1. Overview

In this supplementary material, we aim to provide extensive results, implementation details and further analysis that are not present on our main paper. We accompany this document with a video.

We divide this supplementary material in different sections, as follows:

- On Section 2, we provide exhaustive implementation details for our model design and training configuration, as well as our metrics.
- On Figures 1 to 3, we show diagrams of our model architectures.
- On Figures 4 and 5, we show statistics of our dataset.
- On Figure 6, we show results of our normals re-parameterization algorithm.
- On Figure 7, we show additional results for our delighting and relighting models.
- On Figures 8 and 9, we show additional results on the consistency of our method across multiple scanners.
- On Figure 10, we show additional comparisons with UMat [1] on scanned images.
- On Figure 11, we show additional comparisons with previous work on images captured with a smartphone.

2. Implementation Details

2.1. Delighting

Our delighting and relighting models are trained using a cycle-consistent GAN framework. In this section, we detail our design choices for the generator and discriminator architectures.

Generator Design. Following [1, 2], for the generator, we use a U-Net [3]. We specify the full model architecture and layer sizes on Figure 1. We use residual connections [4, 1, 5] in every convolutional block of the model. We use 1×1 convolutions on the skip connections. We use Group Normalization [6] (with 16 *groups* per layer) and SiLU [7] non-linearities throughout the model. Each convolutional block in the encoder is enhanced with a lightweight Linear Attention module [8], with 4 *attention heads*, each with a dimension of 32 hidden units. On the bottleneck, we use a lightweight *MobileViT* Transformer block [9], with 128 hidden dimensions for the self-attention and MLPs, 4 layers and a kernel size of 3. We use a Dropout rate of 0.2. We use transposed convolutions for upsampling. Every other implementation detail in the model (strides, bias, poolings) follow [3].

Discriminator Design. For the discriminators, we follow previous work on texture synthesis [10, 11] and use a 4-layer PatchGAN [12]. Please refer to the implementation in [11] for details.

2.2. SVBSDF Estimation

Our SVBSDF estimation model is trained using a GAN framework following. In this section, we detail our design choices for the generator and discriminator architecture, and their training.

Generator Design. Following [1, 2], for the generator, we use a U-Net [3]. We specify the full model architecture and layer sizes on Figure 2. We use residual connections [4, 1, 5] in every convolutional block of the model. We use 1×1 convolutions on the skip connections. Following [10, 1, 13], to maximally preserve the appearance and characteristics of every target map, we use a single decoder for each. We append a pixel-wise MLP to each decoder in the model, with Dropout regularization. We use Group Normalization [6] (with 16 *groups* per layer) and SiLU [7] non-linearities throughout the model. Each convolutional block in the encoder is enhanced with a lightweight Linear Attention module [8], with 4 *attention heads*, each with a dimension of 32 hidden units. On the bottleneck, we use a lightweight *MobileViT* Transformer block [9], with 128 hidden dimensions for the self-attention and MLPs, 4 layers and a kernel size of 3. We use a Dropout rate of 0.2. We use transposed convolutions for upsampling. For the normal map estimation, we introduce a final *tanh* non-linearity to its corresponding decoder. Every other implementation detail in the model (strides, bias, poolings) follow [3].

Discriminator Design. Following [1, 14], for the discriminator, we use a U-Net [3] model,. We specify the full model architecture and layer sizes on Figure 3. As in the generator, we use residual connections [4, 5, 1] in every convolutional block of the models. We use Spectral Normalization [15] and SiLU [7] non-linearities throughout the model. On the bottleneck, we use a lightweight *CBAM* attention block [16]. The single-scalar estimation of the discriminator $\mathcal{D}_{enc}$ is provided by a MLP with a similar architecture to the *CBAM* module. We initialize the entire model using orthogonal initialization [17]. We use transposed convolutions for upsampling. Every other implementation detail in the model (strides, bias, pooling) follow [3].

Model Training

Optimization. Following [1], we train the models using PyTorch [18] and TorchVision [19]. We leverage Kornia for data augmentation [20]. To accelerate the training process, we leverage mixed precision training and automatic gradient scaling [21], and train the whole model natively on GPU. Optimization is done using AdamW [22]. Following [14, 1], we use different learning rates for the generator lr = 0.0008 and the discriminator lr = 0.004 and a batch size of 46, betas= $(0.9, 0.99)$ and a weight decay of 0.000003. Both generators and discriminators are initialized using *orthogonal initialization* [17]. We use a cosine learning rate scheduler.

Loss Function. We use the following weights for the loss function: $\lambda_{\measuredangle} = 3, \lambda_{spec} = 1, \lambda_{rough} = 1, \lambda_{transmittance} = 0.33, \lambda_{opacity} = 0.5, \lambda_{adv} = 0.2, \lambda_{style} = 0.2, \lambda_{freq} = 0.25, \lambda_{cons} = 0.3$. For the style loss function, we use the *AlexNet* variant of LPIPS [23], as in [1, 24] and our delighting framework.

Data Augmentation. Training is done using random crops of 256×256 pixels. We perform random rescales uniformely on the 300, 1200 PPI range, with bilinear interpolation. We randomly rotate the SVBSDF on the $[-10, 10]$ angle range. We use the algorithm in [25] to correctly rotate normal map. On the HSV color space, we randomly change the properties of the model inputs. We randomly change the hue of the images, selected at random on the whole hue ranges. We also change the saturation, value and contrast of the input images with factors of $[0.84, 1.16]$, $[0.85, 1.15]$, $[0.9, 1.1]$, respectively, as specified on the Torchvision [19] `ColorJitter` implementation. With a probability of 0.25, we apply random Gaussian noise to the input images $\mu = 0, \sigma = 0.10$. With a probability of 0.2, we apply random erasing [26] to the input images. Half of the times, we apply a random Gaussian blur to the input images $\sigma \sim U(0.1, 5)$. Finally, with a probability of 0.5, we apply cutmix data augmentation to the discriminator, as in [14].

Model Evaluation. We use half precision for model evaluation, which makes the process faster and allows us to generate SVBSDFs of very large sizes.

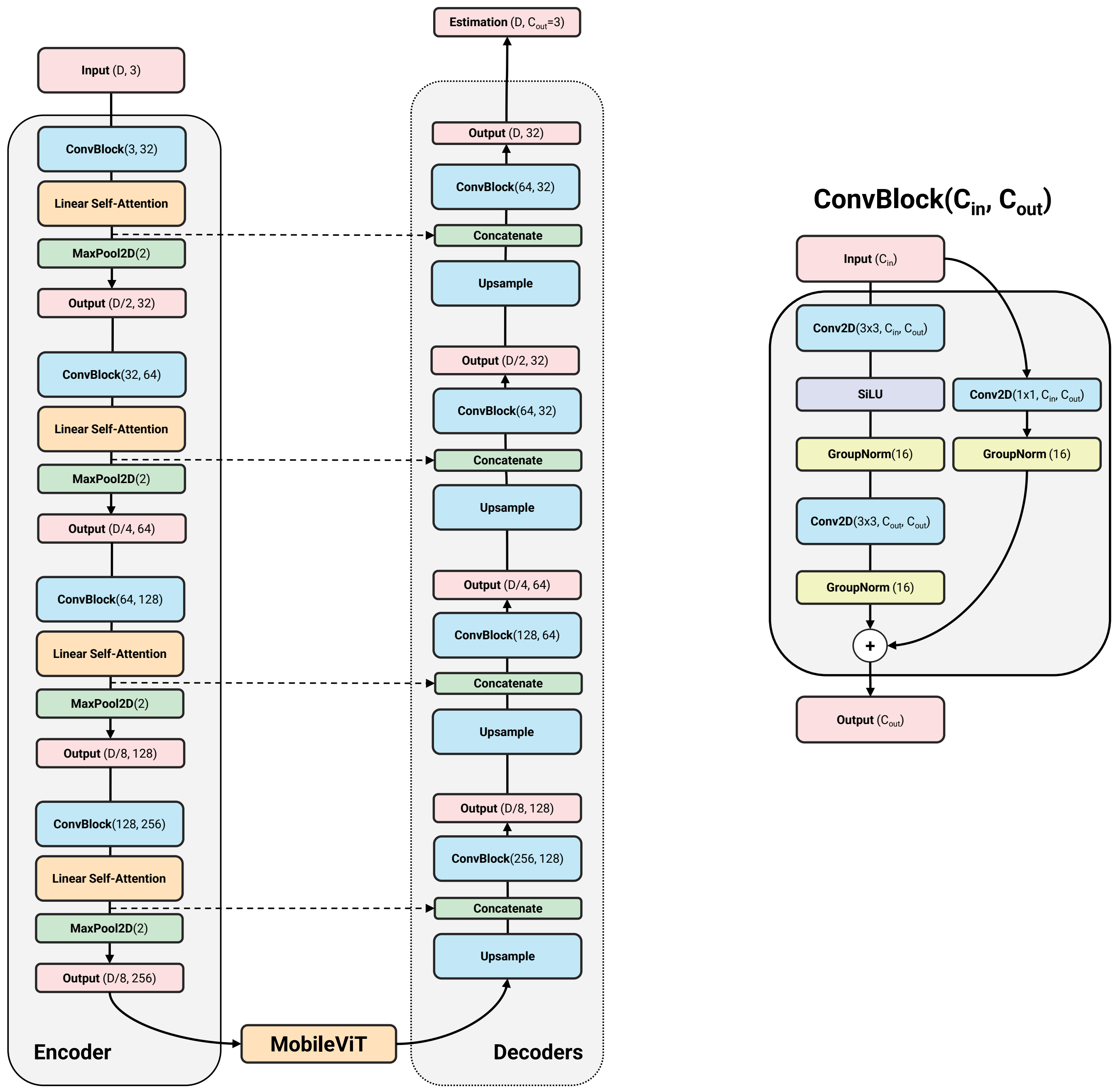

Fig. 1: *A full diagram of our generator, used in our* ***delighting and relighting*** *estimation networks, including layer sizes and output dimensions for each layer. Note that this model follows [1]. For Self-Attention, we leverage Linear Attention [8], we use a MobileVIT transformer on the bottleneck [9], Group Normalization [6] and SiLU [7] non-linearities, and residual connections in every convolutional block. In red, we show the input/output dimensions (spatial, channels) of each layer; in orange, we show attention modules; in blue, convolutional blocks and layers; in green, upsampling and concatenating operations; in yellow, normalization layers; and in purple, regularizations and non-linearities.*

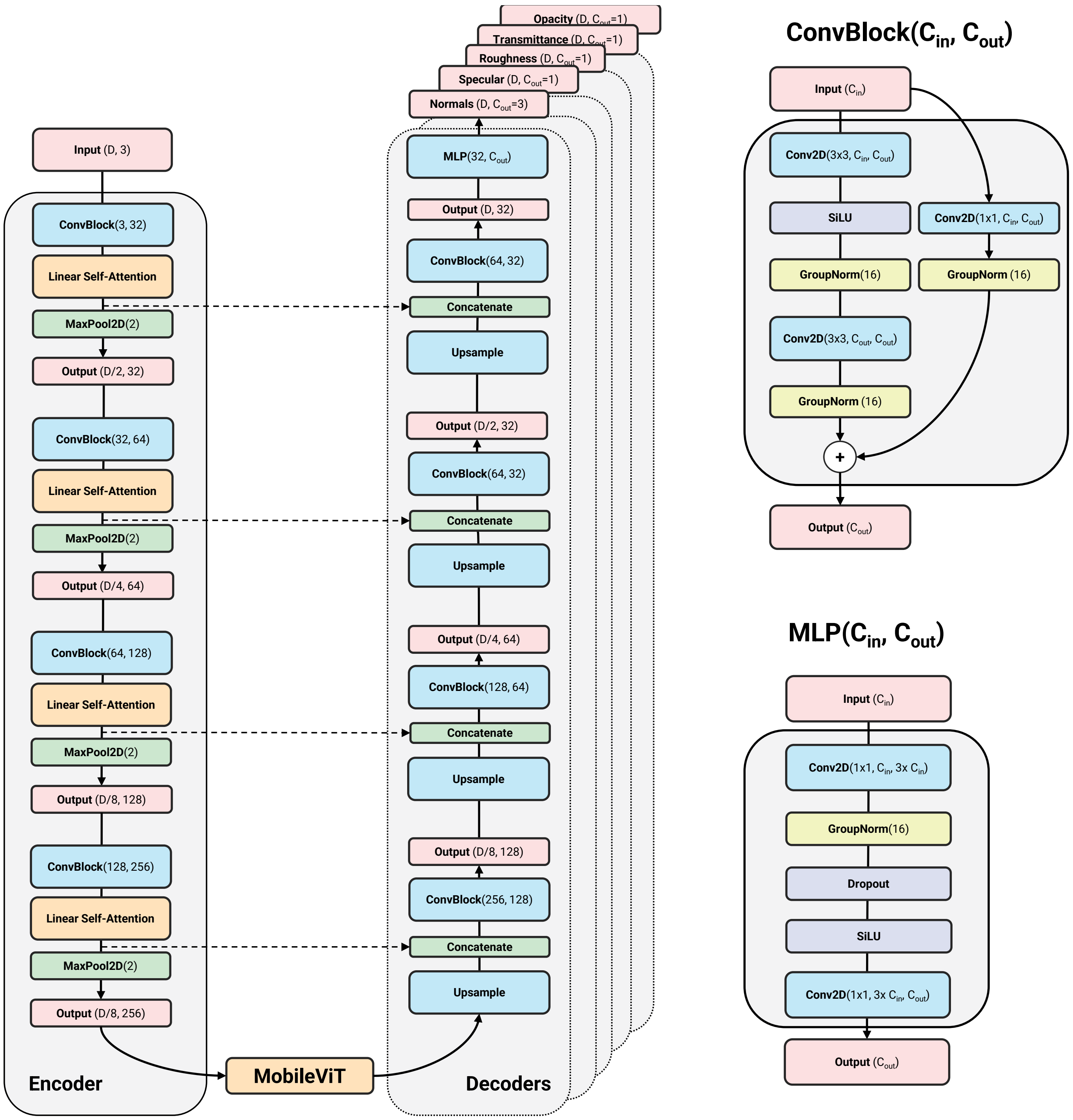

Fig. 2: *A full diagram of our generator, used in our* ***SVBSDF estimation network****, including layer sizes and output dimensions for each layer. Note that this model follows [1]. For Self-Attention, we leverage Linear Attention [8], we use a MobileVIT transformer on the bottleneck [9], Group Normalization [6] and SiLU [7] non-linearities, one decoder per output map and residual connections in every convolutional block. In red, we show the input/output dimensions (spatial, channels) of each layer; in orange, we show attention modules; in blue, convolutional blocks and layers; in green, upsampling and concatenating operations; in yellow, normalization layers; and in purple, regularizations and non-linearities.*

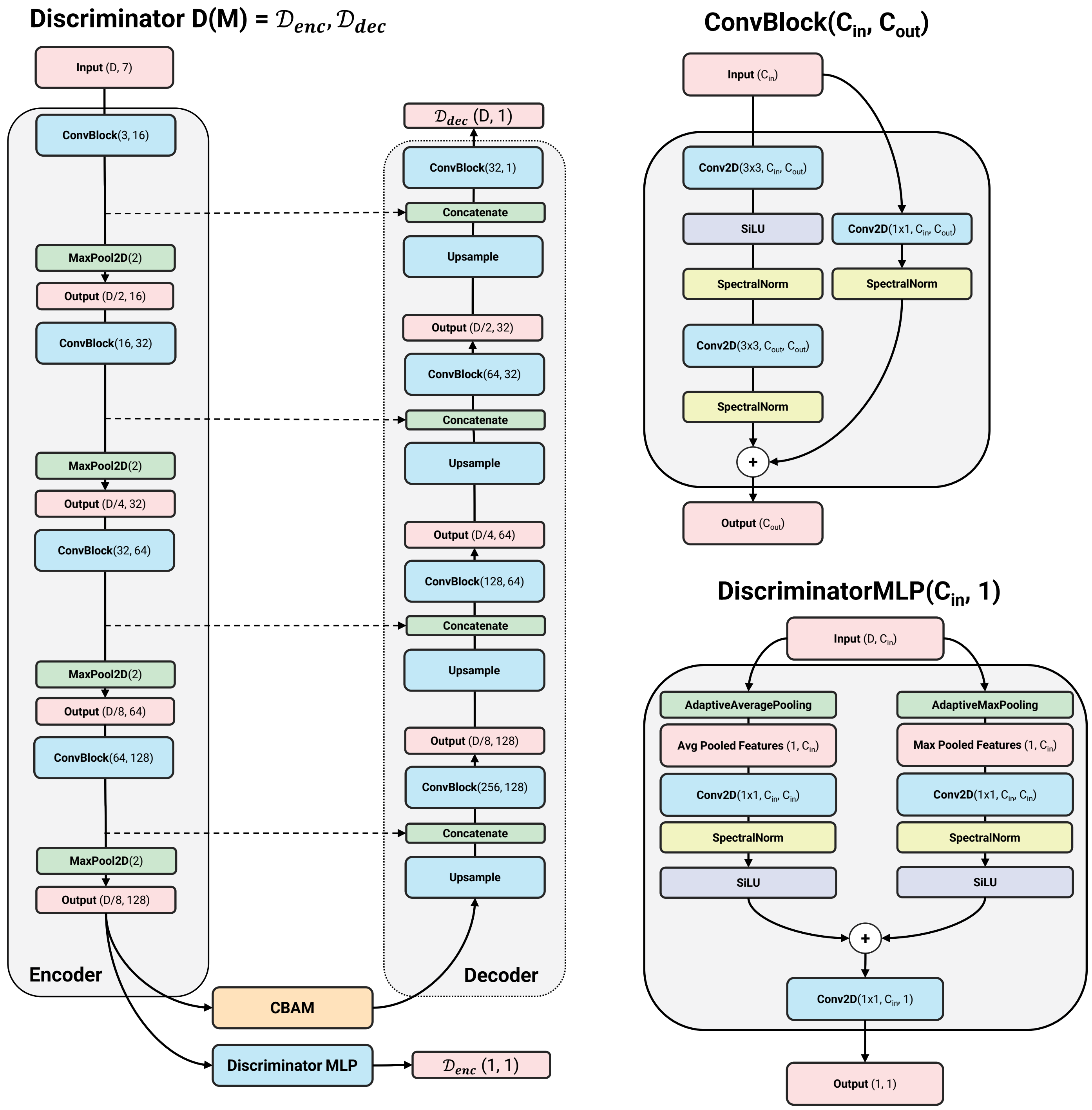

Fig. 3: *A full diagram of our U-Net residual discriminator, used in our SVBSDF estimation network, including layer sizes and output dimensions for each layer. Note that this model follows [1]. We use a CBAM [16] module on the bottleneck and Spectral Normalization [15] throughout the network and residual connections in every convolutional block. In red, we show the input/output dimensions of each layer; in orange, we show attention modules; in blue, convolutional blocks and layers; in green, upsampling and concatenating operations; in yellow, normalization layers; and in purple, non-linearities.*

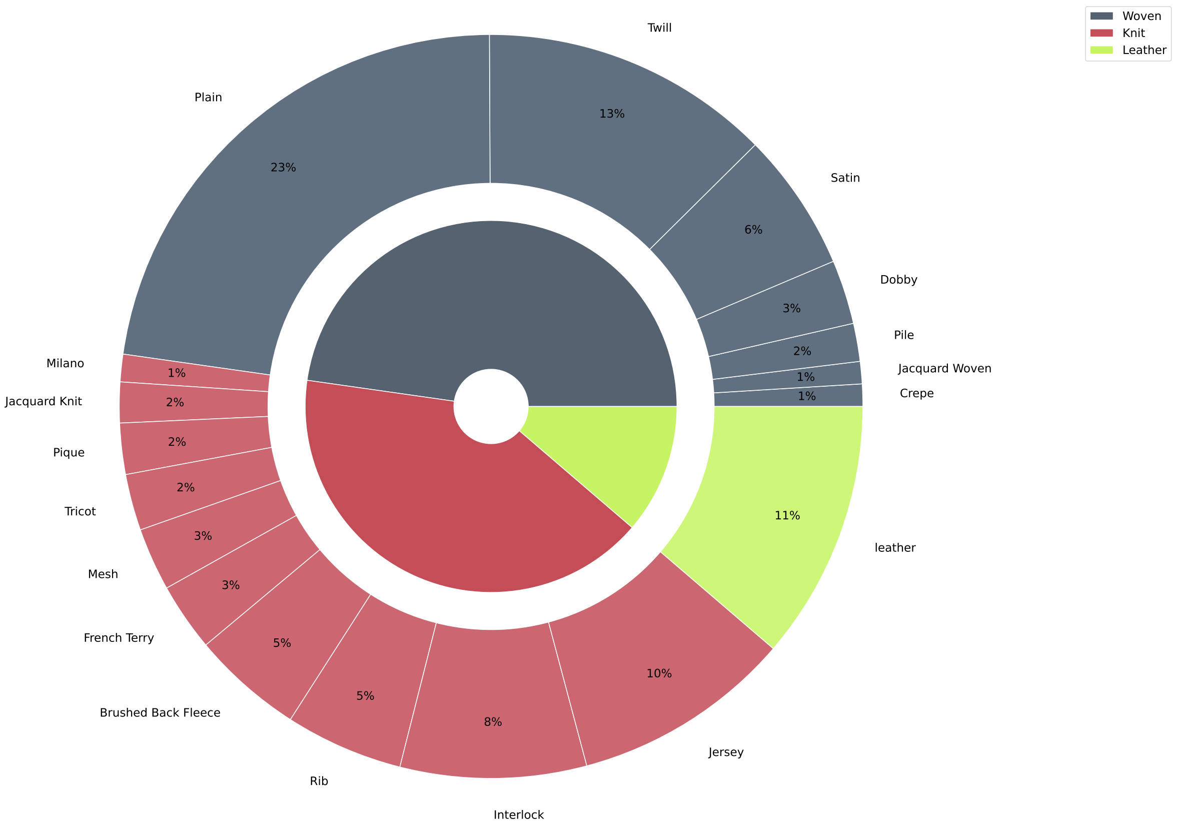

Fig. 4: *Distribution of material categories in our training set.*

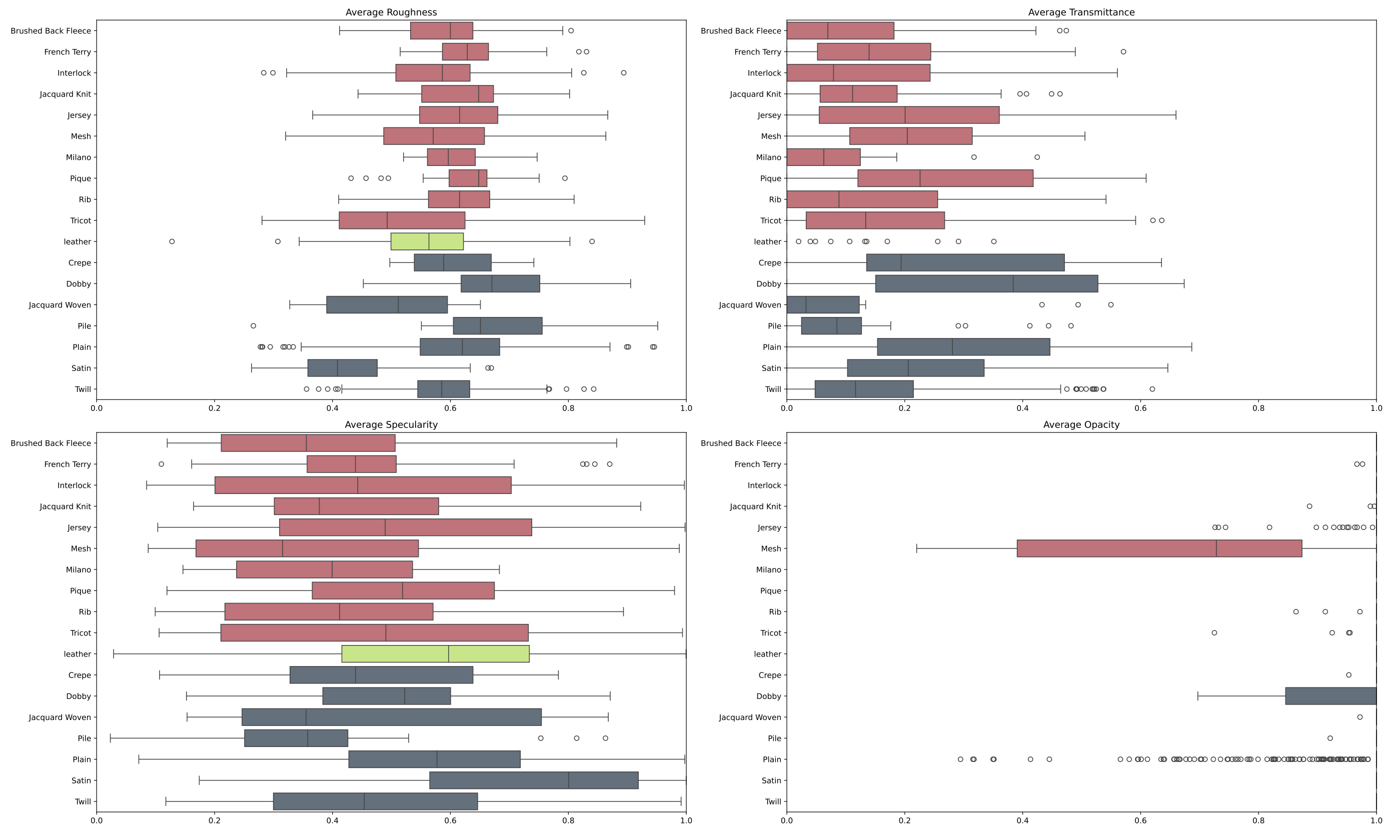

Fig. 5: *Average value of each BSDF map for each category in our training set. Notice how opacity is not evenly distributed.*

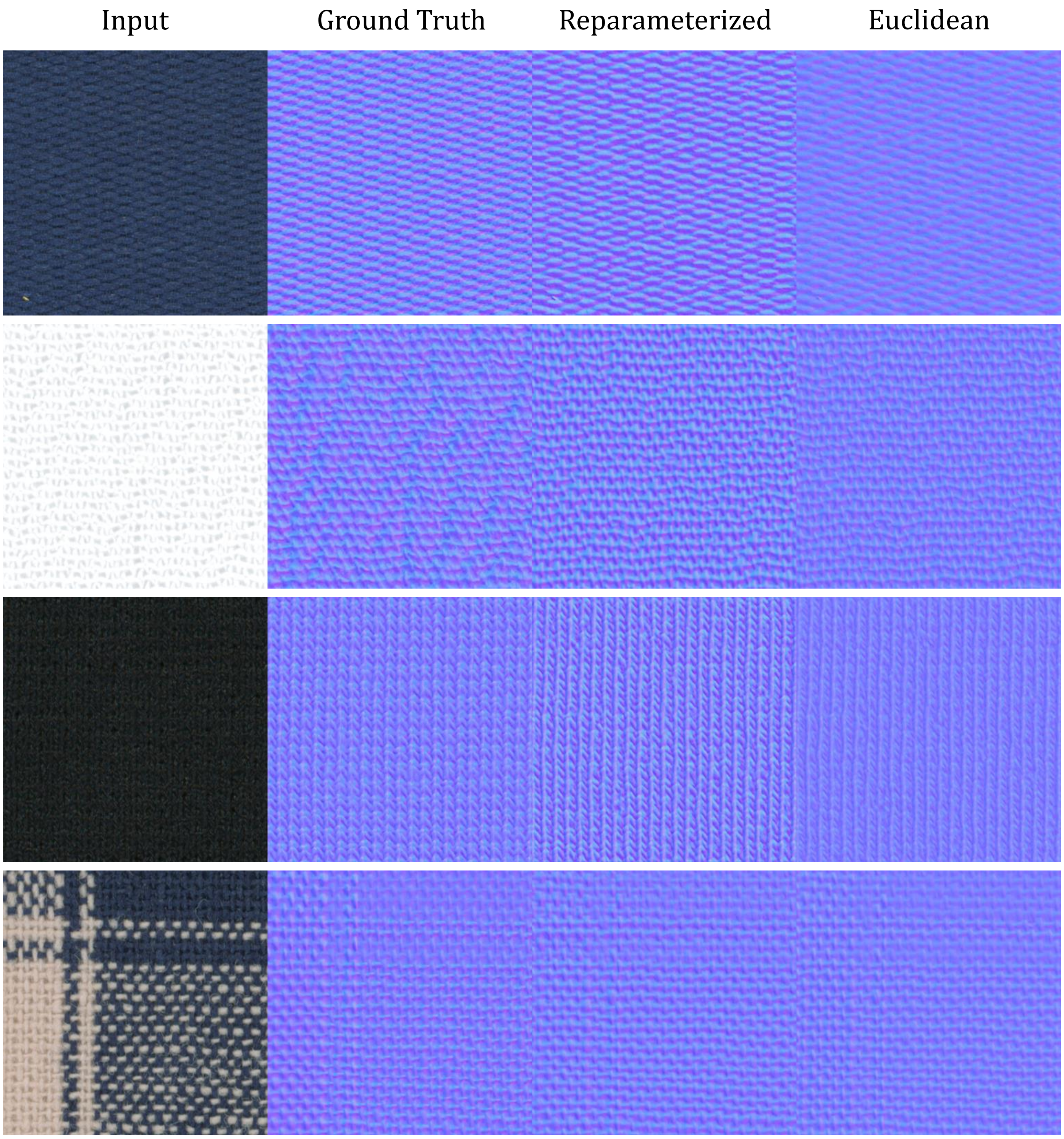

Fig. 6: *Qualitative results of our normals reparameterization. From left to right, we show input images, ground truth normals, our estimated normals using elliptical grid mapping and polar θ, ϕ coordinates (our final model), and a baseline euclidean xyz coordinates model. As shown, our normals reparameterization allows for higher quality, sharper normal maps.*

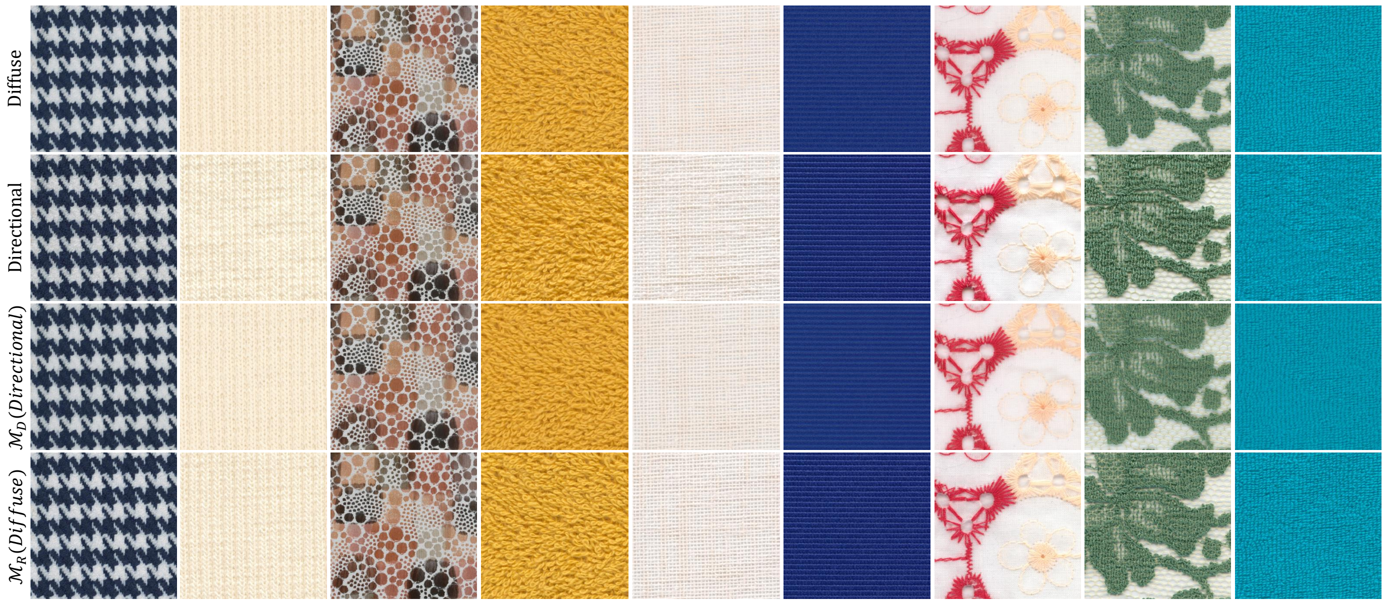

Fig. 7: *Additional qualitative results of our material delighting framework. On the first two rows, we show images captured with flatbed scanners under diffuse (top) and directional (bottom) illumination. We use those as input to our delighting $\mathcal{M}_D$ and relighting $\mathcal{M}_R$ models, respectively, for which we show the results on the bottom rows.*

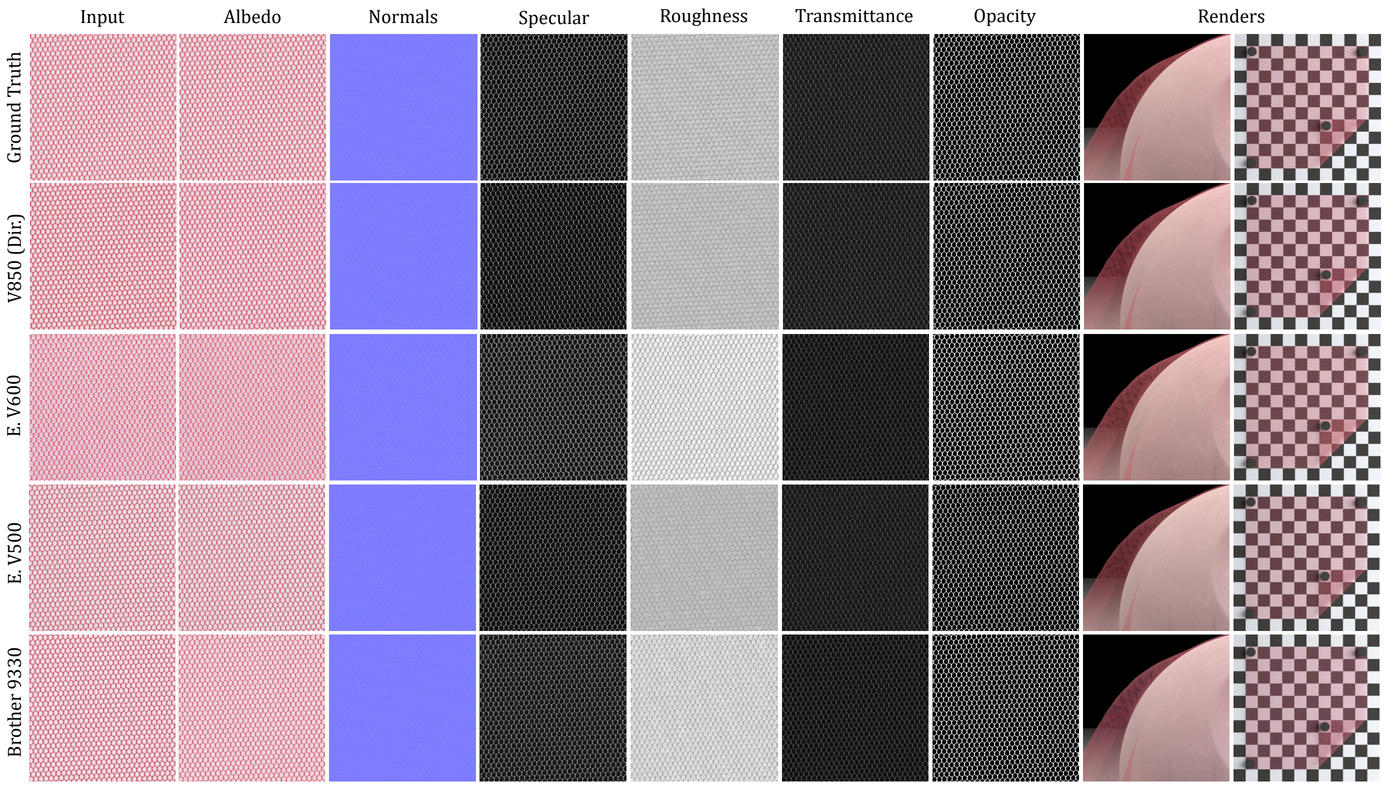

Fig. 8: *Qualitative comparison between several flatbed scanners for the same material. Note that the images are not exactly pixel-wise coherent across scanners.*

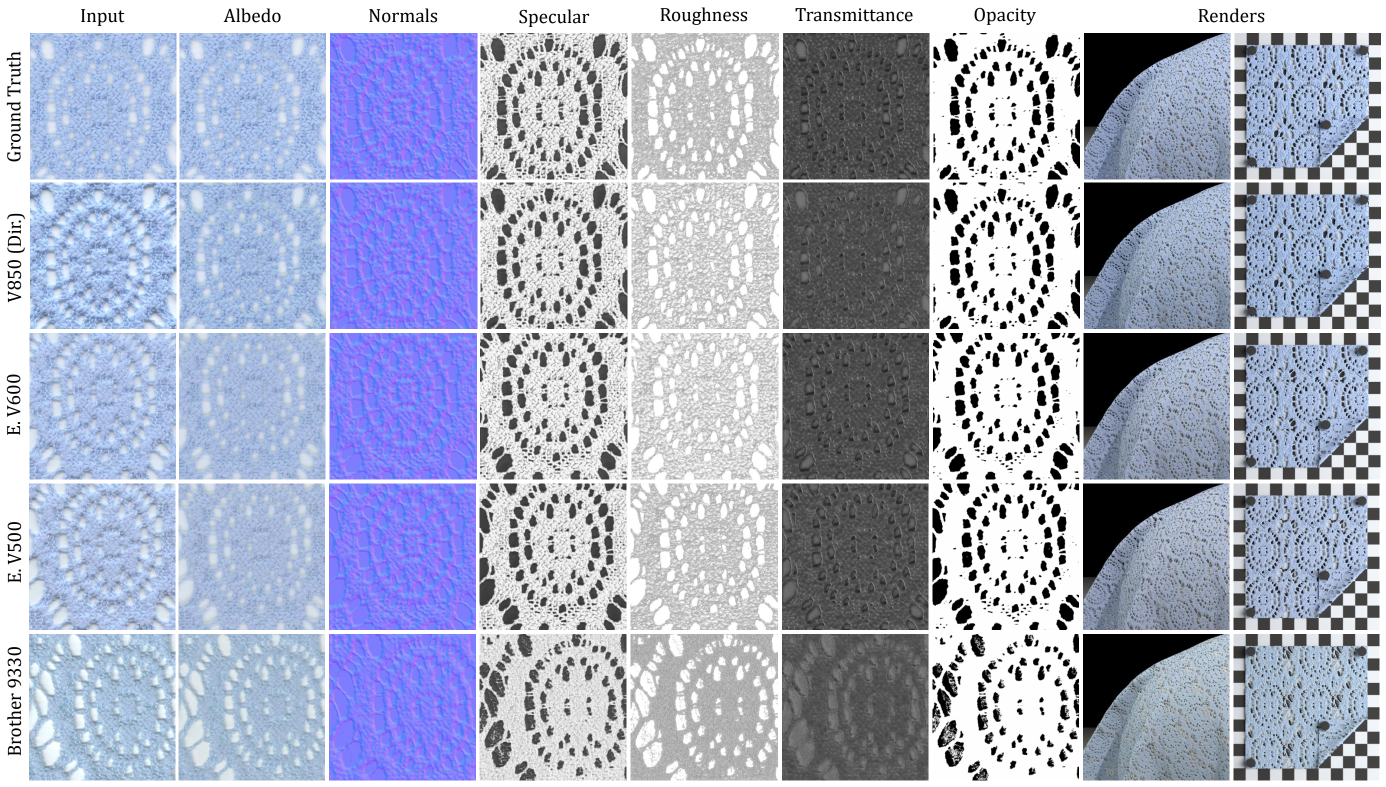

Fig. 9: *Qualitative comparison between several flatbed scanners for the same material. Note that the images are not exactly pixel-wise coherent across scanners.*

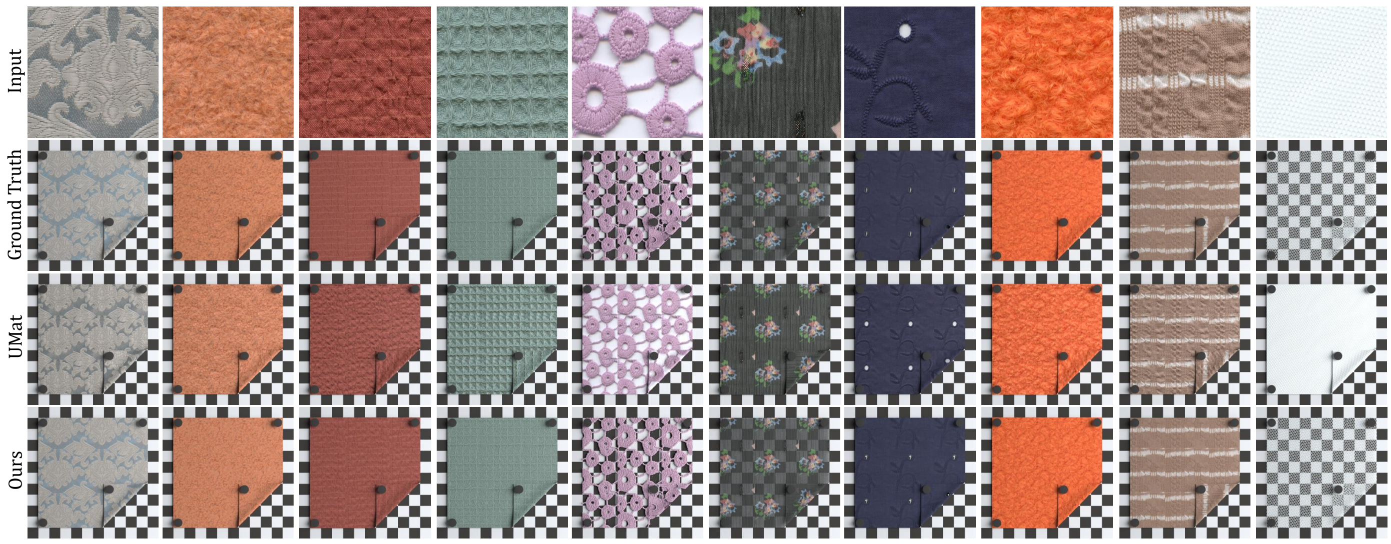

Fig. 10: *Additional comparisons with [1]. We show the input image (top row), and renders using the ground truth materials (captured with a gonioreflectometer), the estimation of [1] and ours, on the second, third and fourth rows, respectively. Best viewed in color on a screen.*

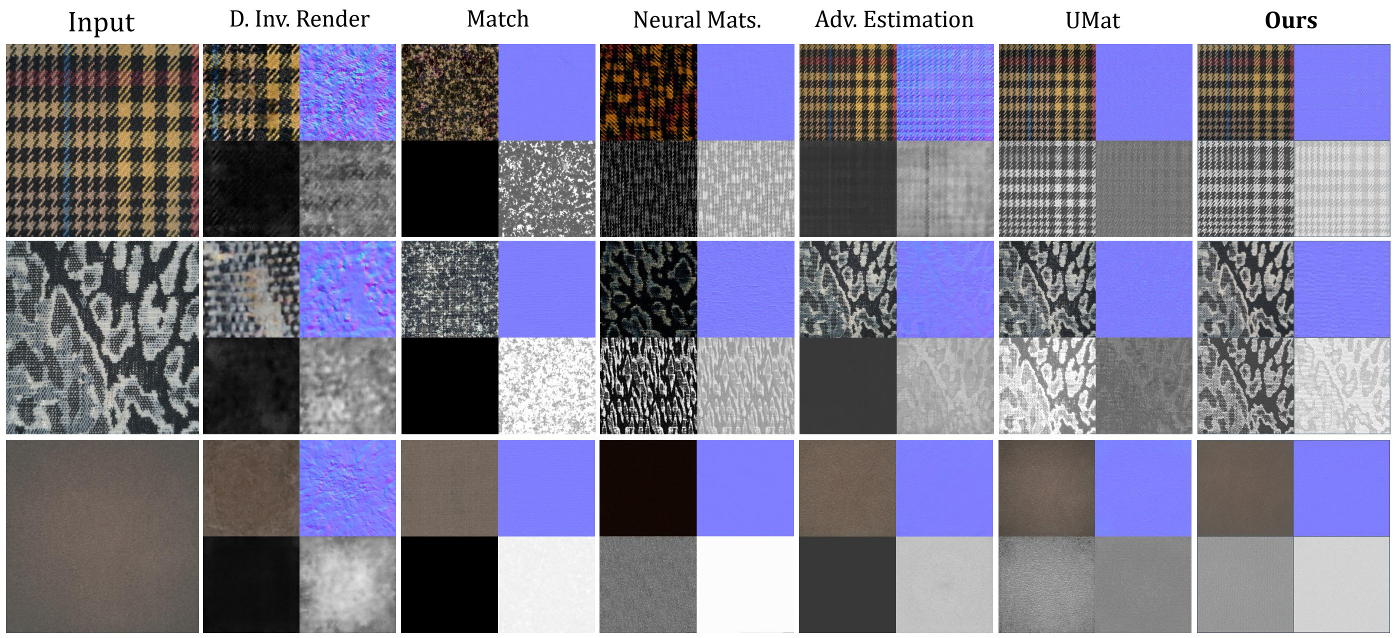

Fig. 11: *Additional comparisons of our method with previous work on images captured with a smartphone, using flash illumination, at different levels of resolution. From left to right, we show input images, and the results of Deep Inverse Rendering [27], Match [28], Neural Materials [29], Adversarial SVBRDF Estimation [30], UMat [1], and ours. Note that we only show the four reflectance maps used by every method: albedo, normals, specular and roughness.*

References

[1] Rodriguez-Pardo, C, Dominguez-Elvira, H, Pascual-Hernandez, D, Garces, E. Umat: Uncertainty-aware single image high resolution material capture. Proceedings of the IEEE/CVF International Conference on Computer Vision 2023;.

[2] Rodriguez-Pardo, C, Kazatzis, K, Lopez-Moreno, J, Garces, E. Neubtf: Neural fields for btf encoding and transfer. Computers & Graphics 2023;.

[3] Ronneberger, O, Fischer, P, Brox, T. U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer; 2015, p. 234–241.

[4] He, K, Zhang, X, Ren, S, Sun, J. Deep residual learning for image recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2016, p. 770–778.

[5] Deschaintre, V, Lin, Y, Ghosh, A. Deep polarization imaging for 3d shape and svbrdf acquisition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021, p. 15567–15576.

[6] Wu, Y, He, K. Group normalization. In: Proceedings of the European Conference on Computer Vision (ECCV). 2018, p. 3–19.

[7] Elfwing, S, Uchibe, E, Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Neural Networks 2018;107:3–11.

[8] Wang, S, Li, BZ, Khabsa, M, Fang, H, Ma, H. Linformer: Self-attention with linear complexity. arXiv preprint arXiv:200604768 2020;.

[9] Mehta, S, Rastegari, M. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. arXiv preprint arXiv:211002178 2021;.

[10] Rodriguez-Pardo, C, Garces, E. SeamlessGAN: Self-Supervised Synthesis of Tileable Texture Maps. IEEE Transactions on Visualization and Computer Graphics 2022;.

[11] Zhou, Y, Zhu, Z, Bai, X, Lischinski, D, Cohen-Or, D, Huang, H. Non-stationary texture synthesis by adversarial expansion. ACM Transactions on Graphics (TOG) 2018;37(4):1–13.

[12] Isola, P, Zhu, JY, Zhou, T, Efros, AA. Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2017, p. 1125–1134.

[13] Garces, E, Arellano, V, Rodriguez-Pardo, C, Pascual-Hernandez, D, Suja, S, Lopez-Moreno, J. Towards material digitization with a dual-scale optical system. ACM Transactions on Graphics (TOG) 2023;42(4):1–13.

[14] Schonfeld, E, Schiele, B, Khoreva, A. A u-net based discriminator for generative adversarial networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, p. 8207–8216.

[15] Miyato, T, Kataoka, T, Koyama, M, Yoshida, Y. Spectral normalization for generative adversarial networks. arXiv preprint arXiv:180205957 2018;.

[16] Woo, S, Park, J, Lee, JY, Kweon, IS. Cbam: Convolutional block attention module. In: Proceedings of the European Conference on Computer Vision (ECCV). 2018, p. 3–19.

[17] Hu, W, Xiao, L, Pennington, J. Provable benefit of orthogonal initialization in optimizing deep linear networks. arXiv preprint arXiv:200105992 2020;.

[18] Paszke, A, Gross, S, Chintala, S, Chanan, G, Yang, E, DeVito, Z, et al. Automatic differentiation in pytorch 2017;.

[19] Marcel, S, Rodriguez, Y. Torchvision the machine-vision package of torch. In: Proceedings of the 18th ACM international conference on Multimedia. 2010, p. 1485–1488.

[20] Riba, E, Mishkin, D, Ponsa, D, Rublee, E, Bradski, G. Kornia: an open source differentiable computer vision library for pytorch. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 2020, p. 3674–3683.

[21] Micikevicius, P, Narang, S, Alben, J, Diamos, G, Elsen, E, Garcia, D, et al. Mixed precision training. arXiv preprint arXiv:171003740 2017;.

[22] Loshchilov, I, Hutter, F. Decoupled weight decay regularization. arXiv preprint arXiv:171105101 2017;.

[23] Zhang, R, Isola, P, Efros, AA, Shechtman, E, Wang, O. The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2018, p. 586–595.

[24] Rodriguez-Pardo, C, Kazatzis, K, Lopez-Moreno, J, Garces, E. Neubtf: Neural fields for btf encoding and transfer. Computers & Graphics 2023;.

[25] Rodriguez-Pardo, C, Garces, E. Neural photometry-guided visual attribute transfer. IEEE Transactions on Visualization and Computer Graphics 2021;.

[26] Zhong, Z, Zheng, L, Kang, G, Li, S, Yang, Y. Random erasing data augmentation. In: Proceedings of the AAAI Conference on Artificial Intelligence; vol. 34. 2020, p. 13001–13008.

[27] Gao, D, Li, X, Dong, Y, Peers, P, Xu, K, Tong, X. Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images. ACM Transactions on Graphics (ToG) 2019;38(4):134:1–134:15.

[28] Shi, L, Li, B, Hašan, M, Sunkavalli, K, Boubekeur, T, Mech, R, et al. Match: differentiable material graphs for procedural material capture. ACM Transactions on Graphics (TOG) 2020;.

[29] Henzler, P, Deschaintre, V, Mitra, NJ, Ritschel, T. Generative modelling of brdf textures from flash images. ACM Transactions on Graphics (Proc SIGGRAPH Asia) 2021;40(6).

[30] Zhou, X, Kalantari, NK. Adversarial single-image svbrdf estimation with hybrid training. In: Computer Graphics Forum; vol. 40. Wiley Online Library; 2021, p. 315–325.