Computer Vision and Image Understanding
journal homepage: www.elsevier.com

Physics-based Shading Reconstruction for Intrinsic Image Decomposition Supplementary Material

Anil S. Baslamisli[a,**], Yang Liu[b], Sezer Karaoglu[b], Theo Gevers[a,b]

[a] *University of Amsterdam, Science Park 904, 1098XH Amsterdam, the Netherlands*
[b] *3DUniversum, Science Park 400, 1098XH Amsterdam, the Netherlands*

ABSTRACT

This document provides the implementation and training details together with a workflow of the proposed algorithm. Finally, a recoloring application is demonstrated.

© 2021 Elsevier Ltd. All rights reserved.

1. Workflow

We illustrate the steps of the proposed method in Figure 1. (1.) We calculate the albedo gradients to identify true color (reflectance) changes. If the albedo gradient map shows no significant changes in a local neighborhood, it manifests a homogeneously colored patch. A homogeneously (single) colored patch means that the only source causing pixel values to change is the shading component. (2.) Therefore, we calculate image gradients only for homogeneously colored patches. Then, we use an off-the-shelf algorithm to compute the global least squares reconstruction of the shading map from its shading gradient fields. This process generates a sparse shading map, where the albedo changes are masked out. (3.) Using an optimization framework, a shading smoothness constraint is employed to fill in the gaps (i.e. the masked out albedo changes) based on neighboring pixel information. This process generates a dense shading map. The dense shading map, filled with the smoothness constraint, is mostly blurry, suffers from scale problems and lacking crisp geometry changes. (4.) Therefore, the final dense shading map is integrated into a deep learning framework to further refine it and also to predict the reflectance image to achieve full intrinsic image decomposition. Therefore, Steps 1 and 2 are computed directly from the corresponding *RGB* image in a learning-free manner. Step 3 involves an optimization process using a single constraint, and is also learning-free. Step 4 is based on an end-to-end deep CNN model trained using supervised learning.

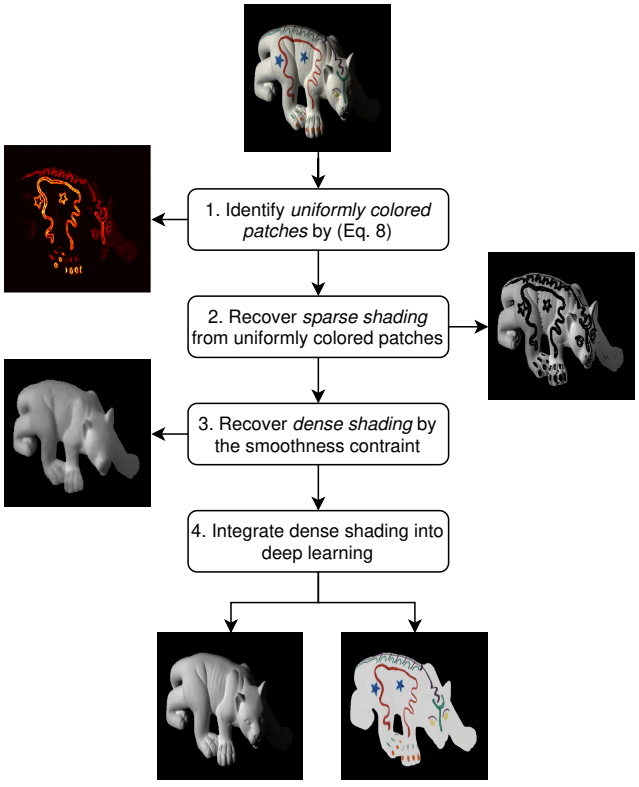

Fig. 1: Workflow of the proposed method. Steps 1 and 2 are computed directly from the corresponding *RGB* image in a learning-free manner. Step 3 involves an optimization process using a single constraint, also learning-free. Step 4 is based on an end-to-end deep CNN model trained using supervised learning.

**Corresponding author:
e-mail: a.s.baslamisli@uva.nl (Anil S. Baslamisli)

2. CNN Model and Training Details

2.1. Network

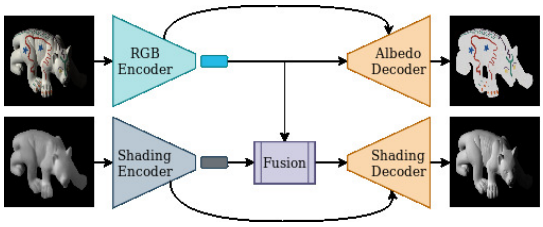

Fig. 2: Proposed model. *RGB* guides the shading estimation during the fusion phase using a $1x1$ convolution and a contextual attention module (Yu et al., 2018). The shading decoder receives the shading encoder features through skip connections not to be affected by high resolution *RGB* color features. The albedo decoder only receives *RGB* features through skip connections.

We design a CNN model such that the *RGB* image only refines the initial shading estimation, and it is not directly involved in the reconstruction phase to avoid color leakage. The model is provided in Figure 2.

Encoders: Encoder blocks use strided convolution layers for downsampling (4 times). Each convolution is followed by residual blocks (He et al., 2016). The *RGB* encoder uses 4 consecutive residual blocks, while the shading encoder uses 1 block after each strided convolution. Only the last block of the shading encoder has 4 consecutive residual blocks with different dilation rates. An encoder block is illustrated in Figure 3 with the residual block in Figure 4.

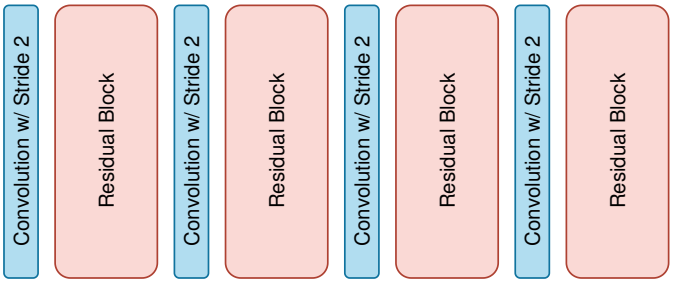

Fig. 3: An encoder block for feature extraction. Strided convolution layers are used for downsampling, followed by residual blocks.

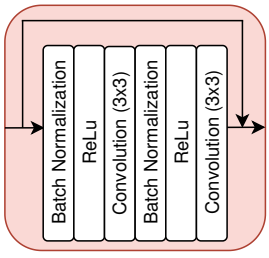

Fig. 4: A residual block.

Fusion: The last layers of the *RGB* encoder and the shading encoder are fused with a $1x1$ convolution and a contextual attention module to create a bottleneck such that the related *RGB* features can properly guide the shading estimation. The fusion block is illustrated in Figure 5. Before going into the contextual attention module, *RGB* features are first fed to a $1x1$ convolution kernel for dimension reduction. As a result, the *RGB* features are fused with the shading features (1) as a (learnable) weighted combination using a $1x1$ convolution, and (2) by the contextual attention module. Then, those two strategies are combined by concatenation and fed through *only* the shading decoder. The contextual attention module, introduced by Yu et al. (2018), learns where to use feature information from known background patches to generate missing patches for the image inpainting task. We adopt their module to our problem such that the shading features uses feature information from the *RGB* features. It is useful, because we show that in a homogeneously colored patch, the only source causing pixel values to change is the shading component, i.e. $\Delta I = \Delta S$. Therefore, in those regions, the shading map and the *RGB* image are highly correlated. Thus, the shading features are guided by the *RGB* features.

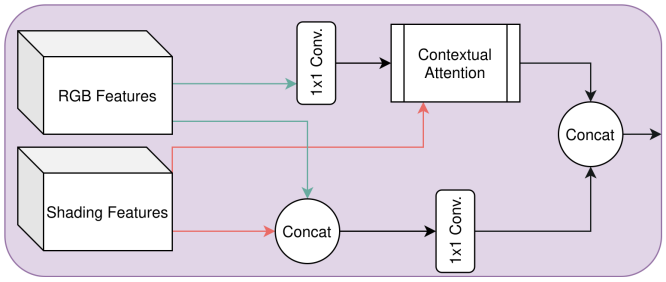

Fig. 5: The Fusion Module for the *RGB* features to guide the shading features.

Decoders: The fusion output is fed to the shading decoder, while the albedo decoder takes *RGB* encoder's last layer as input. Both decoders share the same structure, which is illustrated in Figure 6 with a convolutional block in Figure 7. Encoder features are passed through a *Conv Block* sequence. Then, the feature maps are (bilinearly) up-sampled and concatenated with their encoder counterpart by skip connections (Mao et al., 2011). The process is repeated 4 times to reach the final resolution. Shading decoder only receives shading encoder features through skip connections not to be affected by high resolution color features. The albedo decoder only receives *RGB* features through skip connections. Since the *RGB* encoder has 4 Residual Blocks before a downsampling, we use the last block for skip connections.

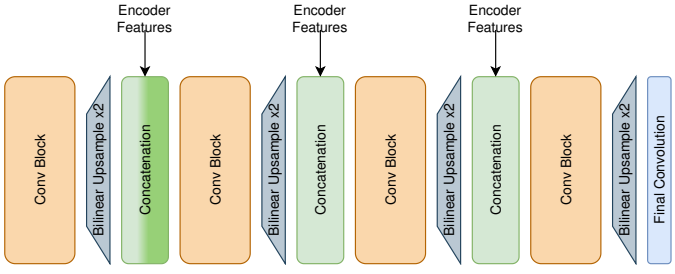

Fig. 6: A decoder block for feature reconstruction.

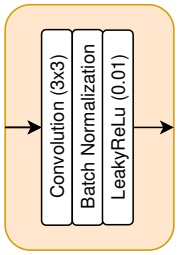

Fig. 7: A Convolutional block.

2.2. Loss Functions

The loss functions used to train the intrinsic images are as follows:

$$\mathcal{L}_{Albedo} = \lambda_{A1}\ \mathcal{L}_{pixel} + \lambda_{A2}\ \mathcal{L}_{gradient} + \lambda_{A3}\ \mathcal{L}_{dssim} + \lambda_{A4}\ \mathcal{L}_{perceptual}\ , \tag{1}$$

$$\mathcal{L}_{Shading} = \lambda_{S1}\ \mathcal{L}_{pixel} + \lambda_{S2}\ \mathcal{L}_{gradient} + \lambda_{S3}\ \mathcal{L}_{dssim}\ , \tag{2}$$

$$\mathcal{L}_{Total} = \lambda_{A}\ \mathcal{L}_{Albedo} + \lambda_{S}\ \mathcal{L}_{Shading} + \lambda_{I}\ \mathcal{L}_{Image}\ , \tag{3}$$

where $\mathcal{L}_{pixel}$ is the pixel-wise reconstruction loss, $\mathcal{L}_{gradient}$ denotes the gradient-wise reconstruction loss, $\mathcal{L}_{dssim}$ assesses the structural dissimilarity, $\mathcal{L}_{perceptual}$ measures the reconstruction distance in several feature spaces of a pre-trained network, $\mathcal{L}_{Image}$ is the image formation loss. Table 1 provides the values of the λs as the weighting factors of the loss functions. The individual loss functions are defined as follows. Let $\hat{I}$ be the ground-truth intrinsic image and I be the estimation of the network. Then, mean squared error (MSE) is defined as follows:

$$MSE(\hat{I}, I) = \frac{1}{N}\sum_{p,c} \|\hat{I} - I\|_2^2\ , \tag{4}$$

where c is the color channel index, p denotes the pixel coordinate, and N is the total number of valid pixels. For the albedo estimations $c \in \{R, G, B\}$, whereas the shading is gray-scale (single channel). Scale-invariant MSE loss first scales (with α) the estimation I, then compares its MSE with the ground-truth $\hat{I}$ as follows:

$$SMSE(\hat{I}, I) = MSE(\alpha I, \hat{I})\ , \tag{5}$$

$$\alpha = argmin\ MSE(\alpha I, \hat{I})\ . \tag{6}$$

Following the common practice, a weighted combination of MSE and SMSE are used for the pixel-wise reconstruction loss as follows:

$$\mathcal{L}_{pixel}(\hat{I}, I) = 0.95 \times SMSE(\hat{I}, I) + 0.05 \times MSE(\hat{I}, I)\ . \tag{7}$$

Then, the gradient-wise reconstruction loss is defined as follows:

$$\mathcal{L}_{gradient}(\hat{I}, I) = MSE(\nabla\hat{I}_x, \nabla I_x) + MSE(\nabla\hat{I}_y, \nabla I_y)\ , \tag{8}$$

where ∇ denotes the gradient operation over a pixel which is the difference between an adjacent and the current pixel. The loss is computed both horizontally (x) and vertically (y). The structural dissimilarity is derived from the structural similarity index (SSIM) by Wang et al. (2004) to quantify image reconstruction quality degradation as follows:

$$\mathcal{L}_{dssim}(\hat{I}, I) = \frac{1 - SSIM(\hat{I}, I)}{2}\ , \tag{9}$$

where the SSIM is calculated using the implementation provided by Tensorflow (Abadi et al., 2015). We only modify the filter size for the Gaussian kernel from 11×11 to 7×7. As the training takes place on object centered images with the size of 256×256, we intuitively use a smaller sized kernel. The perceptual loss measures the distance in the feature spaces of a 16-layer VGG network (Simonyan and Zisserman, 2015) trained on the ImageNet (Deng et al., 2009) dataset, denoted as ϕ, as follows:

$$\mathcal{L}_{perceptual}(\hat{I}, I) = \sum_{i} \|\phi(\hat{I}) - \phi(I)\|_2^2\ . \tag{10}$$

We use the feature maps obtained by the first three blocks of the VGG-16: *conv1_1, conv1_2, conv2_1, conv2_2, conv3_1, conv3_2* and *conv3_3*. Finally, the image formation loss is used to force that the estimated reflectance (R) and shading (S) images should reconstruct the original *RGB* image as follows:

$$\mathcal{L}_{Image} = MSE((R \times S),\ RGB)\ . \tag{11}$$

Table 1: Weight values for the loss functions.

λ_{A1}	λ_{A2}	λ_{A3}	λ_{A4}	λ_{S1}	λ_{S2}	λ_{S3}	λ_{A}	λ_{S}	λ_{I}
2.0	1.0	1.0	0.1	2.0	1.0	1.0	1.0	1.0	0.1

2.3. Training and Implementation Details

The input and output image sizes are fixed to 256×256 pixels. Images that are not 265×256 are resized to this resolution by bilinear interpolation. We do not apply any pre-processing step and use 8-bit images for both as inputs and outputs. Tensorflow framework is utilized for the experiments (Abadi et al., 2015). Convolution weights are initialized by using He initialization (He et al., 2015) without any weight decay. The batch size is fixed to 8 images and the experiments do not include any data augmentation process. Adam Optimizer (Kingma and Ba, 2014) is utilized with an initial learning rate of 0.000512. An exponential decay is applied to the learning rate such that the learning rate is lowered every 6000 iterations (approximately one epoch) with a base of 0.94. The models are trained approximately 35 epochs. The training takes around 2.5 days on a single NVIDIA GeForce GTX TITAN X GPU.

2.4. Feature Maps

Finally, we provide the details of the feature maps. For simplicity, D is for dilation rate, S is for stride size and C is for number of feature maps (channels). All convolutions use 3×3 kernels, except the ones utilized in the fusion module, which use 1×1. The *RGB* encoder is designed to have more features maps than the shading encoder, because it needs to disentangle both albedo and shading cues.

2.4.1. RGB Encoder:

S2C64 - S1C64 - S1C64 - S1C64 - S1C64 - S1C64 - S1C64 - S1C64 - S1C64 - S2C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S2C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S1C128 - S2C256 - S1C256 - S1C256 - S1C256 - S1C256 - S1C256 - S1C256 - S1C256 - S1C256.

2.4.2. Shading Encoder:

S2C32 - S1C32 - S1C32 - S2C32 - S1C32 - S1C32 - S2C32 - S1C32 - S1C32 - S2C32 - D2S1C32 - D2S1C32 - D4S1C32 - D4S1C32 - D8S1C32 - D8S1C32 - S1C32 - S1C32.

2.4.3. Albedo Decoder:

S1C128 - resize (2×) - concatenate via skip connection - S1C128 - resize (2×) - concatenate via skip connection - S1C128 - resize (2×) - concatenate via skip connection - S1C128 - resize (2×) - S1C3.

2.4.4. Shading Decoder:

S1C128 - resize (2×) - concatenate via skip connection - S1C128 - resize (2×) - concatenate via skip connection - S1C128 - resize (2×) - concatenate via skip connection - S1C128 - resize (2×) - S1C1.

2.4.5. Fusion Module:

As illustrated in Figure 5, before feeding *RGB* features to the contextual attention module, a $1x1$ convolution kernel is used for dimension reduction. It maps 256-dimensional features to 32. The other $1x1$ convolution kernel is used for learning a weighted combination of the *RGB* features and the shading features. It maps 288-dimensional features to 64. Thus, the output of the fusion module provides 96 features maps to the shading decoder.

3. Recoloring Application

In this section, we provide a fundamental recoloring application to show that the results of the paper can help to solve a computer vision problem. It demonstrates the importance of being robust to the color leakage effects (color invariance). In an ideal case, recoloring algorithms aim at manipulating the colors (albedo) of an image while the illumination (shading) effects remain the same. For the demonstration, we substitute an object's albedo with a single color, then the newly created albedo is multiplied by the estimated shading to reconstruct the final image (i.e I = R x S). Figure 8 provides a number of cases, where the color leakages in the estimated shading maps cause even a very fundamental application to fail, having undesired color artefacts in the final reconstructed images. Our reconstructions are way more robust and closer to the ground-truths.

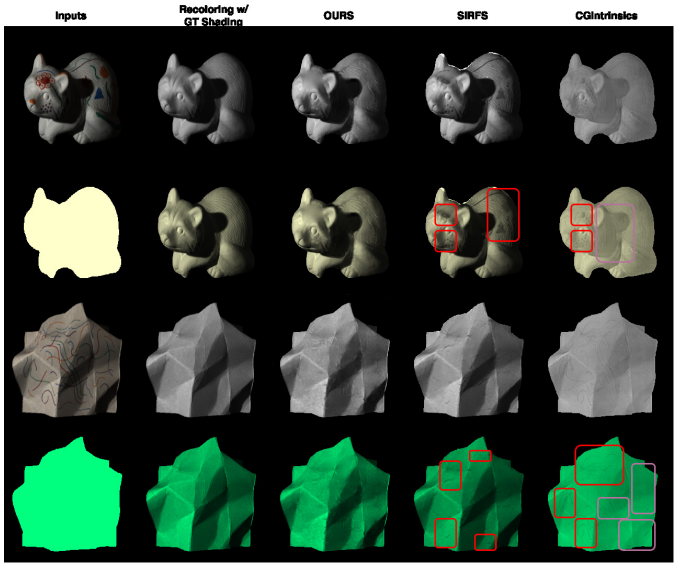

Fig. 8: Recoloring: albedo of the input is replaced with a single color. Leakage in the shading causes the application to fail having undesirable color artefacts. CGIntrinsics's uses over-smoothing to address color leakages, but the constraint causes it to fail to capture crisp shadows, marked with purple, and finer geometry. Our reconstructions are way more robust and closer to the ground-truths (Best viewed in digital format).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: `http://tensorflow.org/`. software available from tensorflow.org.

Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L., 2009. Imagenet: A large-scale hierarchical image database, in: IEEE Conference on Computer Vision and Pattern Recognition.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: IEEE International Conference on Computer Vision.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: IEEE Conference on Computer Vision and Pattern Recognition.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization, in: International Conference on Learning Representations.

Mao, X., Shen, C., Yang, Y., 2011. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections, in: Advances in Neural Information Processing Systems.

Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations.

Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P., 2004. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing , 600–6012.

Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T.S., 2018. Generative image inpainting with contextual attention, in: IEEE Conference on Computer Vision and Pattern Recognition.