Supplementary Material for Inverse Rendering for Complex Indoor Scenes: Shape, Spatially-Varying Lighting and SVBRDF from a Single Image

Zhengqin Li* zhl378@eng.ucsd.edu
Mohammad Shafiei* moshafie@eng.ucsd.edu
Ravi Ramamoorthi* ravir@cs.ucsd.edu
Kalyan Sunkavalli† sunkaval@adobe.com
Manmohan Chandraker* mkchandraker@eng.ucsd.edu
*University of California, San Diego †Adobe Research, San Jose

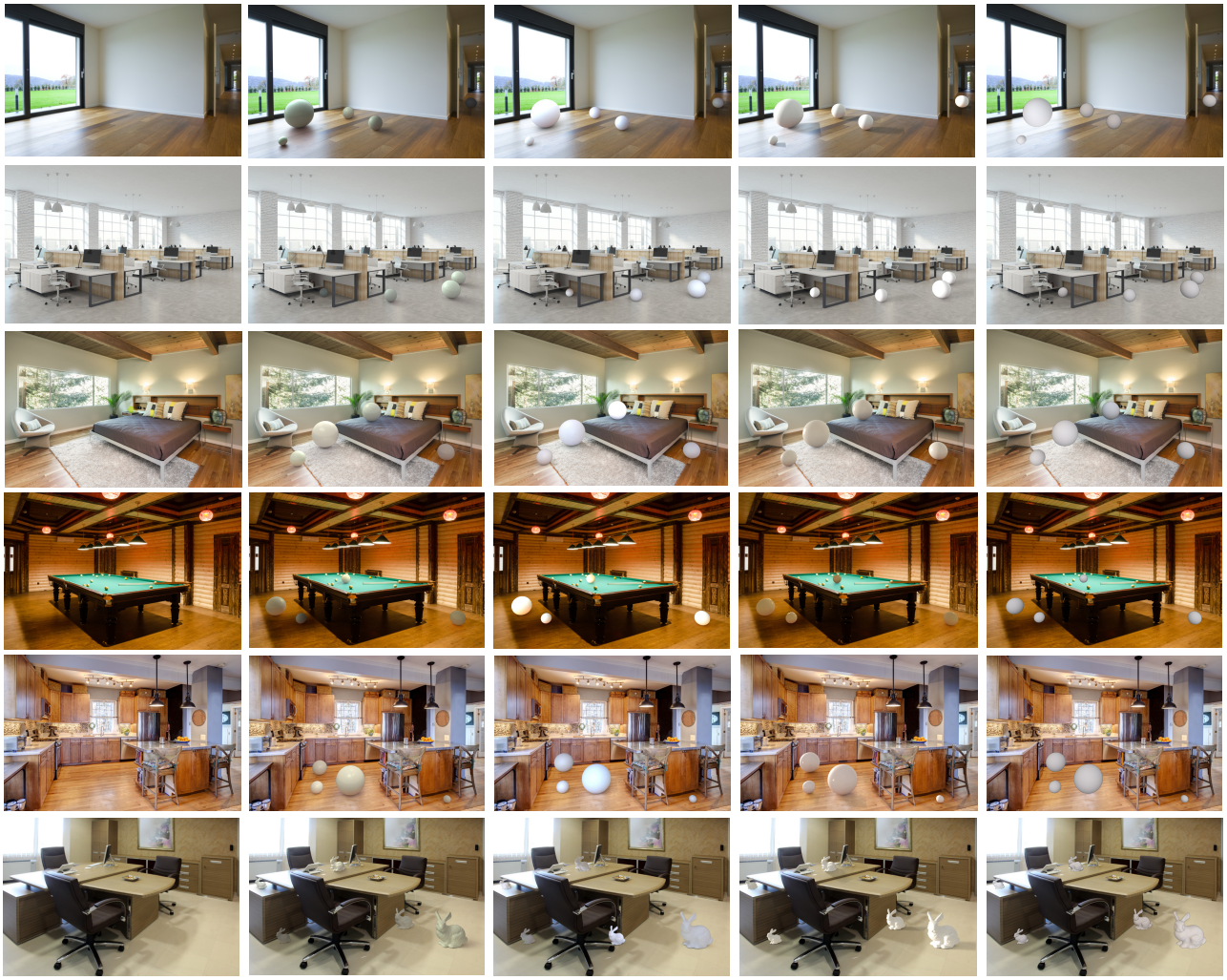

Figure 1: Object insertion examples and comparisons on real images. Our proposed method estimates shape (depth and surface normals), spatially-varying complex reflectance (based on a micro-facet SVBRDF model) and spatially-varying lighting from a single image of an indoor scene. Given these estimates, we can insert virtual 3D objects into these images and produce photo-realistic results where the objects look like they truly belong in the scene. Note the shading and specular highlights on the inserted spheres, the realistic shadows cast on the ground, the reflections from the ground onto the spheres and adaptation of the appearance of the spheres to the local shadows and shading in the scene. We also compare with previous works of Garon et al. [5], Barron et al. [2] and Gardner et al. [4] on real images. Note that in the results of [4], the shadows of some objects might be truncated by the plane we segment from the scene.

We have presented a method to automatically disentangle a single image of a complex indoor scene into its constituent physical scene factors – geometry, spatially-varying BRDF and spatially-varying illumination. In this document, we present more results, analyses and details to supplement the main paper. This includes: (i) several additional examples of object insertion and material editing (Sec. 1), (ii) ablation studies on both our synthetic test set and real images (Sec. 2), (iii) generalization to outdoor scenes (Sec. 3), (iv) a failure case (Sec. 4), (v) SG comparisons with SH (Sec. 5), (vi) SG parameter predictions (Sec. 6), (vii) optimization for SVSG (Sec. 7), (viii) tileable texture synthesis (Sec. 8), (ix) our OptiX-based GPU renderer (Sec. 9), (x) our BRDF model (Sec. 10), (xi) training details (Sec. 3).

1. Further Details of Novel Applications

Learning a disentangled shape, SVBRDF and spatially-varying lighting representation allows new applications that were hitherto not possible. We consider two of them here: object insertion and material editing in a single real image.

1.1. Object insertion

To render a new object into the scene, we first crop a planar region and pick a point on that plane to place the new object. The orientation, diffuse albedo and roughness value of the plane are all obtained from our predictions. We then render the plane and the object together using the lighting predicted at the point where we place the object. We render the plane and the new object together to ensure inter-reflections between them are properly simulated. We compute a high resolution environment map (512×1024) from the estimated spherical Gaussian parameters so that even very glossy material can be correctly handled.

We render two images, I_{all} and I_{pl} and two binary masks, M_{obj} and M_{all}. I_{all} is the rendered image of plane and object and I_{pl} is the rendered image of the plane only. M_{obj} is the mask of the object and M_{all} is the mask covering both the cropped plane and the object. We then edit the region of object and the region of cropped plane separately. Let I be the original image and I_{new} be the new image with the new rendered object. For the object region, we directly use the intensities as rendered in I_{all} on the virtual object:

$$I_{\text{new}} \odot M_{\text{obj}} = I_{\text{all}} \odot M_{\text{obj}}. \quad (1)$$

For the remaining region on the plane, we blend in the original image intensities with the ratio of I_{all} and I_{pl}:

$$I_{\text{new}} \odot (M_{\text{all}} - M_{\text{obj}}) = I \odot \frac{I_{\text{all}}}{I_{\text{pl}}} \odot (M_{\text{all}} - M_{\text{obj}}). \quad (2)$$

All operations in the above relation are pixel-wise. This compositing procedure utilizes the idea of ratio (or quotient images) that has been used in the past for relighting [11, 12].

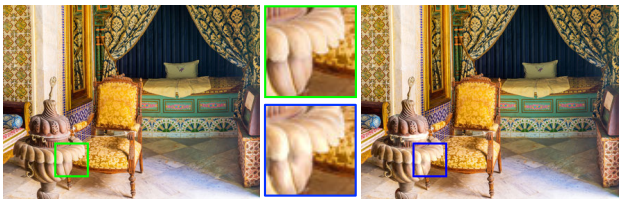

Figure 2: Changing the specularity of an object in a real image. We keep the predicted geometry, diffuse albedo and spatially varying lighting as predicted, but change the roughness value to 0.2 and re-render the object, leading to more prominent specular highlights.

It ensures that global effects due to object-plane interaction, such as soft shadows, are visualized (since they are rendered in I_{all} but absent in I_{pl}), while keeping intensities consistent with the overall image. This suppresses high frequency artifacts in I_{all} that might be caused by minor errors in estimation of albedo, roughness and lighting, thereby achieving greater photorealism. Note that we choose planar regions for insertion only due to ease of compositing. Our method does estimate per-pixel depth maps and captures its effects in I_{all}.

Further examples on real images Figures 1, 2, 9, 10 and 11 in the main paper show examples on real images. Figure 1 in this supplementary material shows several other examples with comparisons to other methods. In all these examples, we render white glossy objects (Ball and Stanford Bunny) with diffuse albedo $(0.8, 0.8, 0.8)$ and roughness value 0.2. We use white color so that the color of lighting and global illumination effects can be clearly observed. We keep the shape simple and the roughness low to better demonstrate the high frequency component in the predicted lighting.

Videos on real images To further illustrate the performance, the supplementary material also includes a video of an object moving around scenes corresponding to real images, as rendered by our prediction. Note that no temporal smoothness of constraints are imposed, yet the insertion is quite consistent with photorealistic shading and shadows as the object traverses various parts of the scene. This demonstrates the high quality of shape, material and spatially-varying lighting recovered by our method.

1.2. Material Editing

In Figures 3 and 12 of the main paper, we demonstrate three material editing examples by change the texture of a planar surface. We observe spatially varying lighting effects for all three examples. In particular, we observe the specular highlight in the original image is preserved after changing the material in Figure 3 in the main paper. Such specular highlight effects cannot be modeled by traditional intrinsic decomposition method since the direction of the incoming lighting is unknown. In Figure 2 here, we add a specular highlight to the selected non-planar object in a real image by

changing the roughness value to 0.2 and render the object with predicted diffuse albedo, geometry and spatially varying lighting. We compute the residual image before and after changing the roughness value and add it back to the original image. Even though the difference is subtle, we observe that the distribution of the specular highlight looks plausible.

1.3. Scales of lighting and diffuse albedo

We use scale invariant loss for both diffuse albedo and lighting prediction. However, for real applications discussed above, we need to recover the scale of both diffuse albedo and lighting. Let c_a and c_l be the coefficients of diffuse albedo and lighting, respectively. Recall that our rendering layer outputs a diffuse image $\tilde{I}_d$ and a specular image $\tilde{I}_s$. We can compute coefficients c_d and c_s to minimize the L_2 error between $c_d\tilde{I}_d + c_s\tilde{I}_s$ and input image I. Since our specular albedo is a constant, the scaling factor for our lighting prediction will be c_s and we have

$$c_l = c_s, \qquad c_a = \frac{c_d}{c_l}. \tag{3}$$

However, for some images, specularity might be hard to observe, in which case we neglect the specularity term and simply compute the coefficient using

$$c_a = \frac{1}{\max(A)}, \qquad c_l = \frac{c_d}{c_a}. \tag{4}$$

That is, we set the scale of diffuse albedo c_a so that the largest albedo in the image is 1 and compute the coefficient of the lighting accordingly. To decide which strategy to use to compute the scale of lighting and albedo, we compute the following determinant when we regress c_d and c_s:

$$\mathcal{D} = \frac{(\tilde{I}_d \cdot \tilde{I}_d)(\tilde{I}_s \cdot \tilde{I}_s) - (\tilde{I}_d \cdot \tilde{I}_s)^2}{K}, \tag{5}$$

where K is the number of pixels in the image. If $\mathcal{D} > 1e^{-7}$, we use (3), otherwise we use (4) to compute the coefficient.

2. Further Qualitative Examples

Ablation study on real images Figure 10 in the main paper includes an ablation study where qualitative outputs are shown for shape, material and lighting predictions, with a real image as input. Figure 3 shows several more examples on real input images, where we show the influence of the cascade design and bilateral solver on depths, normals, albedo, roughness and lighting. In each case, we observe that the recovered scene factors are plausible, with noticeable improvements in the reconstruction quality due to the cascade and bilateral solver. A rendering is also shown using the estimated parameters for shape, material and lighting, which is quite close to the original image, demonstrating the accuracy of the predictions. Note that this is a very challenging problem and no prior work, to the best of our knowledge, estimates arbitrary geometry, complex SVBRDF and spatially-varying lighting from a single input image.

Ablation study on synthetic images Figure 9 in the main paper shows a similar ablation study as the above on a synthetic image, where ground truth is available. Figure 4 depicts the same, that is, stage-wise outputs for various estimated scene components on a synthetic example from our test set. We observe that the cascade structure produces more globally correct (less noisy and sharper) estimates for albedo, roughness, normals and lighting, while the bilateral solver contributes to smoothness while preserving sharp boundaries. The rendered image based on the estimated entities closely resembles the input, which also shows the effectiveness of our rendering layer.

Figure 5 visualizes our spatially varying lighting predictions under different configurations using a synthetic example with ground truth. We observe that without direct supervision on spherical Gaussian parameters, the predicted lighting loses high frequency details while without the predicted material and geometry as input, the predicted lighting, especially the ambient color, does not sufficiently adapt spatially to the scene (possibly because of ambiguities between lighting and surface reflectance).

3. Generalization to Outdoor Scenes

In this section, we test how well our model, which is trained with synthetic indoor scenes only, generalizes to outdoor scenes. The qualitative results are shown in Figure 6. While the network tries to interpret the outdoor scene as a room surrounded by walls, we observe that the overall estimation of geometry, lighting and diffuse albedo look reasonable. We also try to insert a new object into the scene using our predictions following the pipeline proposed in Section 1, then compare with a state-of-the-art outdoor lighting estimation method [6]. As shown in the last two columns in Figure 6, the method of [6] can better preserve high-frequencies in outdoor illumination, which results in shadows with hard boundaries, while our method tends to predict more low frequency lighting. This is probably due to the domain gap between our training and test images. However, we note that our model can usually predict the direction of the incoming light correctly and the predicted spatial variation in the lighting looks more realistic compared to [6], which predicts a single lighting model for the whole image.

4. A Failure Case

While we observe largely successful object insertions in most experiments, some failure cases do occur. The ambiguity between albedo and lighting is a hard one to disentangle. In some cases, the albedo is estimated to be too bright (dark), with the lighting correspondingly estimated as too dark (bright). An example is shown in Figure 7. Nevertheless, we emphasize that being able to estimate spatially-varying lighting along with SVBRDF and shape is an extremely hard problem, for which our network succeeds

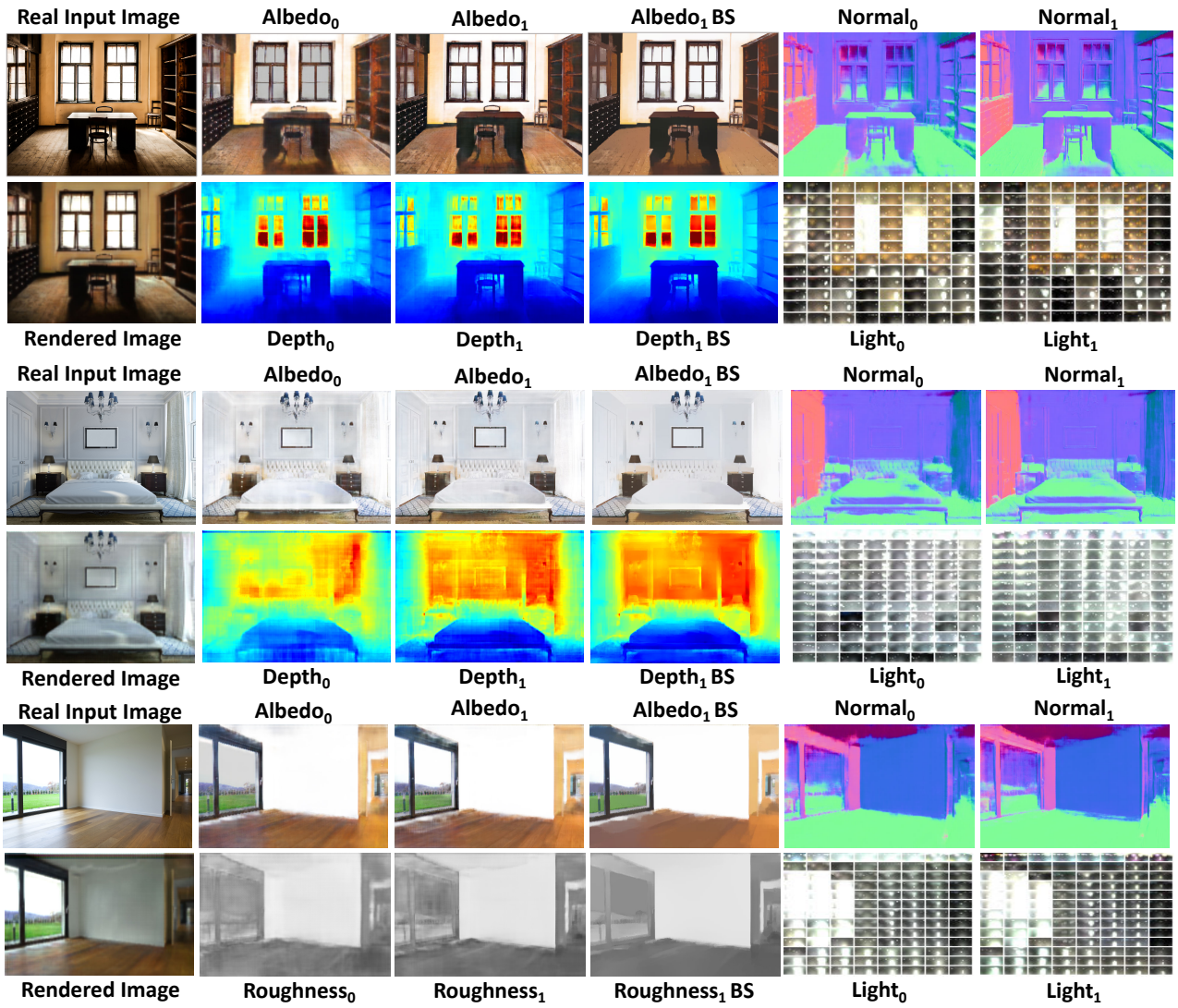

Figure 3: Qualitative comparisons of adding cascade structure and bilateral solver on three real examples. The iterative refinement by the cascade structure adds more fine details and removes artifacts for all intrinsic components, leading to sharper predictions. The bilateral solver brings noticeable improvements through piece-wise smoothness priors on depth, roughness and albedo.

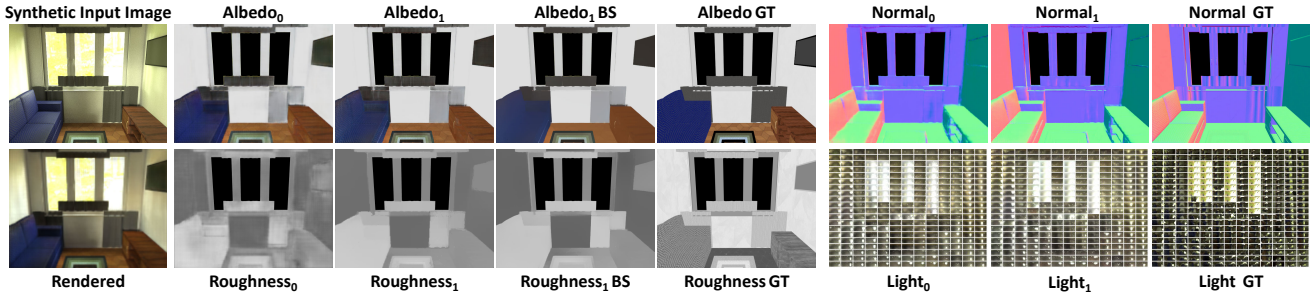

Figure 4: Impact of cascade and bilateral solver on a synthetic example from our test set. We observe that all the scene components benefit from cascaded estimation, while the bilateral solver is effective at refinement. The output of the rendering layer closely matches the input.

in an overwhelming number of experiments.

5. Comparison of SVSG and SVSH

Figure 8 compares using spherical Gaussian and spherical harmonics to approximate the spatially varying lighting,

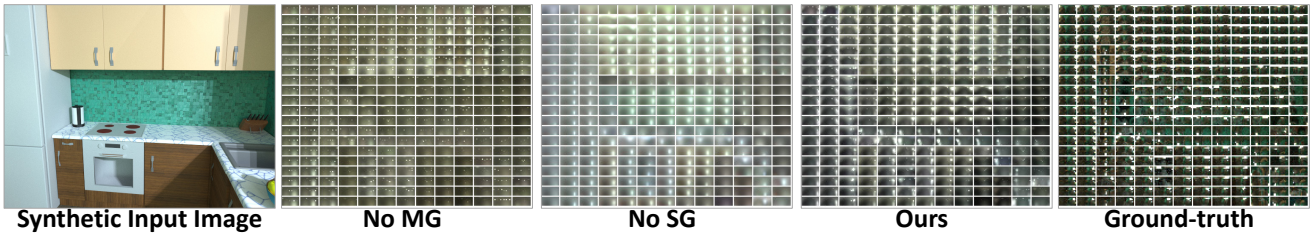

Figure 5: Comparison of lighting predictions. From left to the right are input image, **No MG:** without predicted material and geometry as input, **No SG:** without ground-truth spherical Gaussian parameters as supervision and our predictions and the ground-truth lighting.

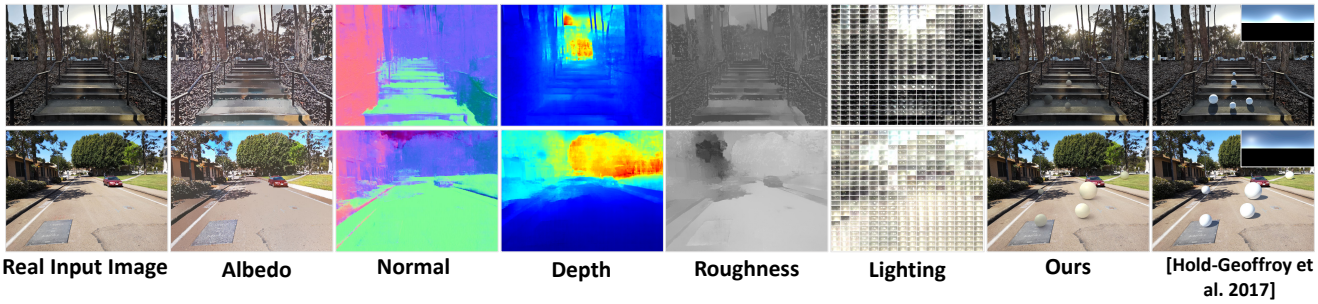

Figure 6: Our proposed method with a single real image of an outdoor scene as input. Even though our method is trained with synthetic indoor scenes only, it generalizes reasonably well to outdoor scenes, which allows us to achieve reasonable object insertion results (the second last column) compared to the state-of-the-art [6] (the last column).

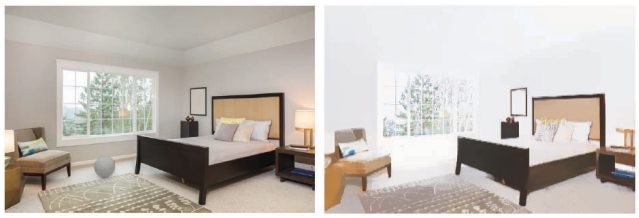

Figure 7: A failure case. (Left) The inserted object is rendered with an appearance that is darker than expected. (Right) The likely cause is an over-bright albedo estimation, which is traded off by a lighting estimate that has lower intensities.

	Lighting ($\log L_2$)	Image (L_2)
SH (75 para.)	4.43	8.6×10^{-3}
SG (72 para.)	**1.56**	$\mathbf{7.6 \times 10^{-3}}$

Table 1: Quantitative comparison of using spherical harmonic (SH) and spherical Gaussian (SG) for lighting representation. From left to the right, the average error when using each representation to approximate per pixel lighting in Figure 8, the MSE of the rendered images in Figure 8 in the main paper. Spherical Gaussian performs better in both metrics.

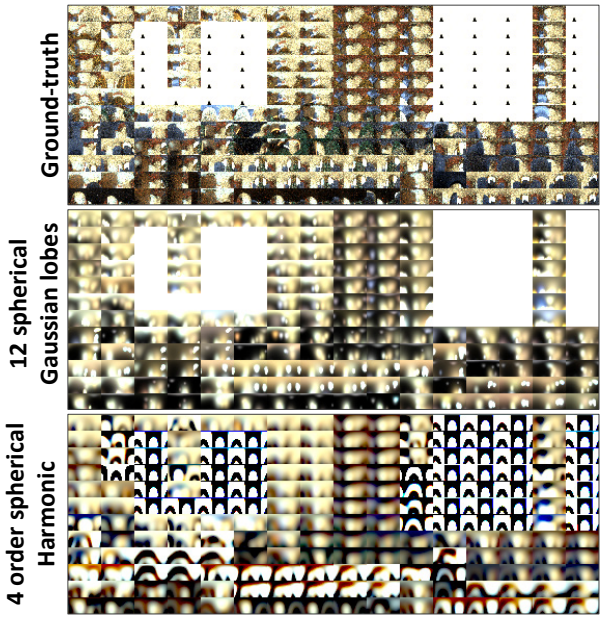

Figure 8: Comparison of approximating lighting with spherical harmonics and spherical Gaussian.

which corresponds to the examples in Figure 8 of the main paper. It is clearly observed that with a similar number of parameters, spherical Gaussians can better recover high frequency effects, resulting in a reconstructed spatially-varying lighting closer to ground truth. Table 1 shows the error of using spherical Gaussian and spherical Harmonics to approximate per-pixel lighting. Wish similar number of parameters, the approximation error of using spherical Gaussian is significantly lower.

6. Spatially Varying Lighting Predictions

As shown in Equation (3) of the main paper, the lighting prediction is given by $\{\tilde{\xi}_k\}$, $\{\tilde{\lambda}_k\}$ and $\{\tilde{F}_k\}$. In practice, those parameters are obtained by a non-linear transformation on the output of $\mathbf{LightNet}_0$. Specifically, for an output of $\mathbf{LightNet}_0$ given by $\{\bar{\xi}_k\}$, $\{\bar{\lambda}_k\}$ and $\{\bar{F}_k\}$, the lighting prediction is computed as:

$$\tilde{\xi}_k = \frac{\bar{\xi}_k}{||\bar{\xi}_k||_2^2} \quad (6)$$

$$\tilde{\lambda}_k = \tan\left(\frac{\pi}{4}(\bar{\lambda}_k + 1)\right) \quad (7)$$

$$\tilde{F}_k = \tan\left(\frac{\pi}{4}(\bar{F}_k + 1)\right). \quad (8)$$

Note that in the cascade structure, the inputs to $\mathbf{LightNet}_1$ are $\{\bar{\xi}_k\}$, $\{\bar{\lambda}_k\}$ and $\{\bar{F}_k\}$ instead of $\{\tilde{\xi}_k\}$, $\{\tilde{\lambda}_k\}$ and $\{\tilde{F}_k\}$, since we observed training to be more stable if the inputs to the network are of low dynamic range.

7. Ground Truth Spherical Gaussian Lobes

We compute ground-truth spherical Gaussian lobe parameters by approximating the environment lighting using the LBFGS method. These parameters are used to supervise the spatially varying lighting prediction. We use 12 lobes to approximate per-pixel lighting. To facilitate the training, we assign an order to the 12 lobes by constraining each lobe to be in a certain region of the hemisphere, roughly divided into 2×6 regions. Following the notation in Section 4 of the main paper, we define $\{\theta_k\}$, $\{\phi_k\}$, $\{\lambda_k\}$ and $\{F_k\}$ to be the spherical Gaussian parameters, where

$$\xi_k = (\sin\theta_k \cos\phi_k, \sin\theta_k \sin\phi_k, \cos\theta_k). \quad (9)$$

In order to add the constraints, we reparameterize the spherical Gaussian parameters as $\{\hat{\theta}_k\}$, $\{\hat{\phi}_k\}$, $\{\hat{\lambda}_k\}$ and $\{\hat{F}_k\}$:

$$\lambda_k = \exp(\hat{\lambda}_k), \quad (10)$$

$$F_k = \exp(\hat{F}_k), \quad (11)$$

$$\theta_k = a\tanh(\hat{\theta}_k) + b_k, \quad (12)$$

$$\phi_k = c\tanh(\hat{\phi}_k) + d_k, \quad (13)$$

where $a = \frac{3\pi}{8}$ and $c = \frac{\pi}{2}$ are scaling factors. Here, b_k and d_k are offset parameters that are computed as

$$b_k = \frac{\pi}{4}(k \bmod 2 + \frac{1}{2}), \quad (14)$$

$$d_k = \frac{\pi}{3}(k \bmod 6 + \frac{1}{2}) - \pi. \quad (15)$$

The initialization of the parameters are $\hat{\theta}_k = 0$, $\hat{\phi}_k = 0$, $\hat{F}_k = 0$ and $\hat{\lambda}_k = \log(\frac{\pi}{2})$. The loss function is the log-encoded loss as described in Eq. (6) in the main paper.

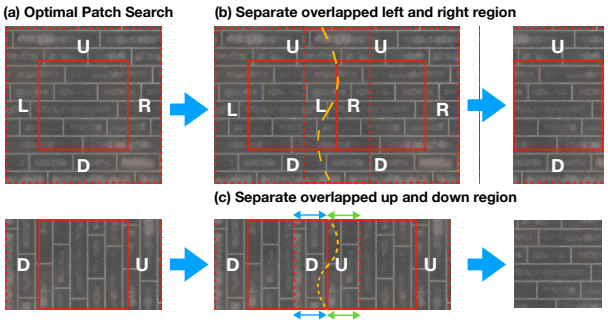

Figure 9: Our graph-cut based method for tileable texture synthesis. We first stitch the left-right boundaries of textures and then the up-down boundaries. When stitching up-down boundaries, we add a hard constraint so that left-right boundaries remain tileable.

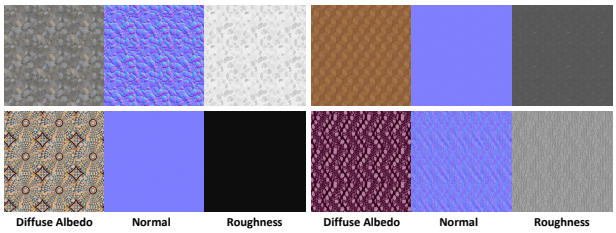

Figure 10: Results of tileable texture synthesis. Each image is generated by tiling 3×3 patches together.

8. Tileable Texture Synthesis

We use graph-cut based approach to generate tileable texture, which has the advantages of keeping the original texture structures [9]. The overall process is summarized in Figure 9. We first crop smaller patch from the original SVBRDF textures and synthesize the cropped patch into tileable texture. Given the required size of the patch, we first globally search for the optimal patch by minimizing the gradient perpendicular to the boundary of the patches. More specifically, let $\mathcal{B}_x$ and $\mathcal{B}_y$ be the set of pixels on the horizontal and vertical boundaries of the patch. A_i, N_i and R_i are the diffuse color, normal and roughness at pixel i respectively. We search for a patch so that

$$\sum_{i \in \mathcal{B}_y} \lambda_A \nabla_x A_i + \lambda_N \nabla_x N_i + \lambda_R \nabla_x R_i + \sum_{j \in \mathcal{B}_x} \lambda_A \nabla_y A_j + \lambda_N \nabla_y N_j + \lambda_R \nabla_y R_j \quad (16)$$

is minimized. The equation (16) can be efficiently computed using integral graph. The overall complexity of finding optimal patch will be $\mathcal{O}(K)$ where K is the number of pixels in the image. By minimizing (16), we avoid strong gradient near the boundaries of the patch, so that we can reduce artifacts in tileable texture synthesis.

Once we find the patch, we crop not only the patch but also its surrounding regions. To make the patch tileable in x direction, we overlap the right and left surrounding regions

and use graph-cut method to find the best seam to separate the overlapping regions by minimizing a customized energy function. Unlike energy function in [9] which encourages the value of pixels at seam to be similar to the value of pixels in the original textures, our energy function encourages the gradients of pixels at the seam to be similar to the gradients of pixels in the original textures. As in [9], we formulate the problem as a labeling problem. Let $I_{r,c}$ be an pixel in the overlapped texture map and $L_{r,c} \in \{1, 2\}$ be its label. With some abuse of notation, we define $I^i_{r,c}$ to be the value of pixel from patch i, $i \in \{1, 2\}$. The gradient across the patches i and j is defined as $\nabla_x I^{i,j}_{r,c} = I^j_{r,c+1} - I^i_{r,c}$. Then the loss $\mathcal{L}_I(L_{r,c} = 1, L_{r,c+1} = 2)$ is defined as

$$\min \left(\frac{||\nabla_x I^{1,2}_{r,c} - \nabla_x I^{1,1}_{r,c}||_1}{\max\left(||\nabla_x I^{1,1}_{r,c}||_1, 0.1\right)}, \frac{||\nabla_x I^{1,2}_{r,c} - \nabla_x I^{2,2}_{r,c}||_1}{\max\left(||\nabla_x I^{2,2}_{r,c}||_1, 0.1\right)} \right)$$

The final loss for is a weighted combination of losses from different texture map:

$$\lambda_A \mathcal{L}_A + \lambda_N \mathcal{L}_N + \lambda_R \mathcal{L}_R. \tag{17}$$

To make the texture tileable in y direction, we repeat the above process by overlapping the up and down surrounding regions and finding the seam to separate them using graph-cut again. Notice that when separating the overlapping up and down regions, we need to make sure that the pixels at the right and left boundaries of the patches are from the same region so that the patch will remain tileable in x direction. We achieve this by adding an infinite smoothness term between every pair of pixels at the left and right boundaries in the same row so that they will always come from the same region. Figure 10 shows some texture synthesis examples. Each example is generated by tiling 3×3 original patches together. For each material from our dataset, we crop three patches of different sizes and the three patches will be considered as different materials in the following mapping SVBRDFs stage.

9. Fast Physically-Based Rendering

To render high quality images with realistic appearances, it is necessary to use a physically based renderer that models complex light transport effects such as global illumination and soft shadows. However, current open source CPU renderers are too slow for creating a large dataset, especially to render per-pixel lighting. Thus, we implement our own physically-based GPU renderer using Nvidia OptiX [1]. To render a 480 $\times$ 640 image with 16384 samples per pixel, our renderer on Tesla V100 GPU needs 3-6 minutes, while Mitsuba on 16 cores of Intel i7-6900K CPU needs around 1 hour. Figure 11 compares images rendered with Mitsuba [7] and with our renderer using the same amount of time.

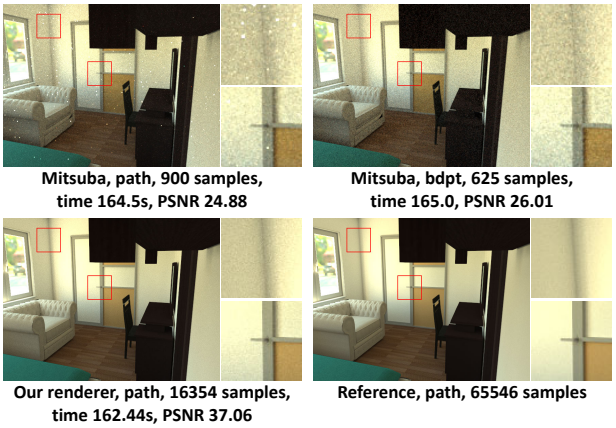

Figure 11: Comparisons of images rendered with Mitsuba and our GPU renderer in the same amount of time using path tracing. The quality of the image rendered by our renderer in less than three minutes is much better. It takes about 50 minutes for Mitsuba to achieve similar results.

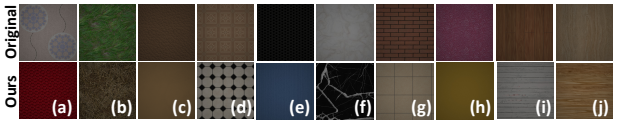

Figure 12: The ten material categories and the corresponding spatially varying diffuse textures from both the original and our dataset. From left to the right: (a) fabric, (b) ground, (c) leather, (d) stone floor, (e) plastic, (f) stone specular, (g) stone wall, (h) wall paint, (i) wood floor, (j) wood.

.

wall paint	stone wall	leather	stone floor	plastic
127	185	10	172	94
stone specular	ground	fabric	wood floor	wood
25	243	180	25	42

Table 2: The distribution of materials in our dataset, for the chosen semantic categories.

10. BRDF Model and Material Categories

Our microfacet model We use a physically motivated microfacet BRDF model in our dataset. Let A, N and R be the spatially-varying diffuse albedo, normal and roughness, respectively. The BRDF model $f(l, v; A, N, R)$ is:

$$f(l, v; A, N, R) = f_d(l, v; A, N) + f_s(l, v; N, R), \tag{18}$$

$$f_d(v, l; A, N) = \frac{A}{\pi}, \tag{19}$$

$$f_s(v, l; N, R) = \frac{D(h, R)F(v, h)G(l, v, N, R)}{4(N \cdot l)(N \cdot v)}, \tag{20}$$

where $f_d(\cdot)$ and $f_s(\cdot)$ are the diffuse and specular BRDF components. Here, v and l are view and lighting directions, and h is the half angle vector, while $D(h, R)$, $F(v, H)$ and $G(l, v, h, R)$ are the distribution, Fresnel and geometric

terms respectively, defined as

$$\begin{aligned} D(h,R) &= \frac{\alpha^2}{\pi\left[(N\cdot h)^2(\alpha^2-1)+1\right]^2} \\ \alpha &= R^2, \\ F(v,h) &= (1-F_0)2^{-[5.55473(v\cdot h)+6.8316](v\cdot h)}, \\ G(l,v,R,N) &= G_1(v,R,N)G_1(l,R,N), \\ G_1(l,R,N) &= \frac{N\cdot l}{(N\cdot l)(1-k)+k}, \\ G_1(v,R,N) &= \frac{N\cdot v}{(N\cdot v)(1-k)+k}, \\ k &= \frac{(R+1)^2}{8}. \end{aligned}$$

We set $F_0 = 0.05$ as suggested in [8].

Material categories For mapping our materials on the geometry of the original dataset [14] in a manner consistent with semantics such as objects in the scene, we manually classified our dataset as well as the original materials into 10 categories. Samples from each dataset for these categories are shown in Figure 12. The number of material samples for each in our dataset are shown in Table 2.

11. Network Structures and Training Details

The network structures are summarized in Figure 13. Note that we use group normalization [15] instead of batch normalization so that we can train the network with smaller batch size. The padding size is dynamically assigned according to the feature map size so that the feature maps after up-sampling can be aligned with the feature maps coming from skip links. Therefore, our network can process image of arbitrary size without scaling and cropping. The network for spatially varying lighting predictions has more parameters because we find it necessary to achieve reasonable performances for this task.

We use Adam optimizer to train our network. Each level of cascade network is trained separately. To train cascade network of level i, we first train $\mathbf{MGNet}_i$ and $\mathbf{LightNet}_i$ separately and then fine-tune them together. The loss function to train $\mathbf{MGNet}_i$ is

$$\alpha_A\mathcal{L}_A + \alpha_N\mathcal{L}_N + \alpha_R\mathcal{L}_R + \alpha_D\mathcal{L}_D, \tag{21}$$

with various terms as defined in Section 4 in the main paper. We add the rendering loss $\mathcal{L}_{ren}$ when training $\mathbf{LightNet}_i$. The loss function to train $\mathbf{LightNet}_i$ is

$$\alpha_L\mathcal{L}_L + \alpha_{ren}\mathcal{L}_{ren} + \sum_{k=1}^{K}\alpha_\lambda\mathcal{L}_{\lambda_k} + \alpha_\xi\mathcal{L}_{\xi_k} + \alpha_F\mathcal{L}_{F_k}. \tag{22}$$

The loss function for fine-tuning the whole pipeline is defined in Eq. (8) in the main papers. Finally, the loss function to train $\mathbf{BSNet}$ is

$$\alpha_A\mathcal{L}_A + \alpha_R\mathcal{L}_R + \alpha_D\mathcal{L}_D. \tag{23}$$

All other hyper parameters including initial learning rate, training epochs and and coefficients $\alpha_{(\cdot)}$ are summarized in Table 3 and Table 4. The learning rates are decreased by half every 10 epochs.

Fine-tuning on real datasets We use similar strategy to fine-tune on IIW dataset [3] and NYU dataset [13]. We take the trained model and fine-tune each level of cascade sequentially. The learning rate is $1e^{-4}$ and the batch size is 4 for the first level of cascade and 2 for the second level. In each iteration, we send two batches of images to the network, one from our synthetic dataset and the other from the real dataset. The $\mathbf{MGNet}_i$ and $\mathbf{LightNet}_i$ are trained in an end-to-end manner. The loss function for images from our dataset is the same as Eq. (8) in the main paper. The loss function for fine-tuning on NYU dataset is just a combination of the loss on each component. We add the rendering loss by comparing the rendered image with the input image:

$$\alpha_{ren}\mathcal{L}_{ren} + \alpha_N\mathcal{L}_N + \alpha_D\mathcal{L}_D. \tag{24}$$

When fine-tuning on IIW dataset, we include ordinal reflectance loss $\mathcal{L}_{ord}$ which is the same as defined in [10]. The loss function for images from IIW dataset is

$$\alpha_{ren}\mathcal{L}_{ren} + \alpha_{ord}\mathcal{L}_{ord}. \tag{25}$$

When training on NYU dataset, we also do data augmentation by randomly flipping, cropping and scaling the input images with a scale uniformly sampled from 0.8 to 1.2 since the dataset is relatively small.

References

[1] NVIDIA OptiX. https://developer.nvidia.com/optix. 7

[2] Jonathan T Barron and Jitendra Malik. Intrinsic scene properties from a single rgb-d image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 17–24, 2013. 1

[3] Sean Bell, Kavita Bala, and Noah Snavely. Intrinsic images in the wild. *ACM Transactions on Graphics (TOG)*, 33(4):159, 2014. 8

[4] Marc-André Gardner, Kalyan Sunkavalli, Ersin Yumer, Xiaohui Shen, Emiliano Gambaretto, Christian Gagné, and Jean-François Lalonde. Learning to predict indoor illumination from a single image. *ACM Trans. Graphics*, 9(4), 2017. 1

[5] Mathieu Garon, Kalyan Sunkavalli, Sunil Hadap, Nathan Carr, and Jean-François Lalonde. Fast spatially-varying indoor lighting estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6908–6917, 2019. 1

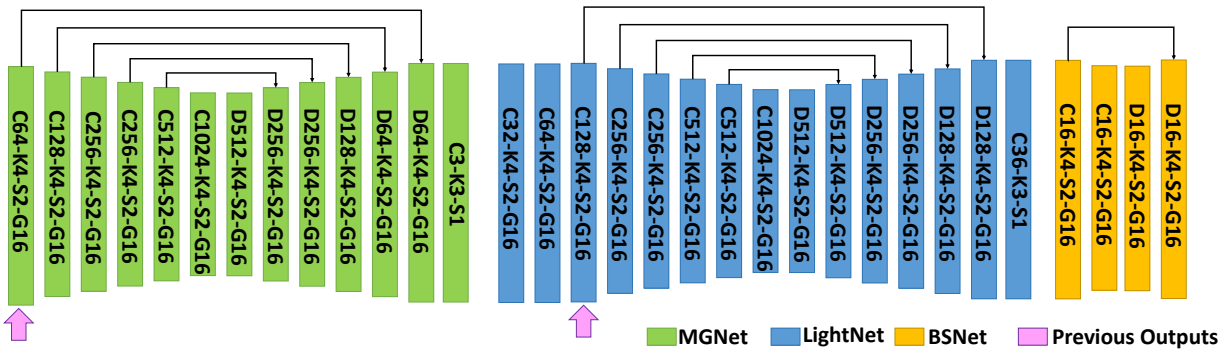

Figure 13: Our network architectures. $\mathbf{C}/\mathbf{D}X$-$\mathbf{K}Y$-$\mathbf{S}Z$-$\mathbf{G}W$ represents a convolution or transpose convolution layer with channel number X, kernel size Y, stride Z and group normalization with W channels in a group. **Previous Outputs** represents the place where we concatenate the predictions of previous level of cascade with current features.

	α_A	α_N	α_R	α_D	α_L	α_{ren}	α_λ	α_ξ	α_F	epochs	iters.	$\text{lr}_{\mathbf{MG}}$	$\text{lr}_{\mathbf{Light}}$	batch
$\mathbf{MGNet}_0$	1.5	1.0	0.5	0.5	-	-	-	-	-	21	-	$1e^{-4}$	-	16
$\mathbf{LightNet}_0$	-	-	-	-	10	10	$5e^{-4}$	1	0.5	15	-	-	$1e^{-4}$	4
Fine Tune_0	7.5	5.0	2.5	2.5	10	10	$5e^{-4}$	1	0.5	-	4000	$1e^{-9}$	$1e^{-6}$	4
$\mathbf{MGNet}_1$	1.5	1.0	0.5	0.5	-	-	-	-	-	8	-	$1e^{-4}$	-	6
$\mathbf{LightNet}_1$	-	-	-	-	10	10	$5e^{-4}$	1	0.5	8	-	-	$1e^{-4}$	4
Fine Tune_1	7.5	5.0	2.5	2.5	10	10	$5e^{-4}$	1	0.5	-	4000	$1e^{-9}$	$1e^{-6}$	3

Table 3: Hyper parameters for training $\mathbf{MGNet}_i$ and $\mathbf{LightNet}_i$.

	α_A	α_R	α_D	iters.	$\text{lr}_{\mathbf{BS}}$	batch
BSNet	1.5	0.5	0.5	600	$1e^{-4}$	6

Table 4: Hyper parameters for training **BSNet**.

[6] Yannick Hold-Geoffroy, Kalyan Sunkavalli, Sunil Hadap, Emiliano Gambaretto, and Jean-François Lalonde. Deep outdoor illumination estimation. In *CVPR*, 2017. 3, 5

[7] Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org. 7

[8] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 2013. 8

[9] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *TOG*, 22(3):277–286, 2003. 6, 7

[10] Zhengqi Li and Noah Snavely. Cgintrinsics: Better intrinsic image decomposition through physically-based rendering. In *ECCV*, pages 371–387, 2018. 8

[11] Stephen R Marschner and Donald P Greenberg. Inverse lighting for photography. In *Color and Imaging Conference*, volume 1997, pages 262–265. Society for Imaging Science and Technology, 1997. 2

[12] Amnon Shashua and Tammy Riklin-Raviv. The quotient image: Class-based re-rendering and recognition with varying illuminations. *PAMI*, 23(2):129–139, Feb. 2001. 2

[13] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012. 8

[14] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 8

[15] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, pages 3–19, 2018. 8