstructed the mesh, materials and lighting to approximate LOD creation. **Top**: LOD level optimized to look good at a resolution of 128×128 pixels with 3k triangles. **Bottom**: LOD level optimized to look good at a resolution of 1024×1024 pixels with 63k triangles.

2.2. Appearance-Aware NeRF 3D Model Extractor

We devise a way to extract 3D models from neural radiance fields [12] (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 [16] (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 2.

2.3. 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 3 we show two ex-

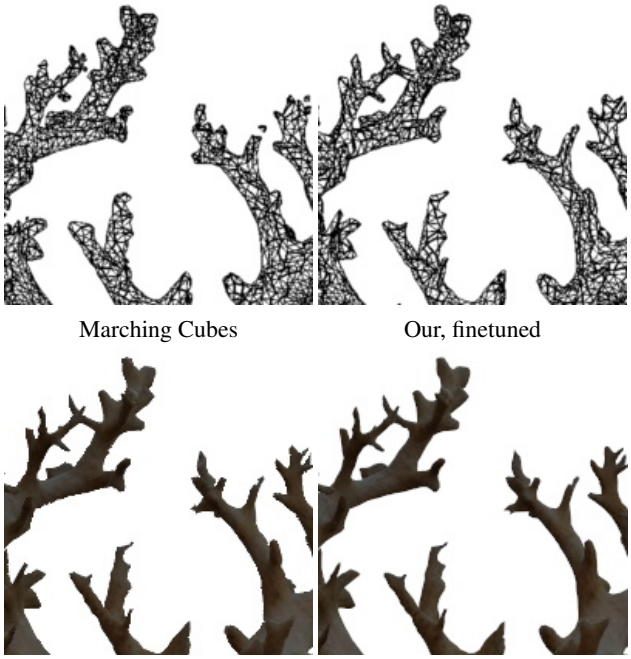


Figure 2. Appearance-aware NeRF 3D model extraction. We show insets of the silhouette quality before and after our optimization pass, alongside insets of the reference and our rendered result.

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 1. 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.

amples from the Smithsonian 3D repository [16]. Note the quality in both the extracted materials and geometric detail.

Figure 3. DMTet can accurately capture topology, even in challenging scenarios. To illustrate this, we show two examples from the Smithsonian 3D repository [16], 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.

3. Results

3.1. Scene Editing and Simulation

This section supplements Section 4.1 in the main paper. In Table 1 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 4, where we relight the Ficus scene with four different light probes, comparing to the results of NeRFactor [23]. Figure 5 shows a visual example of material separation with albedo, $\boldsymbol{k}_d$, and normals, $\mathbf{n}$. In Figure 19 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.

Figure 4. Relighting quality for a scene from the NeRFactor dataset, with our examples relit using Blender, and NeRFactor results generated using the public code.

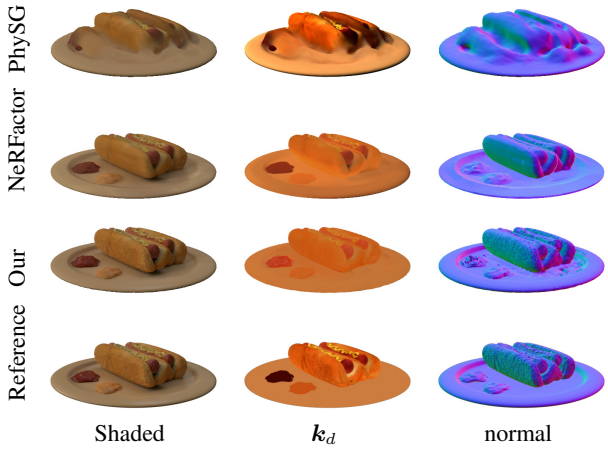


Figure 5. Extracted materials for a scene from the NeRFactor dataset. We directly compare albedo $\boldsymbol{k}_d$ and normals $\mathbf{n}$ to the results of NeRFactor. Specular parameters are omitted as we use different BSDF models. All $\boldsymbol{k}_d$ images have been renormalized using the reference 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 2. 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 [12], with new measurements for PhySG and MipNeRF using their respective publicly available source code. We additionally report FLIP mean scores [2]. Note that the Hotdog outlier LPIPS score for NeRF is consistent with the original paper, but probably a bug.

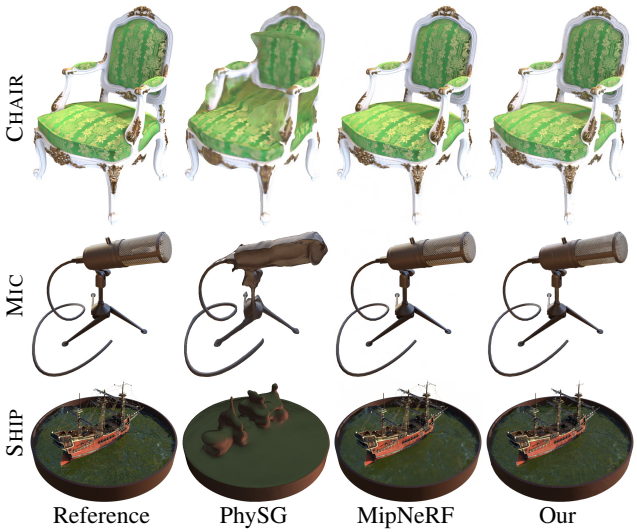


Figure 6. Visual quality examples from the NeRF realistic synthetic dataset comparing our method to PhySG and MipNeRF. PhySG struggles to accurately capture the complex geometry and spatially varying materials of the dataset.

3.2. View interpolation

This section supplements Section 4.2 in the main paper. In Table 2 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 6 shows a visual comparison to PhySG and MipNeRF for

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 3. 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.

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 3. 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 7 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 [19] (neural implicit representation). We provided masks at training for both approaches. We note that on this dataset, our method performs on par

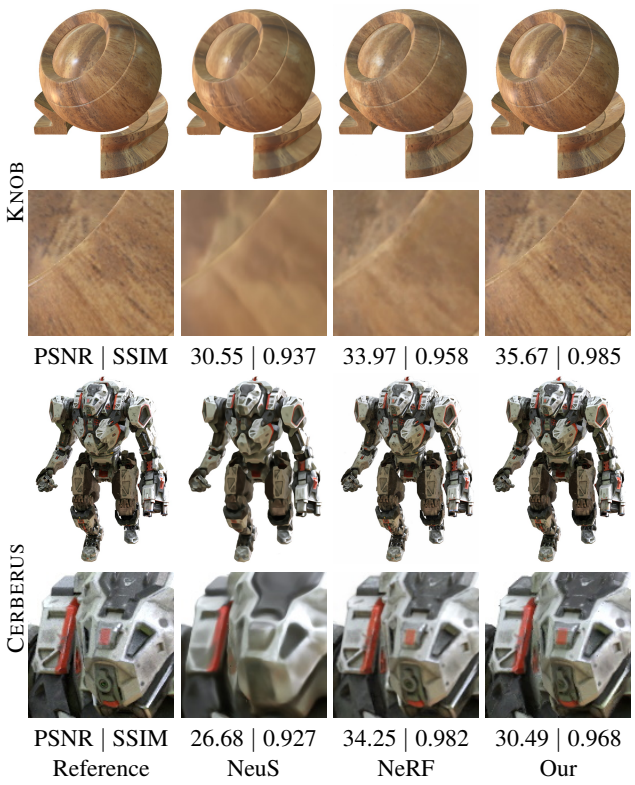


Figure 7. Visual quality examples on the synthetic KNOB and CERBERUS datasets. We observe slightly blurry results from NeuS.

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 4. 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.

with NeRF, and consistently produces results with greater detail and sharpness than NeuS.

3.3. 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

	KNOB	CERBERUS	DAMICORNIS
NeRF [12]	2.77e-01	9.08e-03	3.34e-03
NeuS [19]	2.04e-01	2.84e-02	5.84e-04
Our	1.87e-01	1.03e-02	4.66e-04

Figure 8. 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.

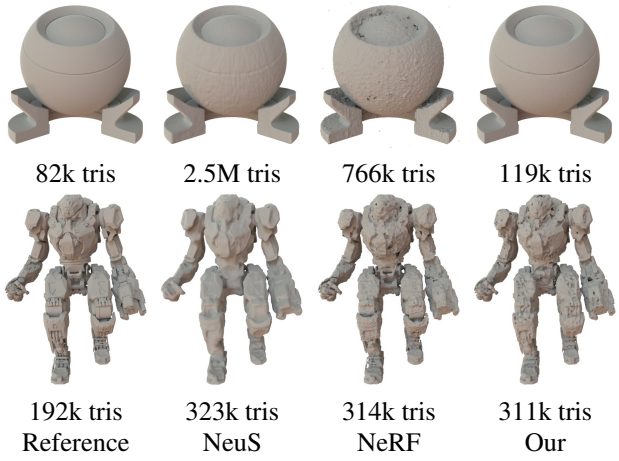


Figure 9. Extracted mesh quality visualization examples on the synthetic KNOB and CERBERUS datasets.

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 4 for completeness. When comparing with NeRF [12], we use pretrained checkpoints provided by JaxNeRF[1] [4], 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/ mask) which further utilizes coverage masks and regularizes density, trading some image quality for better geometric accuracy. We use the NGP-NeRF [13] 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.

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

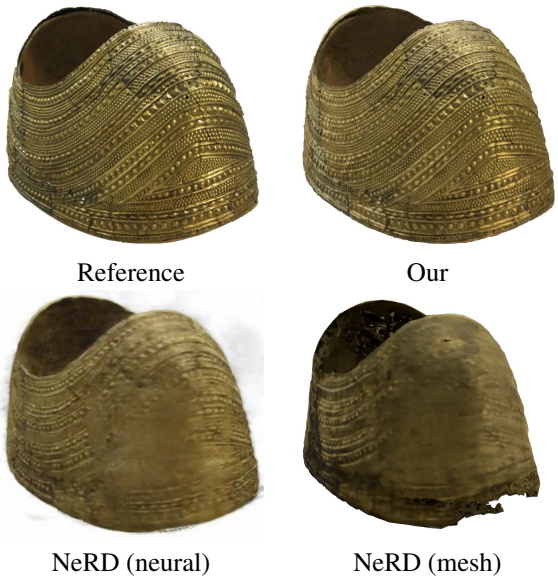


Figure 10. Example of the quality of the neural NeRD representation and their final generated mesh. Note the quality loss in both geometry and appearance (textures).

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.

In Figure 8, 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 7 and a visualization of the Lambertian shaded mesh is included in Figure 9.

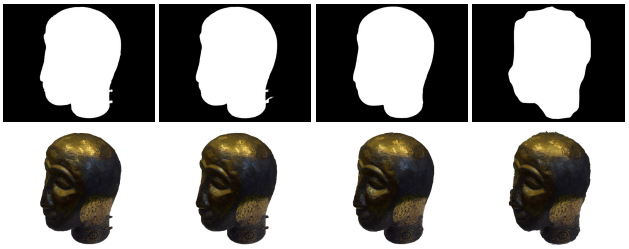
We additionally present an example of an output mesh generated by NeRD [3] in Figure 10. The impact of the mesh extraction step is notable, both to geometry and material quality. As we only have this single data point, with no means of accurately aligning the meshes for measuring geometric loss (NeRD does not provide source code), we will not provide metrics.

3.4. Quality of Segmentation Masks

Like many related works (e.g. NeRD [3], DVR [14], and IDR [21]) 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 14) and the NeRD

Figure 11. 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.

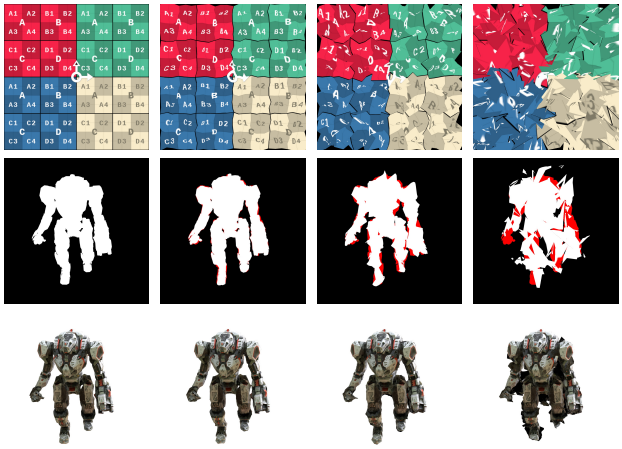


Manual	Photoshop	D2: PointRend	D2: M-RCNN
26.55 dB	25.94 dB	25.31 dB	22.83 dB

Figure 12. 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.

dataset (Figure 10) rely on manually annotated masks, with some frames containing large errors and inconsistencies, as shown in Figure 11. In Figure 12 we automatically generate segmentation masks of varying quality for a real-world dataset. We generate masks using two versions of Detectron2 [20], namely PointRend [9] 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 13, 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.



	$\sigma = 0$	$\sigma = 5$ 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 13. 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.

3.5. 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 [7], shows that our approach work less well than the recent neural implicit versions, such as NeuS [19], Unisurf [15], and IDR [21], 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 5, and visual examples of our results on three scans in Figure 14.

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

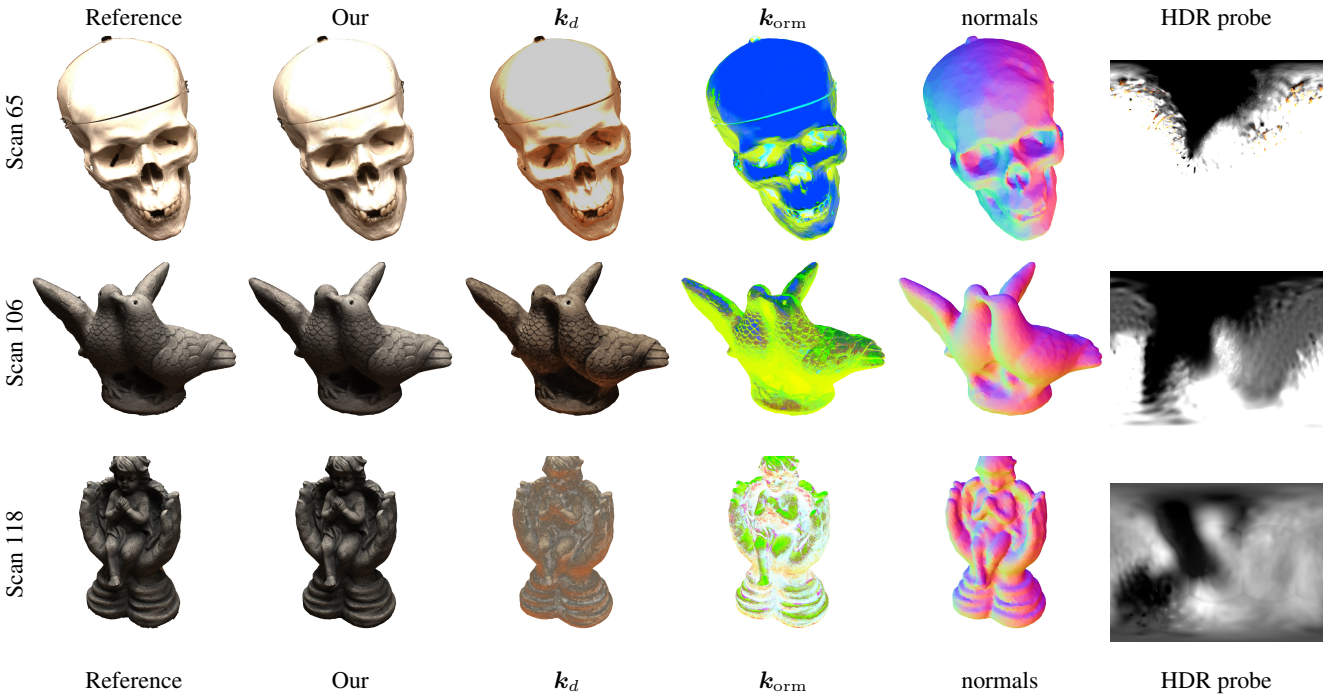


Figure 14. Our decomposition results on scan 65, 106, and 118 of the DTU MVS dataset [7]. Our model is trained on a reduced subset (49 of the 64 views) which has more consistent lighting across views, labelled by DVR [14]. However, we still penalize mask loss on the excluded views.

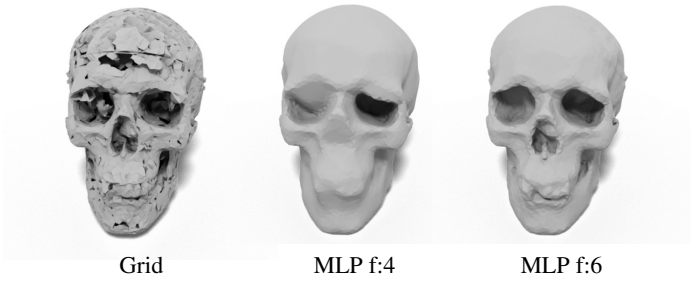


Figure 15. Comparing grid vs. MLP parametrizations of DMTet on scan 65 from the DTU MVS dataset [7]. Directly optimizing SDF values at grid vertices leads to a surface with high-frequency noise (left). In contrast, if we use an MLP to parametrize the SDF values, we can regularize the geometry, with smoothness controlled by the frequency of positional encoding. We use the positional encoding in NeRF [12] with frequency set to 4 (middle) and 6 (right) respectively.

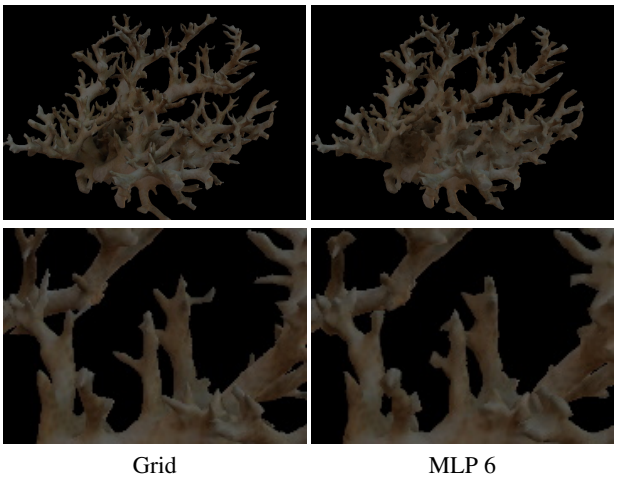


Figure 16. Comparing grid vs. MLP parametrization of DMTet on the synthetic DAMICORNIS dataset. Directly optimizing per-vertex SDF and offsets on a grid is faster to train, and better captures high-frequency geometric details than parametrizing DMTet with a network.

reconstructed shape can be controlled by the frequency of the positional encoding applied to the inputs of the MLP, as shown in Figure 15. 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 16, 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.

	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 5. Quantitative evaluation on the DTU dataset w/ mask. Chamfer distances are measured in the same way as NeuS [19], IDR [21], and DVR [14]. Results for NeRF, IDR and NeuS are taken from Table 1 in the NeuS paper [19], 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 [7] 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.

We use the same MLP as in DVR [14], which consists of five fully connected residual layers with 256 hidden features. In addition, we adopt the positional encoding in NeRF [12] and progressively fit the frequencies similar to SAPE [6]. 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 the rest during training such that:

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

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.

4. Implementation

4.1. 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 [23] 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 [8] 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.

4.2. 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 [5], 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 [18]:

$$\begin{aligned} \Gamma(x) &= \begin{cases} 12.92x & x \leq 0.0031308 \\ (1+a)x^{1/2.4} - a & x > 0.0031308 \end{cases} \quad (2) \\ a &= 0.055. \end{aligned}$$

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:

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

Material Regularizer As mentioned in the paper, we regularize material parameters using a smoothness loss similar to NeRFactor [23]. Assuming that $\boldsymbol{k}_d(\mathbf{x})$ denotes the $\boldsymbol{k}_d$

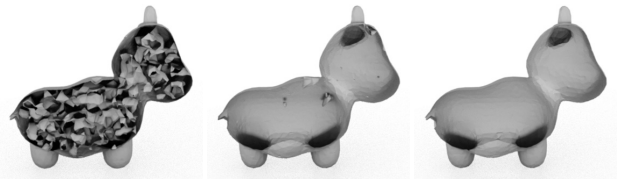


Figure 17. Cross sections of shapes optimized without regularization loss on SDF (left), with smoothness loss used by Liao et al. [11] (middle) and with our regularization loss (right). The random faces inside the object are removed by the regularization loss on SDF.

parameter at world space position $\mathbf{x}$ and ϵ is a random displacement vector, we define the regularizer as:

$$L_{\text{mat}} = \sum_{\mathbf{x}_{\text{surf}}} \left| \boldsymbol{k}_d \left(\mathbf{x}_{\text{surf}} \right) - \boldsymbol{k}_d \left(\mathbf{x}_{\text{surf}} + \epsilon \right) \right| . \tag{4}$$

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 [17] 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. [10] and use a Laplacian regularizer term given by

$$L_{\boldsymbol{\delta}} = \frac{1}{n} \sum_{i=1}^{n} \left\| \boldsymbol{\delta}_i - \boldsymbol{\delta}_i' \right\|^2 , \tag{5}$$

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 inside the object, as shown in Fig. 17, 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. [11] 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:

$$L_{\text{smooth}} = \sum_{i,j \in \mathbb{S}_e} |s_i - s_j|, \tag{6}$$

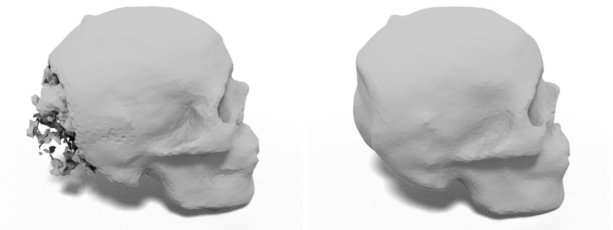


Figure 18. Reconstruction of scan 65 from the DTU MVS dataset [7] without (left) and with (right) the regularization loss based on visibility of faces. The regularization loss removes floaters behind the object that are not visible from the training views.

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`

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. 17.

In our DTU experiments, we use an additional regularization loss to removes the floaters behind the visible surface, as illustrated in Fig 18. 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.

4.3. 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.

5. 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 [16], (CC0). Spot model (public domain) by Keegan 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 [22] (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 [7].

References

[1] Waleed Abdulla. Mask R-CNN for Object Detection and Instance Segmentation on Keras and TensorFlow. https://github.com/matterport/Mask_RCNN, 2017. 6

[2] 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. 3

[3] 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. 5

[4] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. JaxNeRF: an Efficient JAX Implementation of NeRF, 2020. 5

[5] Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. Appearance-Driven Automatic 3D Model Simplification. In *Eurographics Symposium on Rendering*, 2021. 1, 8

[6] 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. 8

[7] 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. 6, 7, 8, 9, 10

[8] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, 2015. 8

[9] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. PointRend: Image Segmentation as Rendering. *arXiv:1912.08193*, 2019. 6

[10] 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. 9

[11] Yiyi Liao, Simon Donné, and Andreas Geiger. Deep Marching Cubes: Learning Explicit Surface Representations. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 9

[12] 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, 3, 5, 7, 8

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

[14] 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. 5, 7, 8

[15] 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. 6

[16] Smithsonian. Smithsonian 3D Digitization, 2018. https://3d.si.edu/. 1, 2, 10

[17] Olga Sorkine. Laplacian Mesh Processing. In *Eurographics 2005 - State of the Art Reports*, 2005. 9

[18] Michael Stokes, Matthew Anderson, Srinivasan Chandrasekar, and Ricardo Motta. A Standard Default Color Space for the Internet - sRGB, 1996. 8

[19] 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. 4, 5, 6, 8

[20] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 6

[21] 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. 5, 6, 8

[22] Greg Zaal. Poly Haven, 2021. https://polyhaven.com/hdris. 10

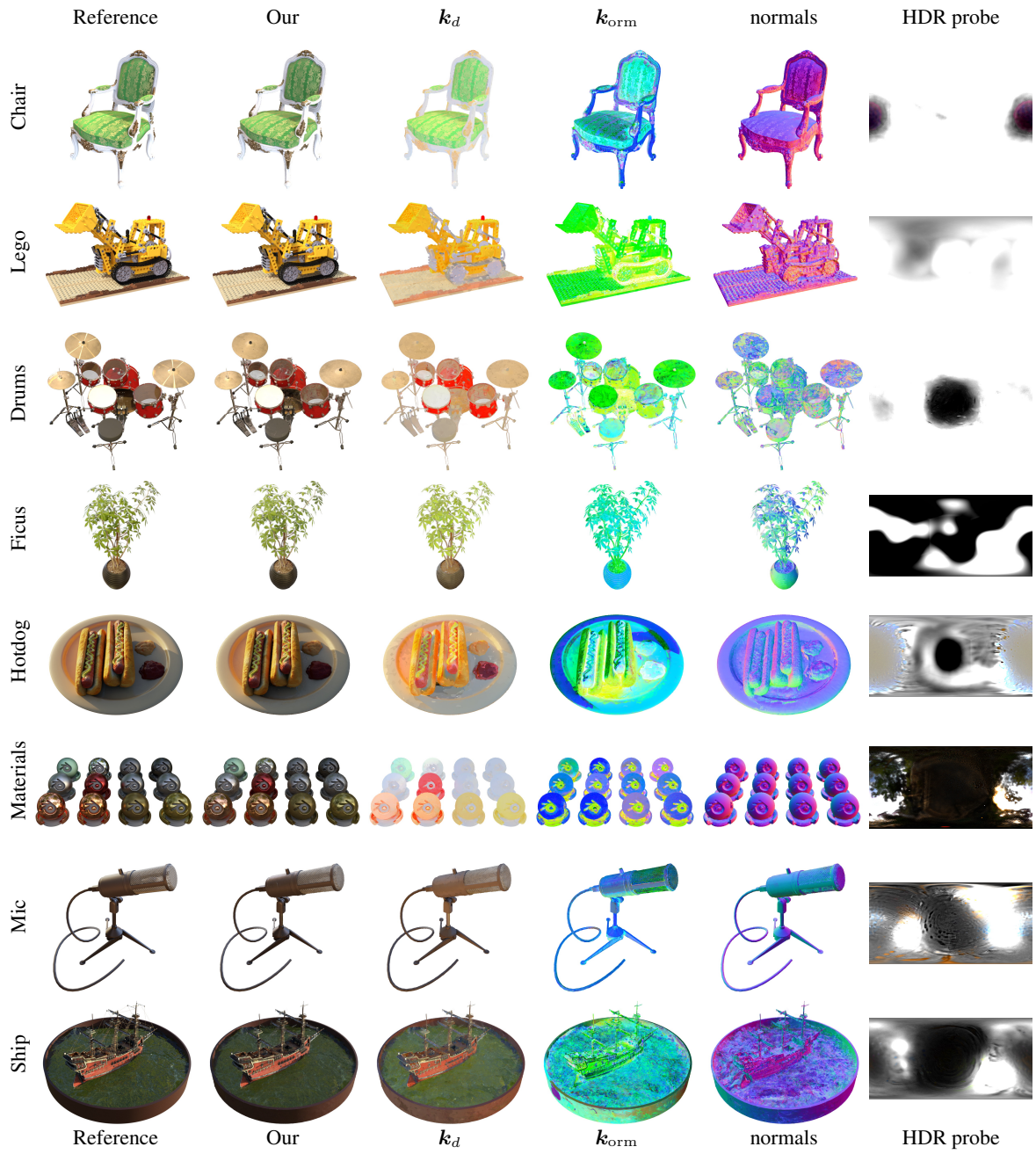


Figure 19. Our decomposition results on the NeRF Synthetic dataset. We show our rendered models alongside the material textures: diffuse ($\boldsymbol{k}_d$), roughness/metalness ($\boldsymbol{k}_{\text{orm}}$), the normals, and the extracted lighting.

[23] 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. 2, 8