Object-based Illumination Estimation with Rendering-aware Neural Networks Supplementary Material

Xin Wei[1,2], Guojun Chen[1], Yue Dong[1], Stephen Lin[1], and Xin Tong[1]

[1] Microsoft Research Asia
[2] Zhejiang Univiersity

1 Introduction

This supplementary material includes additional details on training data preparation (Sec. 2), implementations (Sec. 3) and additional experimental results (Sec. 4).

2 Training data preparation

The training data is composed of many elements such as reflectance maps and angular environment lighting which are difficult to collect in large volumes from real scenes. As a result, we choose to train our neural networks with synthetically generated data. For better network generality to real-world inputs, this synthetic data is prepared to better reflect the properties of real scenes.

Objects The objects used in our synthetic dataset are collected from the Adobe Stock market. We first manually select 80 tables, cabinets, sofas, etc. that often serve as supporting surfaces for objects, and then select 600 objects that commonly appear on top of such surfaces. We chose to use Adobe Stock as our data source since it contains models with realistic textural details and material variations that better reflect real objects.

Object layouts The layouts for the objects are determined by first selecting a support surface and then randomly choosing 1 - 6 objects from the set to be put on top. The objects are placed at random positions and rotations on the surface, with an upright orientation. We regard all the objects in the object dataset to be compatible with each support surface, such that their semantic relationships need not be manually assigned. The upward direction is provided for each object. If an object has multiple possible upward directions, we include a copy of the object for each valid upward orientation.

Viewpoints We uniformly sample viewpoints on the upper hemisphere as defined by the support surface. The θ direction of the view is restricted to the range of $0 - 80$ degrees, avoiding glancing views of the support surface which are rarely seen in AR scenarios.

Light sources To represent a natural distribution of real-world environment lights, we use 1800 real measured environment maps from [1] together with 14K randomly generated multiple area light sources as our light sources. We also use the environment maps to construct the backgrounds of the rendered scenes.

Depth maps We approximate the inaccuracy of real-world depth sensors by generating a lower resolution depth map from the synthetic data and obtaining rough geometry from it through Poisson surface reconstruction. We manually choose the depth map resolution and the oct-tree level such that the rough geometry contains only large-scale geometric features that can be robustly captured by off-the-shelf depth sensors.

Our synthetic dataset is composed of real environment lighting, physically based materials, and scene layouts modeled by experienced artists. Images are rendered with a physically based renderer, where the visibility of the scene for each environment light pixel is computed by a shadow map. After integrating contributions of all environment map pixels, our render generates all-frequency shadow and lighting effects of the scene. Most of our 3D scene only contains a small surrounding region of the objects and the environment out of this region is modeled by the environment lighting. We thus ignore the multi-bounce effect in the local 3D scene due to its expensive computational cost and limited numeric contribution to the final rendering result.

Rendering We render the synthetic images using a custom-built phyically based GPU renderer. Although indoor scenes contains strong interreflection effects, our render only contains a local view of a scene and the interreflection of the surroundings are already included in the environment map, we choose to ignore the local interreflection effects. This speeds up the synthetic training data generation process, also allows real-time rendering for AR applications. Figure 1 compares rendering results with and without the local interreflection effects, since the interreflection of the surroundings are already considered, the rendering results are similar to each other.

For environment lighting, we perform importance sampling over the environment light; then for each sampled light direction, we compute visibility using shadow maps and compute the shading for each pixel via rasterization. To avoid aliasing, we perform 8×8 supersampling for each pixel with all the maps we rendered. The same rendering system is used to synthesize virtual objects in our AR application.

When creating the ground truth training data, the diffuse and specular shading maps are rendered by setting the specular or diffuse coefficient to 0, respectively.

3 Implementation details

Network structures An overview of our network is presented in Figure 8. The detailed structures in the network are subsequently shown: decomposition network

(Figure 9); diffuse shading translation network (Figure 10); specular shading translation network (Figure 11) and the fusion network (Figure 12).

Pre-training of the diffuse shading translation network For the diffuse shading translation network, the training is supervised using the ground truth environment light. Since the $8 \times 8 \times 6$ angular feature map has strong correlation with the $8 \times 8 \times 6$ auxiliary irradiance maps, we also use the environment map at the corresponding cube map resolution to supervise training for those feature maps. In practice, we use a 1×1 convolution to convert those feature maps into a $8 \times 8 \times 6$ RGB image, where the image should match the ground truth environment map converted to the same resolution. The training for both the full resolution and the low resolution cube maps is conducted with the Huber loss.

Distortion-aware angular domain supervision For convenience in convolution, we represent the angular domain environment light with the latitude and longitude parametrization. To compensate for distortion, we design distortion-aware angular supervision. Instead of determining outputs and providing supervision with only one parameterization, we use latitude and longitude parameterizations with different rotations: one with the view direction at the center, one with a 180° yaw rotation, and the other with a 90° pitch rotation. When converting the feature maps into the latitude and longitude parameterization, we make three copies of them, each with one of the three parameterizations, and the network generates outputs for each of them. In computing the loss function, we weight the loss according to the following equation:

$$\omega(\theta, \phi) = \begin{cases} \Omega(\theta, \phi) & -135^\circ \leq \phi \leq 135^\circ \\ \Omega(\theta, \phi)(180 - |\phi|)/45 & \phi > 135^\circ \text{ or } \phi < -135^\circ \end{cases} \tag{1}$$

where $\Omega(\theta, \phi)$ is the solid angle of one pixel with its pixel center at θ, ϕ. There exists a solid angle fall-off toward the polar regions (top and bottom boundaries) of such parameterization, and we added additional fall-offs toward the left and right boundaries, so that distorted and unstable regions will contribute less to the loss function. During inference, we also calculate a weighted average over the three output parameterizations that emphasizes regions with less distortion.

Ideally, the three parameterizations should be processed with three separate group convolutions having shared convolution weights. For the angular convolution layers of the spatial-angular translation networks, we employ such separate group convolutions and shared weights. However, tripling the size of the convolution layers in the fusion network greatly increases the computation and memory cost, so we simply concatenate the three parameterizations as a nine-channel image, while keeping the same number of convolution channels.

Recurrent convolution training When training the recurrent convolution layers with sequential data and expanding the recurrent layers, in practice, we expand the recurrent layers to accommodate 10 consecutive frames during training. Due

to the size of the remaining (non-recurrent) layers and the size of sequential data, training the recurrent layers together with all the other layers consumes more memory and computation. For greater efficiency, we train the recurrent and non-recurrent layers separately. We first train all the non-recurrent layers with single-frame data, which allows us to train with a larger batch size containing more variation. We then initialize the recurrent convolutions with zero weights, and only train the weights for the recurrent layers, with the non-recurrent layers fixed.

Latitude and longitude representation The conversion from cube maps to the latitude and longitude representation, as well as the mapping of specular reflections to the mirror direction, are implemented as custom layers based on the *gather* and *splat* operations, respectively. The rough geometry is directly triangulated from the depth input, and the irradiance map is rendered using shadow maps.

Training stages The decomposition and the diffuse shading translation are first trained separately; then the fusion network is trained with input from fixed diffuse and specular translation networks. We feed the ground truth diffuse and specular shading maps as input when training the diffuse shading translation network and the fusion network. Finally, all the components are connected into a full system and trained end-to-end, while maintaining the supervision for each of the intermediate outputs.

Rendering error We compute the rendering error as one evaluation metric. For synthetic input, we re-render the same scene with the estimated light, and compare the rendering results to the input image, which is lit by the ground truth light. For real measured data, we compose one virtual object into the image using the estimated light, and compare this with composition using the measured ground truth light.

Single image input Due to the recurrent nature of our network, to avoid temporal information affecting the estimation result for single image test cases, we treat them as static scenes with static lighting, and run 10 identical frames of input through the recurrent network, taking the final stable output as the final result.

3.1 Evaluation setup

Synthetic generated environment maps To systematically test the robustness of our system to different lighting environments, we also prepare separate sets of synthetically generated lighting conditions that provide more continuous coverage over the range of lighting environments. For each set of lighting conditions, we render each scene and each view under those lighting conditions with ten random rotations. We then plot the average error of our system on the different lighting conditions in the set. Specifically, the sets of lighting conditions are as follows: (1) One square-shaped area light source of varying size. The area of the light source ranges from 0.1 to 3.0 solid angles. (2) Multiple fixed-sized area light

sources, each of 0.3 solid angles. We put multiple area light sources randomly in the upper hemisphere, and our test set contains cases that range from 1 to 8 area light sources. Experimental results with those synthetic generated environment maps are shown in Figures 3 and 4 of the main paper.

Test set with varying object materials The same geometric shape is rendered with different object materials. We first test materials with different roughness values, using a homogeneous specular BRDF with $[0.5, 0.5, 0.5]$ diffuse albedo, $[0.5, 0.5, 0.05]$ specular albedo, and roughness varying from 0.05 to 0.5. We then fix the roughness to 0.1 (a glossy surface) and vary the diffuse and specular ratios, producing materials that range from purely diffuse to purely specular. The results, shown in Figure 3 of the main paper, indicate that lighting estimation is stable to different object materials.

3.2 Real input capture setup

We captured a set of real images from indoor scenes with the ground truth environment light using a panoramic camera, to enable numerical analysis. In practice, we capture the RGBD input with a PointGrey full-HD RGB camera and a PrimeSense depth sensor. We filled holes in the depth map via a push-pull operation, and then apply a bilateral filter to remove noise. We use KinectFusion to find correspondence between frames, so that we can insert a virtual object and track it between video frames.

4 Results

Here, we include additional results and comparisons.

Synthetic validations Figure 3 exhibits selected results on our synthetic test set. Figure 4 illustrates results of compositing virtual objects into a real measured input image. Rendered images with the ground truth environment map are provided for reference.

Local interreflection Our training and test datasets are rendered with a physically based renderer using only a single light bounce, the global illumination effects for the surrounding objects are included in the environment map, the local interreflection within the local object is ignored. To validate how those ignored local interreflection affects our result, we prepared input image rendered with and without local interreflection and test our method on both inputs. As shown in Figure 1 Since the dominating global illumination effects are already included in the environment map, the renderings with and without local interreflection are very similar to each other. The similar inputs also leads to similar prediction results.

We also numerically analyzed the effect of local interreflections, for the synthetic test set rendered with local interreflection, the render and light RMSE

are 0.070 and 1.426. Compare to RMSE on test set without local interreflection 0.068 and 1.419, we find a 3% increase of relative error, which can be further reduced if we also render the training data with GI.

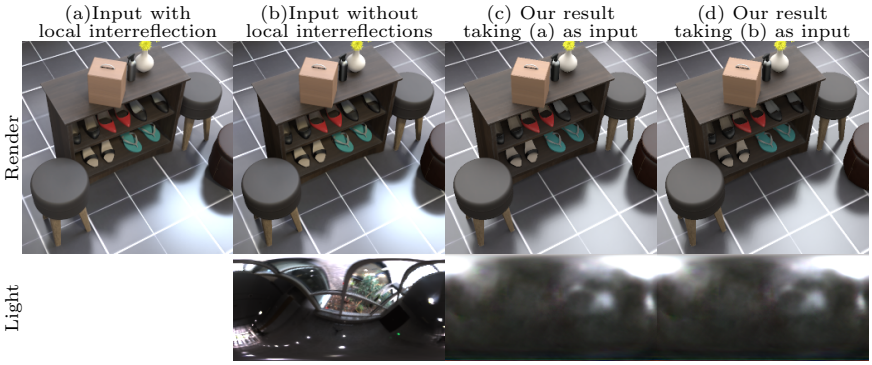

Fig. 1. Synthetic data rendered with and without local interreflection produces similar images, since the environment map already contains interreflection of the surroundings. Our method produces similar results from both inputs.

Comparisons Figure 2 compares our results with Song et al. [3]. The results of [3] are extracted from their paper, they also provided the ground truth environment light. However, the virtual object model and its material is unavailable and the environment map in their paper has an unknown rotation respect to the view point, as a result, we prepare the reference environment map and virtual object insert (with our own virtual object) based on their provided ground truth environment map instead of compare to the images in their paper.

As shown in Figure 2, environment map estimated by [3] contains rich details, however the overall distribution of the light is not consistent with the ground truth (the dominating lights are at different locations). This also leads to larger rendering error compared to the ground truth. The environment maps estimated by our method correctly match the distribution of the ground truth and the rendering results of virtual object insert produce smaller error.

Figures 5 - 6 show a series of comparisons to state-of-the-art lighting estimation methods on real measured input images captured by [2].

Spatially-varying illumination Our object-based lighting estimation can be easily extended to support spatially-varying illumination effects, such as near-field illumination, by taking different local regions of the input image. Figure 7 illustrates an example scene lit by one near-field area light source. Our method with different spatial crops from the input image correctly estimates such spatially-varying illumination effects. The two toy airplanes inserted into the scene cast shadows towards the correct direction compared to the reference rendered with ground truth local lighting. Please refer the supplementary video for a dynamic

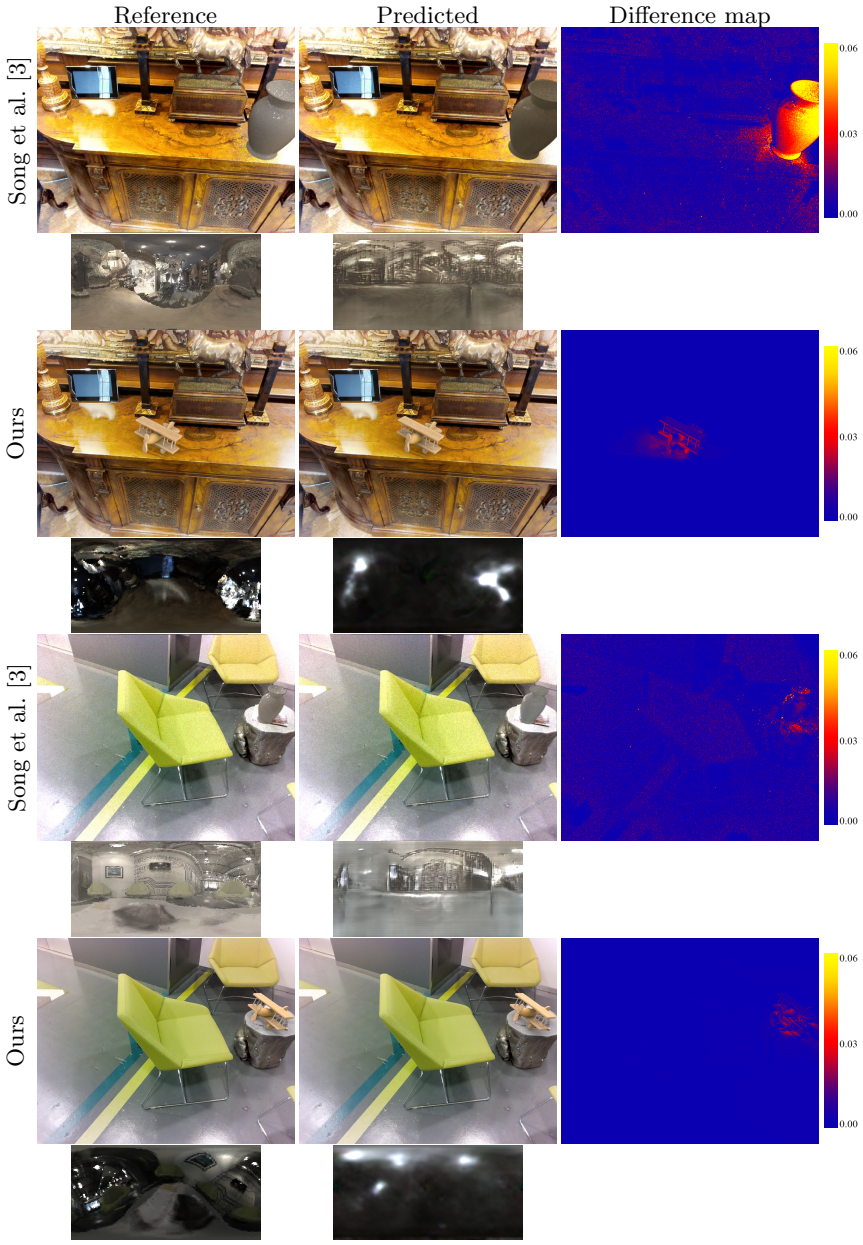

Fig. 2. Comparison with Song et al. [3]. Although [3] produces environment maps with rich details, the overall distribution of the light is wrong which produces inconsistent shading results comparing to the reference. Our method correctly determines the position of the light sources, the rendering results with our predicted light has lower error compared to results of [3].

local illumination example, showing temporal coherent estimation of such local illumination effects.

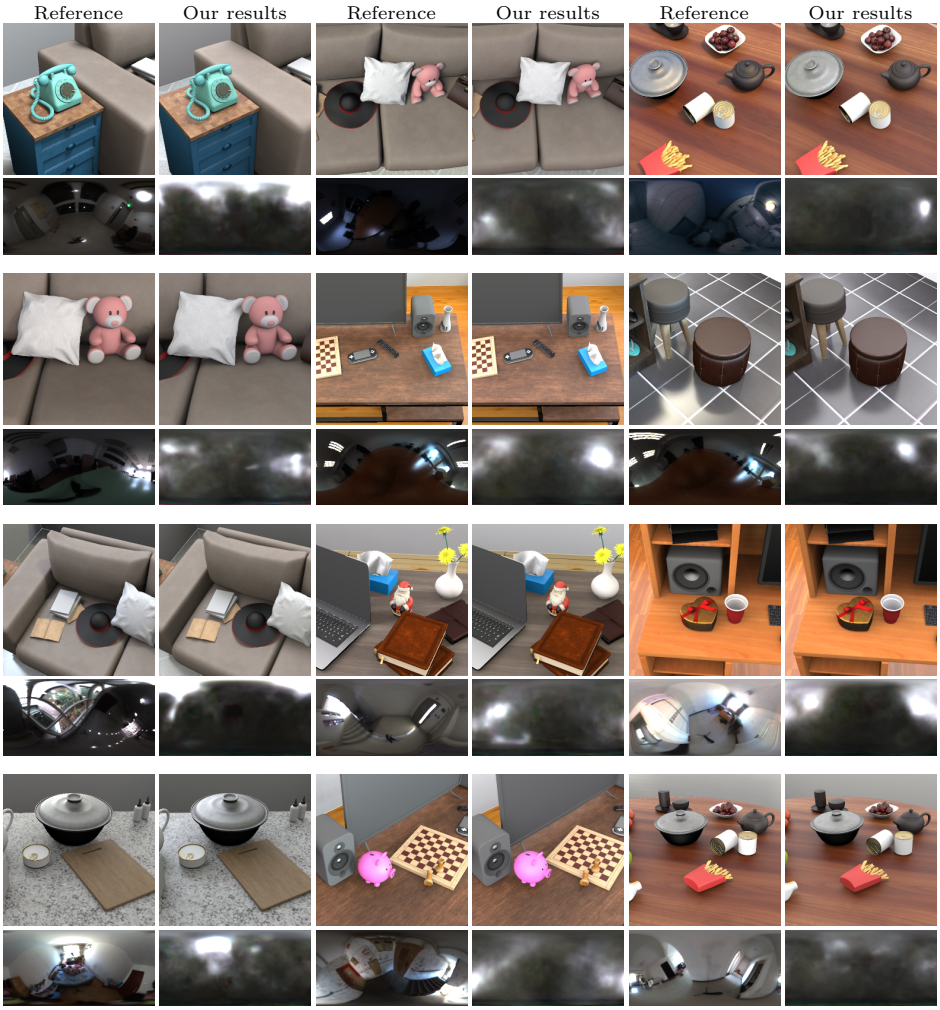

Fig. 3. Selection of evaluation results on our synthetic test set, with the rendering results with ground truth and estimated light shown at the top and the reference and estimated environment light shown at the bottom. The first column shows results under environment maps containing a *single dominant light*; the second row shows results under *multiple dominant lights*; the third row shows results under *large area lights*; the forth row shows results under *near ambient light*.

Fig. 4. Real captured examples with comparison to measured ground truth lighting. We compare the estimated environment map against measured ground truth, as well as results rendered with our lighting and ground truth lighting. Please refer to the input RGB image to determine which object is virtually inserted.

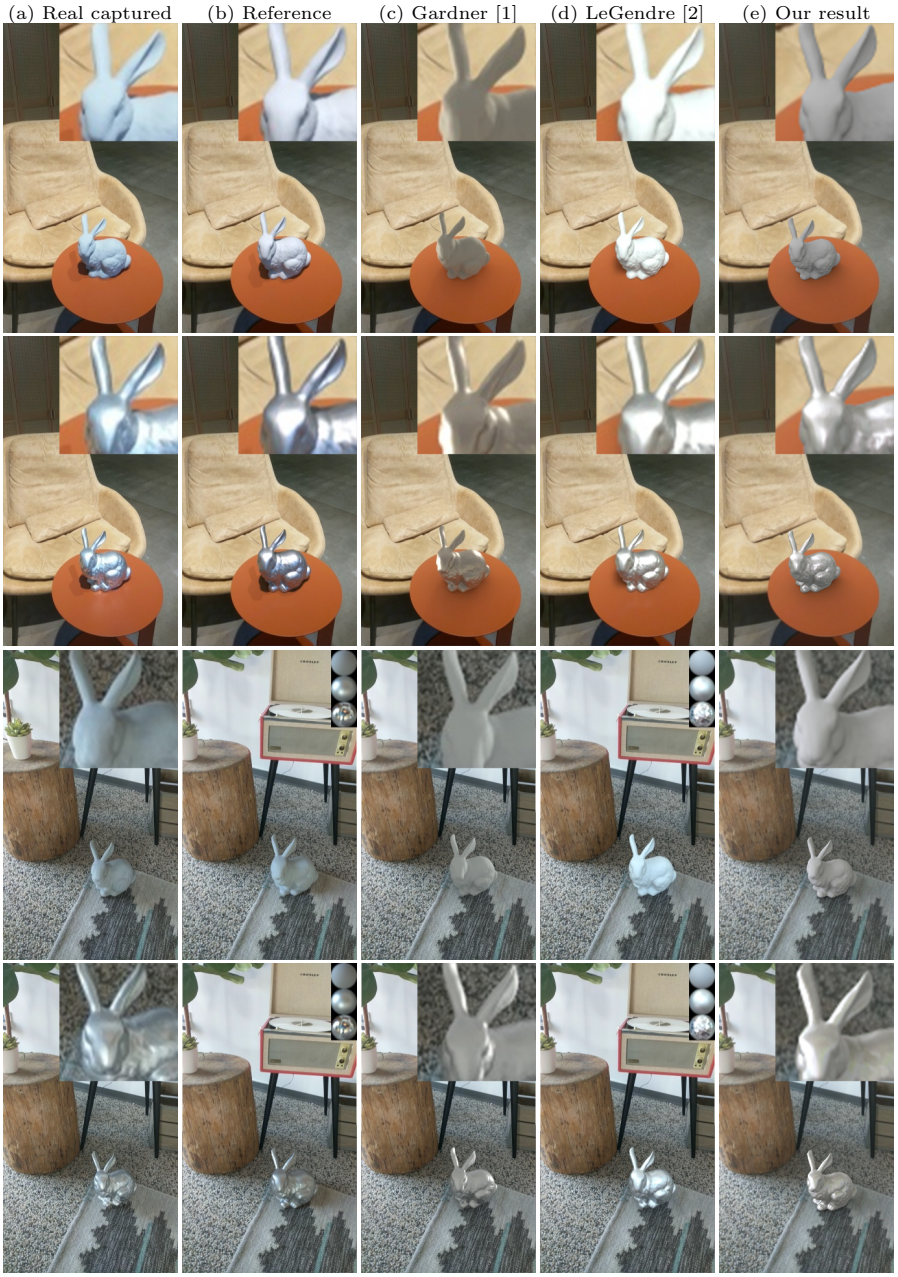

Fig. 5. We compare our method to existing lighting estimation methods. (a) The photograph of a real 3D-printed bunny placed in the scene. Rendering results of a virtual bunny under captured ground truth environment map (b), environment map estimated by (c) Gardner et al. [1], (d) LeGendre et al. [2] and (e) our method.

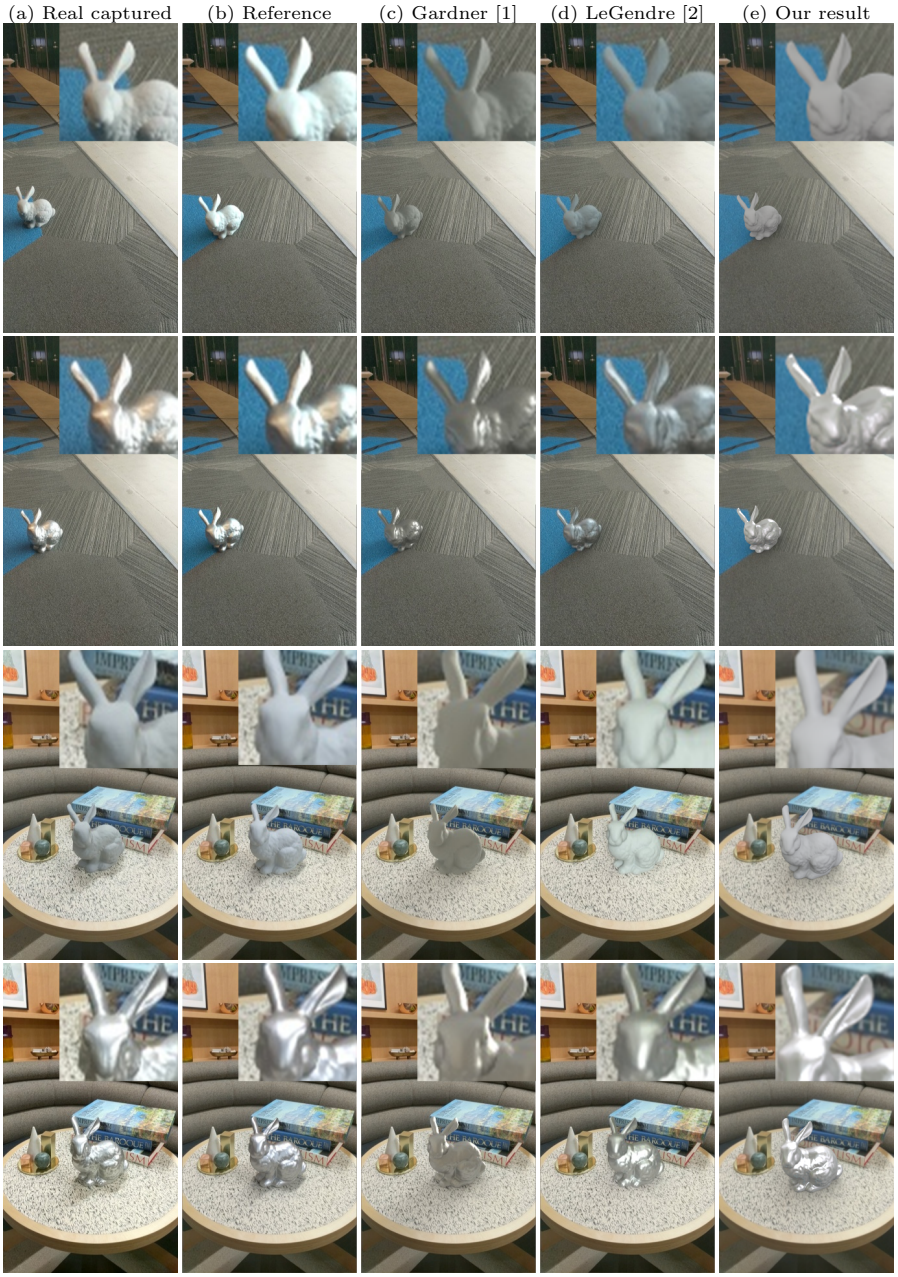

Fig. 6. We compare our method to existing lighting estimation methods. (a) The photograph of a real 3D-printed bunny placed in the scene. Rendering results of a virtual bunny under captured ground truth environment map (b), environment map estimated by (c) Gardner et al. [1], (d) LeGendre et al. [2] and (e) our method.

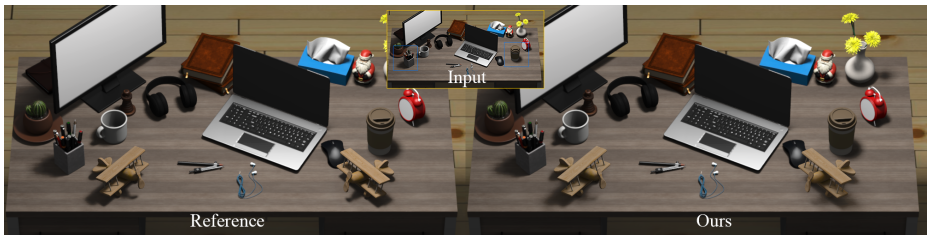

Fig. 7. Spatially-varying illumination result by estimating light with different spatial crops from the input image (marked in blue). The virtually inserted toy planes exhibit shading consistent with the input, as the left plane casts shadow to the left, like the nearby pen stand, and the right plane casts shadow towards the right, similar to the nearby mug. Such spatially varying lighting effects are typical of near-field illumination.

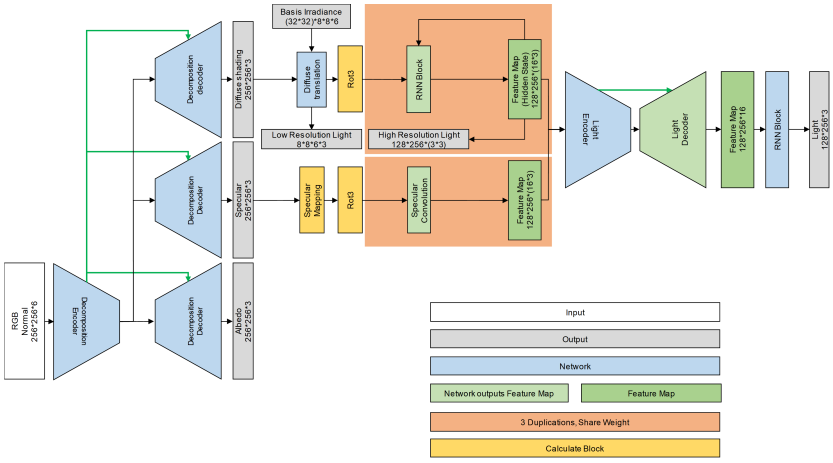

Fig. 8. Overview of our lighting estimation networks. The detailed network structures are illustrated in the following figures.

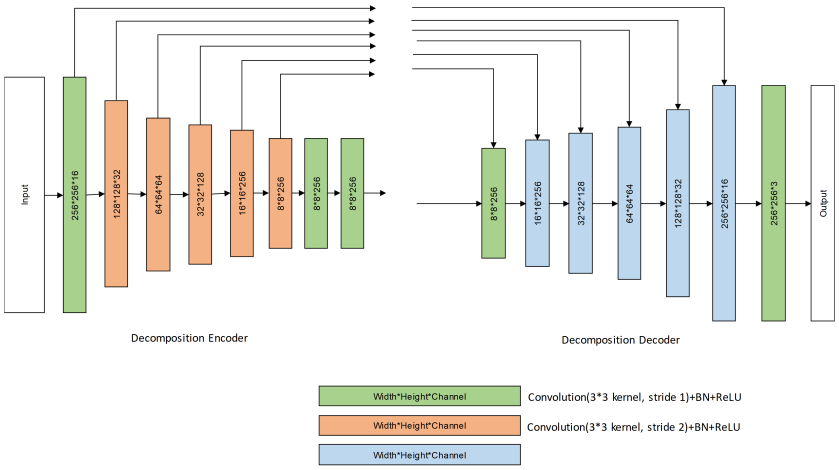

Fig. 9. Detailed network structure of our decomposition network.

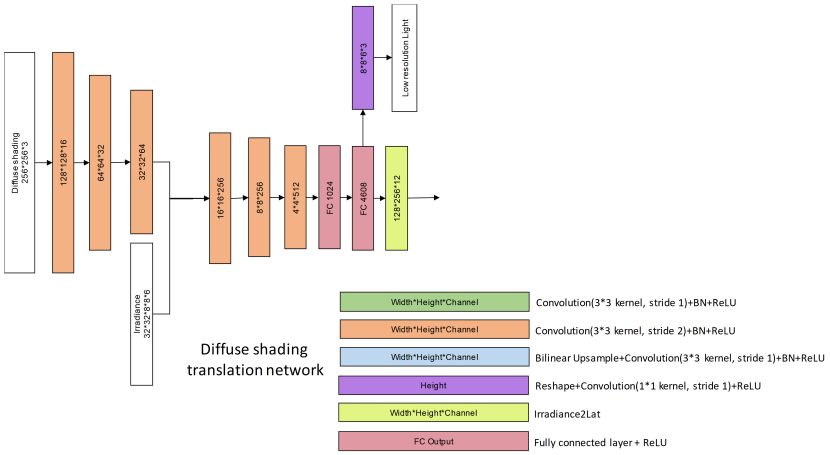

Fig. 10. Detailed network structure of our diffuse shading translation network.

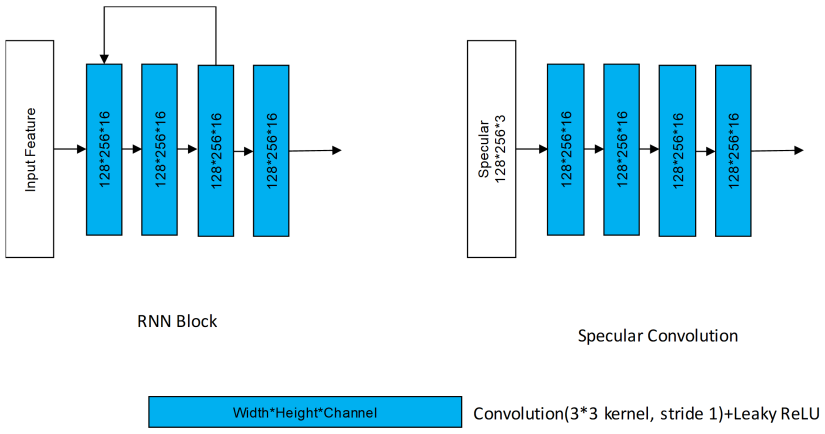

Fig. 11. Detailed network structure of our specular shading translation network, and recurrent convolution block.

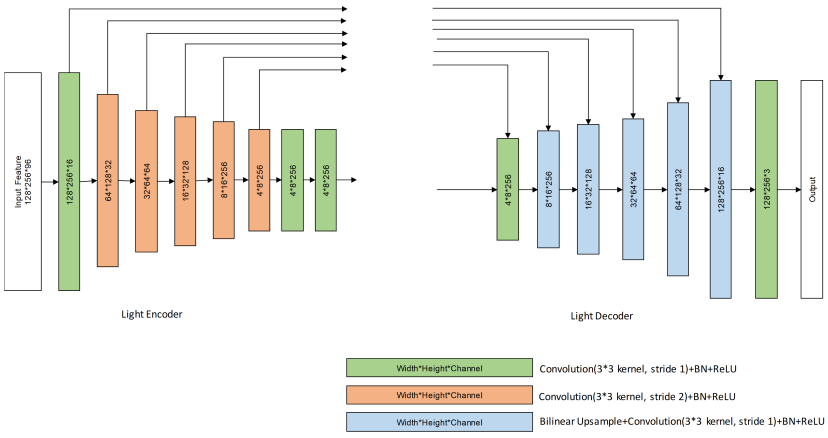

Fig. 12. Detailed network structure of our fusion network.

References

1. Gardner, M.A., Sunkavalli, K., Yumer, E., Shen, X., Gambaretto, E., Gagné, C., Lalonde, J.F.: Learning to predict indoor illumination from a single image. ACM Transactions on Graphics **36**(6), 1–14 (nov 2017)
2. LeGendre, C., Ma, W., Fyffe, G., Flynn, J., Charbonnel, L., Busch, J., Debevec, P.E.: Deeplight: Learning illumination for unconstrained mobile mixed reality. In: CVPR (2019)
3. Song, S., Funkhouser, T.: Neural illumination: Lighting prediction for indoor environments. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)