Lightweight Bilateral Convolutional Neural Networks for Interactive Single-bounce Diffuse Indirect Illumination

Hanggao Xin, Shaokun Zheng, Kun Xu, Ling-Qi Yan

Abstract—Physically correct, noise-free global illumination is crucial in physically-based rendering, but often takes a long time to compute. Recent approaches have exploited sparse sampling and filtering to accelerate this process but still cannot achieve interactive performance. It is partly due to the time-consuming ray sampling even at 1 sample per pixel, and partly because of the complexity of deep neural networks. To address this problem, we propose a novel method to generate plausible single-bounce indirect illumination for dynamic scenes in interactive framerates. In our method, we first compute direct illumination and then use a lightweight neural network to predict screen space indirect illumination. Our neural network is designed explicitly with bilateral convolution layers and takes only essential information as input (direct illumination, surface normals, and 3D positions). Also, our network maintains the coherence between adjacent image frames efficiently without heavy recurrent connections. Compared to state-of-the-art works, our method produces single-bounce indirect illumination of dynamic scenes with higher quality and better temporal coherence and runs at interactive framerates.

Index Terms—Real-Time Rendering, Global Illumination.

1 Introduction

Global illumination is vital in physically-based rendering since it provides photo-level realism. Conventionally, physically correct global illumination is generated using offline rendering techniques such as Monte Carlo path tracing. However, it is time-consuming and takes minutes or even hours to generate a single frame. Recently, with the rapid development of GPUs, real-time/interactive global illumination becomes a hot topic in graphics and has achieved great success.

We classify existing interactive global illumination methods into three types. The first type of works are approximate methods which aim at real-time/interactive performance. However, such methods usually have limited scopes. For example, reflection mapping [3] is limited to glossy reflections only, screen space directional occlusion (SSDO) [2] is limited to local effects, and voxel cone tracing [4] requires a large amount of storage. The second type of works are pre-computation based methods, such as lightmap baking [5], precomputed radiance transfer (PRT) [6], [7], [8], [9] and precomputed light probes [10]. Most of the pre-computation based methods assume static lighting condition or static scene geometry, thus are limited to certain types of applications only. The third type of works exploit sparse sampling and filtering [11], [12] to reconstruct clean and plausible global illumination from noisy images rendered with a low sample rate, e.g., 1 to 32 samples per pixel (spp). However, they still cannot achieve real-time/interactive performance.

A recent trend is to use deep neural networks to predict approximate global illumination, such as the denoising works [13], [14] that help to remove Monte Carlo noise, and deep shading [1] that generates various local shading effects. These methods have unique advantages. They only use screen space information, require no pre-computation, and can be used as a black box. However, to achieve plausible results, the neural networks often have to be deep and complex, which bears a heavy burden to the performance. The performance might not be a problem for offline applications but is extremely crucial in our interactive rendering application for practical use.

To address this issue, we present an interactive novel screen space method that predicts single-bounce indirect illumination, requiring only direct illumination and auxiliary features as input. We keep in mind that our neural network needs to be lightweight to guarantee interactive performance. To achieve this, we design our neural network as follows. First, we introduce a convolution operation that explicitly performs bilateral filtering, to directly feed the depth differences to the neural network. Second, inspired by recent video processing works [15], we avoid heavy recurrent connections and force the neural network to minimize temporal differences by considering optical flow during the training stage. In this way, our method maintains temporal coherence while it remains lightweight and does not introduce run-time overhead. Besides, we integrate our neural network into an OpenGL rendering framework, so that no GPU↔CPU data transfer is further needed. With all these efforts, we can produce high-quality single-bounce indirect illumination for dynamic scenes, while still maintaining interactive performance with all other costs (direct illumination, shadows, etc.) considered together.

To briefly summarize, our main contributions include:

- the first fully predicted single-bounce indirect illumi-

Hanggao Xin, Shaokun Zheng, Kun Xu are with BNRist, Department of Computer Science and Technology, Tsinghua University, Beijing, China. Kun Xu is the corresponding author, email: xukun@tsinghua.edu.cn.
Ling-Qi Yan is with Department of Computer Science, UC Santa Barbara, United States.

Input direct illumination (0.3280) | Deep shading / SSDO (0.0836) / (0.2815) | Our global illumination (0.0784) | Reference

Fig. 1. Our method can predict high-quality global illumination with a resolution of 1024×768 from direct illumination and auxiliary features such as normals and 3D positions. Our method runs at interactive framerates (45.3 FPS in this case) and generates much more plausible results compared with deep shading [1] and screen space directional occlusion [2]. The numbers above indicate the differences to the reference using the 1-SSIM metric.

nation method that achieves interactive performance (> 30 fps) with a resolution of 1024×768;
- a specifically designed bilateral convolutional neural network with newly introduced bilateral convolutional layers.

2 Related Work

Neural networks in rendering. Neural networks have been proven as powerful tools in general, and have also been successfully applied to rendering. Ren et al. [8] use a shallow neural network as an efficient representation of the precomputed data of radiance transfer. Kallweit et al. [16] approximate multiple scattering of light within the volumes of clouds by using a fully connected neural network to approximate light transport from examples. Besides, researchers have employed neural networks in applications such as supersampling [17] and anti-aliasing [18].

Screen space rendering can be regarded as post-processing techniques that only utilize screen space information to generate rendering effects and are very successful in real-time applications. Dachsbacher and Stamminger [19] produce plausible indirect illumination by adaptive sampling on a reflective shadow map. Screen space ambient occlusion (SSAO) [20] generates approximate ambient occlusion effects in real-time. Screen space directional occlusion (SSDO) [2] extends the idea of SSAO to introduce interreflections between objects, but is limited to local indirect illumination only. Deep shading [1] uses deep Convolutional Neural Networks (CNNs) to produce screen space shading effects such as SSAO/SSDO and depth-of-field, but only achieves interactive performance and still cannot capture global effects. Besides, all these methods mentioned above only work for single frames and do not consider temporal coherence between adjacent frames. Specifically, deep shading [1] produces flickering artifacts when taking an image sequence as input.

Temporal coherence is important when a continuous sequence of images is being processed. Sudden changes between adjacent frames can be easily observed by human eyes and are thus objectionable. To guarantee smooth transitions between frames, Huang et al. [15] use optical flow to detect pixel correspondences between adjacent frames. Longer ranges of temporal coherence can also be achieved with non-local networks [21] or recurrent networks [22], [23], but these methods are often too costly for real-time applications, and require long image sequences in the training set.

Monte Carlo rendering. Researchers have always been investigating techniques to enhance the performance of Monte Carlo ray tracing [11], [24], [25], [26], [27]. Generally, they render noisy images at a low sampling rate and obtain high-quality images through a filtering or reconstruction step. Axis-aligned filtering [24] and fast sheared filtering [11] utilize frequency analysis to design theoretically optimal sizes and shapes of filters. Non-local filtering based methods [28], [29] consider contributions from spatially non-adjacent regions for more information. Other methods work with even lower sampling rates ($1 \sim 2$ spp) by exploiting temporal coherence to increase the effective sample count [30].

Neural networks are also widely adopted for Monte Carlo denoising. Chaitanya et al. [14] propose a denoising method for image sequences based on a recurrent autoencoder. It maintains temporal stability, but the recurrent connections introduce performance overhead and require long sequences of rendered images as training data. Bako et al. [13] propose a kernel-predicting neural network targeted at denoising single images. However, the network is rather complicated. Also, neither methods can achieve real-time performance, due to the slow inference caused by the complex neural network structures.

Bilateral modules have been used to aid neural network design in the computer vision field. Inspired by bilateral filtering, Li et al. [31] excavate features from target and guidance images separately and concatenate those features directly for remaining learning passes. Apart from guiding neural networks to learn features well, bilateral filtering modules have been used to improve performance in mobile image processing [32] as an image enhancement technique. To smooth results for image segmentation tasks, Gadde et al. [33] propose bilateral inception module which is used as a layer inserted between existing convolution layers and does not affect the convolution process explicitly. Our bilateral convolutional layer is different from theirs [33] in several aspects. First, bilateral inception is added to existing CNN layers while we directly modify CNN layers; second, bilateral inception is only used as the subsequent layer of CNN layers while we use bilateral convolutional structures for all layers; third, bilateral inception is limited to bilateral filtering (i.e. only considering color differences) while our

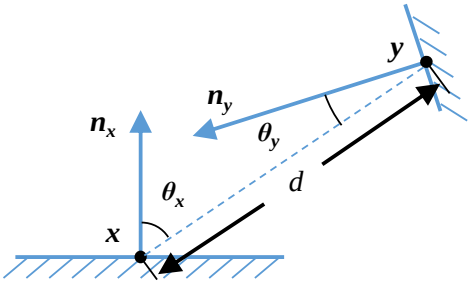


Fig. 2. Illustration of the geometry term.

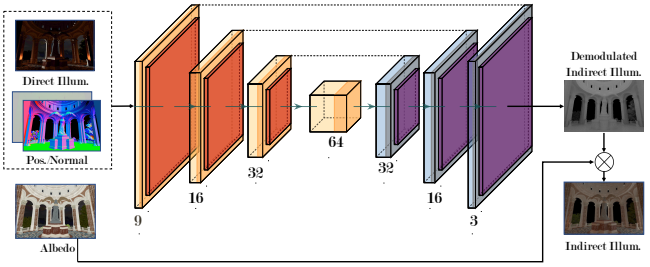


Fig. 3. The architecture of our neural network. Note that all layers in the encoder and the decoder are bilateral convolutional layers. Skip connections are marked by dashed lines between corresponding layers. Direct illumination, normal and camera space 3D positions are fed to the neural network as input. The output demodulated indirect illumination is multiplied by the albedo to obtain the final indirect illumination.

layer is used for joint bilateral filtering.

Jampani et al. [34] introduced bilateral filters onto CNN layers based on a sparse high dimensional permutohedral lattice data structure. The permutohedral lattice is defined in the feature space, which is usually a 5D space combining both 2D positions and 3D colors. To apply a bilateral filter to an input image, 3 steps are involved. First, they splat the values of each pixel into its enclosing lattice points. Second, they convolve the values on the lattice. Finally, they retrieve back the values of each pixel from its enclosing lattice points through interpolation. They also derived gradients for the filter weights, hence the filter weights could be end-to-end trained through backpropagation. However, their method is too slow to be used in real-time/interactive rendering applications.

3 Design Principles

Our goal is to generate indirect illumination from only direct illumination and auxiliary features of the 3D scene (i.e., G-buffers) using neural networks. To achieve interactive performance, we need to keep our neural network lightweight. Below, we will review the light transport of indirect illumination, to help decide which auxiliary features to choose as input to the network.

To make the problem simpler, we assume the BRDFs of all scene objects are diffuse, and we only consider single-bounce indirect illumination. As shown in Fig. 2, the single-bounce indirect illumination at a shading point $\mathbf{x}$ could be written as an integration of direct-to-indirect light transport:

$$L_o(\mathbf{x}) \propto \int L_d(\mathbf{y} \to \mathbf{x}) V(\mathbf{y} \to \mathbf{x}) G(\mathbf{y} \to \mathbf{x})\, \mathrm{d}\mathbf{y}. \quad (1)$$

This is an integration over all surface points $\mathbf{y}$ (reflector) in the scene, where L_d denotes direct illumination contribution received at $\mathbf{y}$, V is the visibility between $\mathbf{y}$ to the shading point $\mathbf{x}$, and $G = (\cos\theta_\mathbf{x} \cos\theta_\mathbf{y})/d^2$ is known as the geometry term. Note that, the exitant direction and the BRDF of $\mathbf{x}$ are both omitted since we assume diffuse BRDFs.

Taking a closer look at the geometry term G in Fig. 2, we immediately find that the two cosine terms are related to the angles between the incident direction that connects $\mathbf{x}$ and $\mathbf{y}$ and the surface normals at $\mathbf{x}$ and $\mathbf{y}$, respectively. Besides, the distance d between $\mathbf{x}$ and $\mathbf{y}$ also significantly affects the result.

The observation of the geometry term G above leads to two immediate conclusions. First, we do need the per-pixel surface normals to determine how closely faced two pixels are. Second, since the contribution of indirect illumination is determined by the world space distance between two pixels, it is important to provide the actual 3D positions to each pixel as a position buffer. Besides, depths are still important for determining discontinuities. Instead of introducing an additional depth buffer, we modify our position buffer such that the 3D positions are stored in canonical/camera space. In this way, we can easily obtain the depth (which is the z component of the 3D position) and still able to compute the world space distance between two pixels.

Now we take a look at the direct illumination term L_d, which is the primary input of our method. Note that different $\mathbf{y}$ may have different textures/albedos and maybe shadowed or not, and these differences will all affect the direct-to-indirect transport onto $\mathbf{x}$ as part of the L_d. Hence, we do not need to separate textures, shadows from direct illumination, and we directly use full direct illumination as input. On the output side, we follow irradiance filtering works [35] to decouple shading from texture/albedo. Specifically, we predict the demodulated (texture/albedo removed) indirect illumination. This is because the demodulated indirect illumination is usually more smooth, and more importantly, the final indirect illumination can be correctly computed by a simple multiplication of the demodulated indirect illumination with the albedo at every shading point $\mathbf{x}$.

Summarization. With the above theoretical analysis, the input to our neural network includes: full direct illumination (with textures and shadows), surface normals and camera space 3D positions. The output is the demodulated indirect illumination (without multiplying albedo).

4 Our Approach

In this section, we will discuss the architecture of our network and the loss function used to train our network.

As discussed in Sec. 3, the distance d between two points $\mathbf{y}$ and $\mathbf{x}$ significantly affects the indirect illumination contribution from $\mathbf{y}$ to $\mathbf{x}$. To encode the effect of distance d, a possible choice is to use standard convolutional layers and implicitly encode such relationships into the network through parameter training. However, this will potentially result in a complex and deep network that cannot achieve interactive framerates. Instead, we introduce *bilateral convolutional layers*, to explicit encode the effect of distance d.

4.1 Bilateral convolutional layers

A traditional convolutional layer is generally defined as:

$$I_{\text{out}}(\mathbf{i}) = I_{\text{in}}(\mathbf{i}) \otimes W(\mathbf{i}, \mathbf{j}), \tag{2}$$

where $\mathbf{i}$ and $\mathbf{j}$ are different locations on a 2D image, $\otimes$ denotes the convolution operator, W is the convolution kernel. I_{in} and I_{out} denote the input and output images of the layer, respectively.

We modify the above equation by explicitly taking depth into account:

$$I_{\text{out}}(\mathbf{i}) = I_{\text{in}}(\mathbf{i}) \otimes \left(W(\mathbf{i}, \mathbf{j}) \odot \exp\left(-(z_{\mathbf{i}} - z_{\mathbf{j}})^2/\sigma^2\right)\right), \tag{3}$$

where $\odot$ denotes element-wise multiplication, $z_{\mathbf{i}}$ and $z_{\mathbf{j}}$ are the depth values at $\mathbf{i}$ and $\mathbf{j}$, respectively, and σ is a parameter to control the influence of depth.

In Equation 3, the original kernel weights W are modulated by a Gaussian that takes account depth differences, similar to applying joint bilateral filtering [36], [37] using depth as guidance. Hence, we name our modified convolutional layer as a *bilateral convolutional layer*. Recall that the input 3D positions are stored in camera space so that the depth values could be directly obtained from the z-component of the 3D position. The intuitive motivation of our bilateral convolutional layers is to remove the influence of neighboring pixels with a large depth difference.

In our implementation, we fix the parameter $\sigma = 1$. We have tried other parameter settings. For example, set the value of σ to be higher for deeper layers, or set it as an unknown parameter and let the network learn it. The results do not differ much.

4.2 Bilateral convolutional neural networks

Now we incorporate the newly introduced bilateral convolution layers to design the architecture of our bilateral convolutional neural networks (BCNNs). As shown in Fig. 3, our network uses a U-Net architecture, with 3 encoder layers and 3 decoder layers. For each layer, we replace the original convolutional layer with our bilateral convolutional layer (Sec. 4.1). That is, each layer in our network applies bilateral convolution to the output of the previous layer, then passes through the pooling function as input of the subsequent layer. The output of each layer in the encoder will also be passed through a skip connection to the corresponding layer in the decoder.

In our implementation, all convolutions have a kernel size of 3×3. Since our bilateral convolutional layer is not a built-in layer in PyTorch, we implemented as a Pytorch extension module, according to Equation 3. The activation layers consist of leaky ReLUs as described by Maas et al. [38], which multiply negative values by a small constant instead of zero.

We concatenate information from our G-buffers (normals and camera space 3D positions) with direct illumination and treat them together as the input channels of the network.Note that the albedo buffer is not used as an input channel, but directly multiplied by the output to get the final indirect illumination.

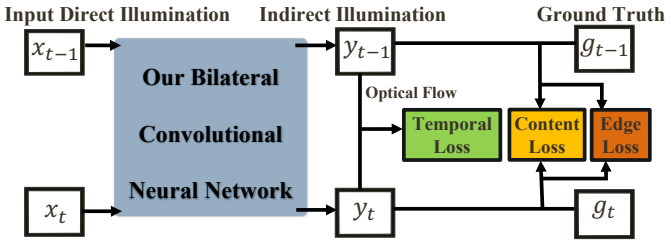


Fig. 4. Loss function. In the training process, adjacent frames will be fed to the network at the same time to maintain the coherence in rendered frames. In the inference process, the network only takes one frame (the current frame) as input and still preserves temporal coherence.

Fig. 5. Examples from our training set. We show global illumination (i.e., direct + single-bounce indirect) for each example. Scene materials are randomly assigned.

4.3 Temporal coherence and loss function

Neural networks based methods generally give approximate solutions to the rendering equation, which is prone to bias. This is usually perceived as temporal flickering artifacts, even with a small amount of adjustment on camera poses and lighting conditions. Since we aim at dynamic scenes, we need to guarantee temporal coherence between adjacent frames. For better runtime performance, we do not use a recurrent structure such as RNN or LSTM. Instead, inspired by recent video processing works [15], we force the neural network to minimize temporal differences after considering optical flow during the training stage. In this way, our method maintains temporal coherence and remains lightweight.

Our loss function $\mathcal{L}$ is defined as the weighted sum of three terms: a content loss $\mathcal{L}_c$, an edge loss $\mathcal{L}_e$ and a temporal loss $\mathcal{L}_t$:

$$\mathcal{L} = \omega_c \mathcal{L}_c + \omega_e \mathcal{L}_e + \omega_t \mathcal{L}_t, \tag{4}$$

where ω_c, ω_e and ω_t are weights to control the relative contributions of different terms. We use $\omega_c = 0.7$, $\omega_e = 0.1$ and $\omega_t = 0.2$ in our implementation, which makes a good trade-off between convergence and quality. The content loss and edge loss are used to constrain the network to produce results close to the ground truth. Specifically, the content loss $\mathcal{L}_c$ is computed as the SSIM difference between our output and the ground truth, and the edge loss $\mathcal{L}_e$ is computed as the L1 difference between the output and the ground truth after applying a Laplacian filter. The temporal loss $\mathcal{L}_t$ is used to maintain the coherence between adjacent

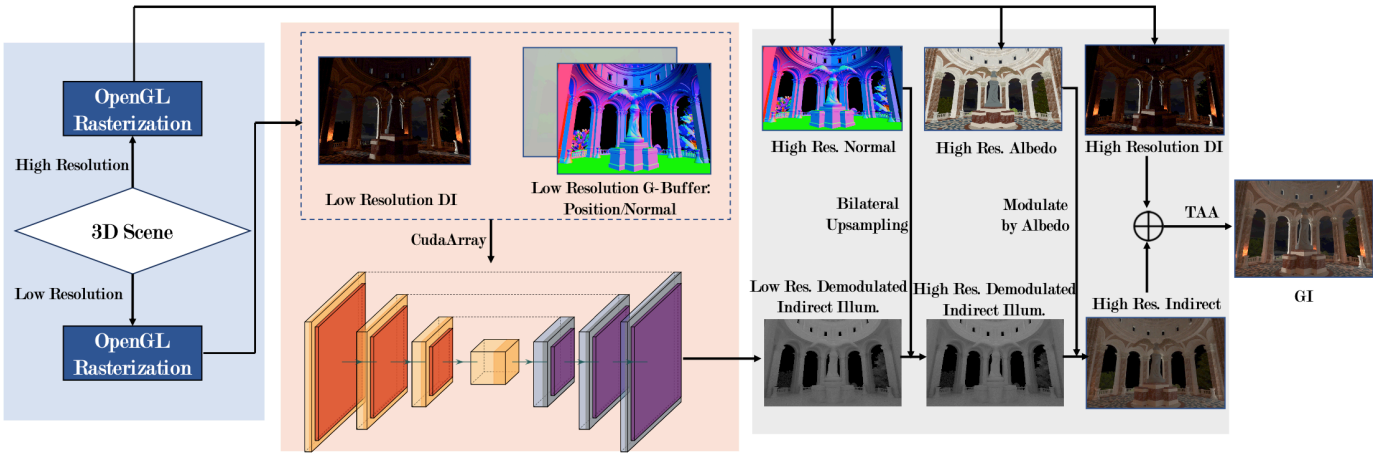


Fig. 6. Our rendering pipeline. We use OpenGL rasterization to generate both high-resolution (i.e., 1024×768) and low-resolution (i.e., 512×384) direct illumination with associated G-buffers. The low-resolution direct illumination and associated G-buffers are fed into our neural network to produce low-resolution demodulated indirect illumination. High-resolution normal/albedo maps are used to generate high-resolution indirect illumination through joint bilateral upsampling and albedo modulation. We then add the upsampled indirect illumination with the high-resolution direct illumination, and further apply Temporal Anti-Aliasing (TAA) to obtain the final high-resolution global illumination results.

frames, and is defined as the L1 difference between two consecutive outputs using optical flow:

$$\mathcal{L}_t = \|f_{\text{warp}}(y_{t-1}) - y_t\|_1, \tag{5}$$

where y_{t-1}, y_t represents the outputs of two consecutive frames, and f_{warp} indicates the warping obtained by optical flow, which is used to eliminate the motion between the two frames. We use the TV-L1 algorithm [39] to calculate the optical flow of adjacent frames.

As shown in Figure 4, during training, our network accepts a pair of temporally consecutive frames x_{t-1} and x_t, both containing only the direct illumination. The two frames are fed into the networks separately, and produce two output images y_{t-1} and y_t, which are the predicted indirect illumination. We then use Equation 5 to compute the temporal loss and then the final loss (Equation 4).

Note that during runtime, our network takes only one frame as input, and no optical flow computation is needed. Consequently, there is no runtime overhead to maintain temporal coherence.

5 Implementation Details

In this section, we explain details on data generation (Sec. 5.1), network training (Sec. 5.2), and the runtime rendering pipeline (Sec. 5.3). We start by scaling each scene, setting its longest edge of its bounding box to 40 units, in order to ensure all scenes have similar scales.

5.1 Data generation

As described in Sec. 4.3, we need adjacent frames for training. Hence, our training dataset is composed of pairs of consecutive frames. Each pair includes two frames rendered with slightly changed camera position or orientation. Each frame includes the direct illumination, associated G-buffers (normals, camera space 3D positions, and albedos), and the ground truth single-bounce indirect illumination. All of them are generated without anti-aliasing. To avoid overfitting, the training dataset should be large and representative. Hence, we have collected 92 complex scenes. For each scene, we further randomly sample about two thousand cameras. For each sampled camera, we generate another camera by randomly and slightly changing its position or orientation for pairing. We then render two frames of the scene from the two cameras using path tracing with NVIDIA OptiX [40]. All images are rendered with a resolution of 512×384 with 2048 to 4096 spps. Rendering of one single frame takes about 40 to 100 seconds. To further increase diversity, we render the scenes with random material colors.

After that, we manually check all frames and discard unsatisfactory ones (For example, those rendered under cameras very close to a scene object). In total, 47,332 pairs of frames are collected as training data.

Our dataset provides sufficient variations of global illumination effects such as soft shadow and color bleeding. Besides moving cameras, our dataset also includes dynamic lighting. The variety of our dataset enables our neural network to predict indirect illumination for complex scenes and to handle moving cameras, dynamic lighting, and moving objects as well.

5.2 Training

Training of our network is implemented using PyTorch on a server with 4 NVIDIA GeForce GTX 1080Ti GPUs. The training process takes 1000 epochs and 125 hours in total. The trained network occupies about 1.5G memory.

In the training process, we use Xavier initialization [41] to initialize the weights in our network. We use batch size of 8 and Adam optimizer [42] with learning rate 0.0001 and momentum 0.9 and decay weights 0.95 for back propagation. As Maas et al. [38] suggested, we use leaky ReLUs with $\alpha = 0.1$ instead of linear ReLUs in our implementation. Recall that in our training process, two adjacent frames are fed to the network to guarantee temporal coherence. However, in the inference stage, only a single frame is taken as input, and temporal coherence can still be maintained.

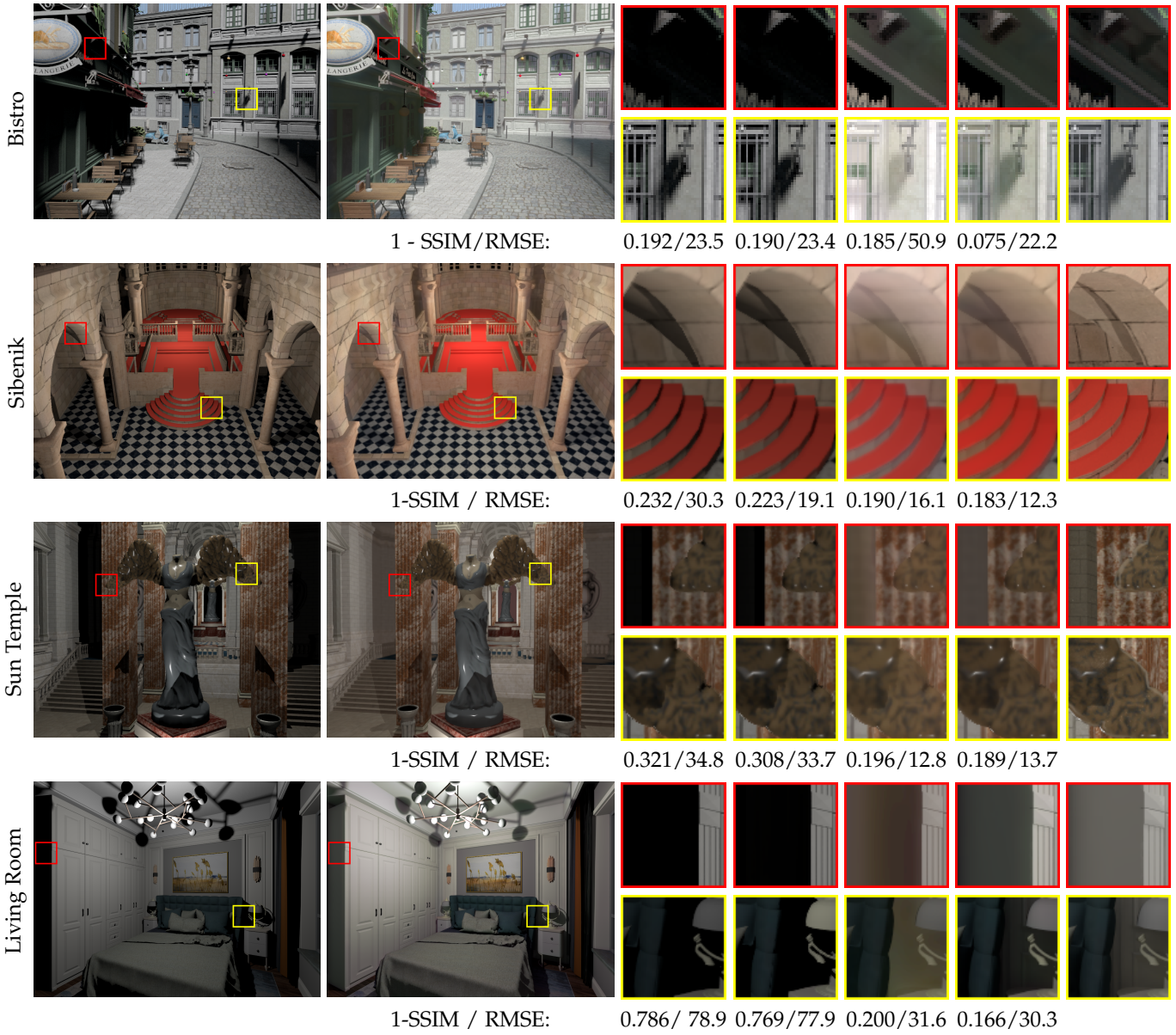


Fig. 7. Comparison of our method with different rendering methods and reference global illumination generated by path tracing. For each scene, from left to right, we provide the input direct illumination, our global illumination result, rendering closeups of direct illumination, SSDO, deep shading, our method, and reference, respectively. We also provide the error of rendering result by each method using metrics including SSIM (left) and RMSE (right). Compared to SSDO and deep shading, our method achieve results with the best visual quality and with the lowest quantitative errors.

5.3 Rendering pipeline

Once our network is trained, we integrate it into a rasterization pipeline, where each image is processed immediately during runtime after being rendered. In this way, our method could be easily combined with existing real-time rendering methods. The whole pipeline is shown in Fig. 6.

We use OpenGL to generate direct illumination along with the G-buffers, including both high-resolution (i.e., 1024×768) and low-resolution (i.e., 512×384) ones. In practice, the low-resolution images do not need to be rasterized again. They are simply downsampled from high-resolution ones, therefore taking negligible time to be generated. In our implementation, for demonstration purposes, direct illumination is computed using a few point lights or directional lights. Soft shadows are calculated using the percent closer soft shadow (PCSS) algorithm [43]. More direct illumination effects could be easily included by applying more sophisticated direct illumination algorithms.

The low-resolution direct illumination and G-buffers are immediately forwarded as the input of our network. Data conversion is required between OpenGL and our network since they use different data formats (i.e., OpenGL framebuffer objects and CUDA arrays). Fortunately, the conversion between them is simply a pointer redirection/reinterpretation inside the GPU, so there is no actual data transferring.

We incorporate joint bilateral upsampling [44], [45], guided by the high-resolution normal map, to scale up the

TABLE 1
Performance Table. For each scene, from left to right, we provide the name, the number of vertices, rendering frame rates, time breakdown for direct illumination, network inference, and post-processing, respectively.

scene	#vert.	fps	dir.tm.	inf. tm.	post. tm.
Bistro	8.4M	36.7	8.0ms	18.2ms	1.0ms
Chess	128k	48.2	1.5ms	18.2ms	0.9ms
Classroom	311k	46.9	1.2ms	19.2ms	0.9ms
Conference	194k	47.6	2.1ms	17.9ms	1.0ms
Gallery	499k	45.6	2.9ms	18.1ms	0.9ms
Pinkroom	2.4M	45.3	2.7ms	18.4ms	1.0ms
Living Room	342k	44.6	3.1ms	18.3ms	1.0ms
Living Room 2	4.1M	41.7	4.7ms	18.2ms	1.1ms
Sibenik	2.3M	41.9	4.2ms	18.4ms	1.1ms
Staricase	320k	48.4	1.3ms	18.3ms	1.0ms
Sun Temple	1.8M	25.4	20.7ms	17.7ms	1.0ms

output of neural networks (i.e., the demodulated indirect illumination). It is further modulated by the high-resolution albedo map and added with high-resolution direct illumination to produce high-resolution (i.e., 1024×768) global illumination results.

Note that we use joint bilateral upsampling on the output of our neural network instead of training an end-to-end neural network that includes the upsampling step or another standalone upsampling network. This is because the joint bilateral upsampling is demonstrated to be much faster ($<$1 ms) than neural network upsampling solutions [31], [46] ($>$20 ms), thus is better suited for our interactive requirements.

Recall that we use aliased direct illumination and G-buffers during training in order to avoid ambiguities of physical parameters within each pixel. Feeding these aliased inputs into our neural network will result in aliased global illumination results. To deal with the aliasing issue, we apply temporal anti-aliasing (TAA) [47] to our final global illumination results. It uses fast but accurately calculated motion vectors to find corresponding pixels in the previous frame, and accumulate the information from the previous frame to the current one.

6 Results and Evaluations

We implement our rendering system on a PC with an NVIDIA RTX2080 graphics card with 12GB video memory and test it on a variety of scenes. For all test scenes, as described in Sec. 5.3, we generate indirect illumination with a resolution of 512×384, upsample it to a resolution of 1024×768, then sum it with direct illumination to produce the final global illumination. Note that none of the test scenes appear in the set of training scenes. Some of the scenes are from [48], [49].

The statistics and performance data of all scenes are given in Table 1. Our method achieves interactive performance for a majority of scenes, e.g. 36.7 fps for scene Bistro containing 8.4M vertices. Thanks to our lightweight network architecture, the time for network inferencing is very stable, taking around 18ms for all scenes with different complexity.

6.1 Comparisons and Results

In Fig. 7, we show global illumination results of our methods for various scenes. Those scenes are all challenging scenes for global illumination rendering. The Bistro scene (Fig. 7 first row) is a complex outdoor scene. It is illuminated by a directional light that casts sharp shadows. Our method successfully predicts indirect illumination in the shadowed areas. For complex indoor scenes like Sibenik (Fig. 7 second row), our method can produce the color bleeding effects from the red floor to the back wall. The Sun Temple scene (Fig. 7 third row) contains a glossy statue. Although our method is designed to handle diffuse indirect illumination, it could also be applied to low-frequency glossy scenes without noticeable artifacts. The Living Room scene (Fig. 7 fourth row) is an indoor scene illuminated by a point light. This scene contains large shadowed areas from direct illumination. Our method produces satisfactory results with the help of our bilateral convolution layers.

For comparison, Fig. 7 also provides the reference generated by path tracing and the rendered results of other screen space methods, including SSDO [2] and Deep Shading [1]. We have also measured the difference between the result of each method and the reference using various metrics, including SSIM and RMSE. Overall, our method is superior to alternative methods in terms of both visual quality and quantitative metrics. Compared to SSDO, which is limited to generating indirect illumination in a local range, our method can produce non-local effects. Compared to deep shading, our method generates much cleaner global illumination with fewer artifacts. A convincing comparison is the Living Room scene (Fig. 7 fourth row). In this case, it is very difficult for screen space methods to produce plausible indirect illumination, since the values of direct illumination are mostly zero in a large area on the screen. SSDO generates little indirect contribution to this area, and deep shading introduces artifacts in the dark shadow areas near the image border. In contrast, our method produces results with much higher quality. Besides, our method also maintains much better temporal coherence.

Fig. 8 compares the indirect illumination component between our method and the reference. Our method produces plausible results. However, when looking at the figure closely, we could still notice subtle visual differences between the indirect illumination results of our method and reference. We believe such differences are hard to eliminate using screen space methods.

As shown in Fig. 9, our method is able to handle dynamic scenes well with moving cameras, dynamic lights, and moving objects. Please see the supplemental video to verify that our rendered sequences preserve nice temporal coherence.

6.2 Evaluations

In this subsection, we will evaluate the design choices in our method, including bilateral convolution layers, loss function and input G-buffers.

Bilateral convolutional layers. In Fig. 10, we further compare the results of our bilateral convolutional neural networks (BCNNs) and those generated by traditional CNNs. Both networks use the same settings and are trained using the same dataset, except that BCNNs use our proposed bilateral convolutional layers while CNNs use traditional convolutional layers. From the results, we can find that

Fig. 8. Comparison of indirect illumination results between our method and the reference global illumination generated by path tracing. For each scene, from left to right, we provide the input direct illumination, our indirect illumination, reference indirect illumination, our global illumination, reference global illumination, and the difference image between our global illumination and the reference, respectively.

traditional CNNs easily produce noticeable artifacts, i.e., incorrect colors in shadow areas and blurring around object boundaries. This is because traditional CNNs cannot learn well on how to handle depth differences when we restrict the network to be lightweight (i.e., with a limited number of layers). In contrast, our bilateral convolutional layers explicitly take depth into account thus produces higher-quality results without easily noticeable artifacts.

Now we explain why we do not use the existing lattice based bilateral convolution method [34]. While their method has shown nice results in tasks like CRF inference, image segmentation, and character recognition, it is not suited for real-time/interactive graphics applications. This is because their method is rather slow while rendering requires real-time/interactive feedbacks. The computations in their method are complex, involving three non-trivial computation steps (i.e., splatting, convolution and slicing) on the high-dimensional lattice for applying bilateral filtering to a single layer. In contrast, our method runs much faster since our network structure is much more lightweight. Our bilateral convolution operator (Equation 3) just adds a depth-aware element-wise multiplication to the traditional convolution operator (Equation 2), which incurs little additional cost. Another possible issue of their method lies in that it may be harder to maintain temporal coherence. Since the sparse structure of the permutohedral lattice is constructed according to input image features, adjacent frames will probably generate permutohedral lattices with different structures, which in turn lead to temporally inconsistent results. In contrast, our method is able to preserve nice temporal coherence.

Metric choices for the content loss. The content loss term $\mathcal{L}_c$ in our training loss function (Equation 4) is used to constrain that the final predicted indirect illumination does not differ much from the ground truth. To compute the difference, we may use various metrics, such as L1, L2, and SSIM. Deep shading [1] has demonstrated that the L2 metric is not appropriate since it will lead to blurry results. As shown in Fig. 11, we have experimented three different metrics (i.e., L1, SSIM, and L1+SSIM) to find out which one is the best choice. The results demonstrate that 'using L1 only' or 'using L1+SSIM' could easily produce color-shifting artifacts in dark areas, but 'using SSIM only' produces the best perceived visual quality. Hence, we use the SSIM metric to compute the content loss.

Input maps to the network. As discussed in Sec. 3, the input to our network includes full direct illumination and two auxiliary G-buffers of normals and 3D positions. We have conducted an ablation study to validate the effectiveness of the two auxiliary maps. We have tested four different settings: with both normals and 3D positions, only with normals, only with 3D positions, and without any auxiliary maps. As shown in Fig. 12, the training loss decreases fastest when both normals and 3D positions are taken as input maps (the red curve).

7 Discussion and Failure Cases

Multiple bounces. Although our method is designed for single-bounce diffuse indirect illumination, it could also be easily extended to handle multiple bounces. One way to do so is to repeatedly apply our neural network to predict the next bounce by using the information of the current bounce as input. However, since our method does not explicitly guarantee energy conservation in the prediction, it may produce results that are too bright (Fig. 13 (c)). This problem could be alleviated by attenuating the brightness of each bounce with a predefined attenuation factor. As shown in Fig. 13 (d), we use an attenuation factor of 50%, and the overall brightness is now much closer to reference. Another way is to re-train the network with the same architecture, but using multi-bounce indirect illumination instead of single-bounce indirect illumination as training data. Such an example is shown in Fig. 13 (e). Its quality is slightly lower than Fig. 13 (d), but it does not need any manually provided hyperparameters.

Failure cases. Fig. 14 shows several failure cases of our method. In the Sponza scene, our method produces darker indirect illumination in the area above the gate compared to the reference. This is because the direct illumination in this area is almost entirely in shadow, and it is very hard to predict indirect illumination accurately in such cases. In the Torus scene, our method fails to handle highly specular and refractive objects since it violates our diffuse BRDF assumption.

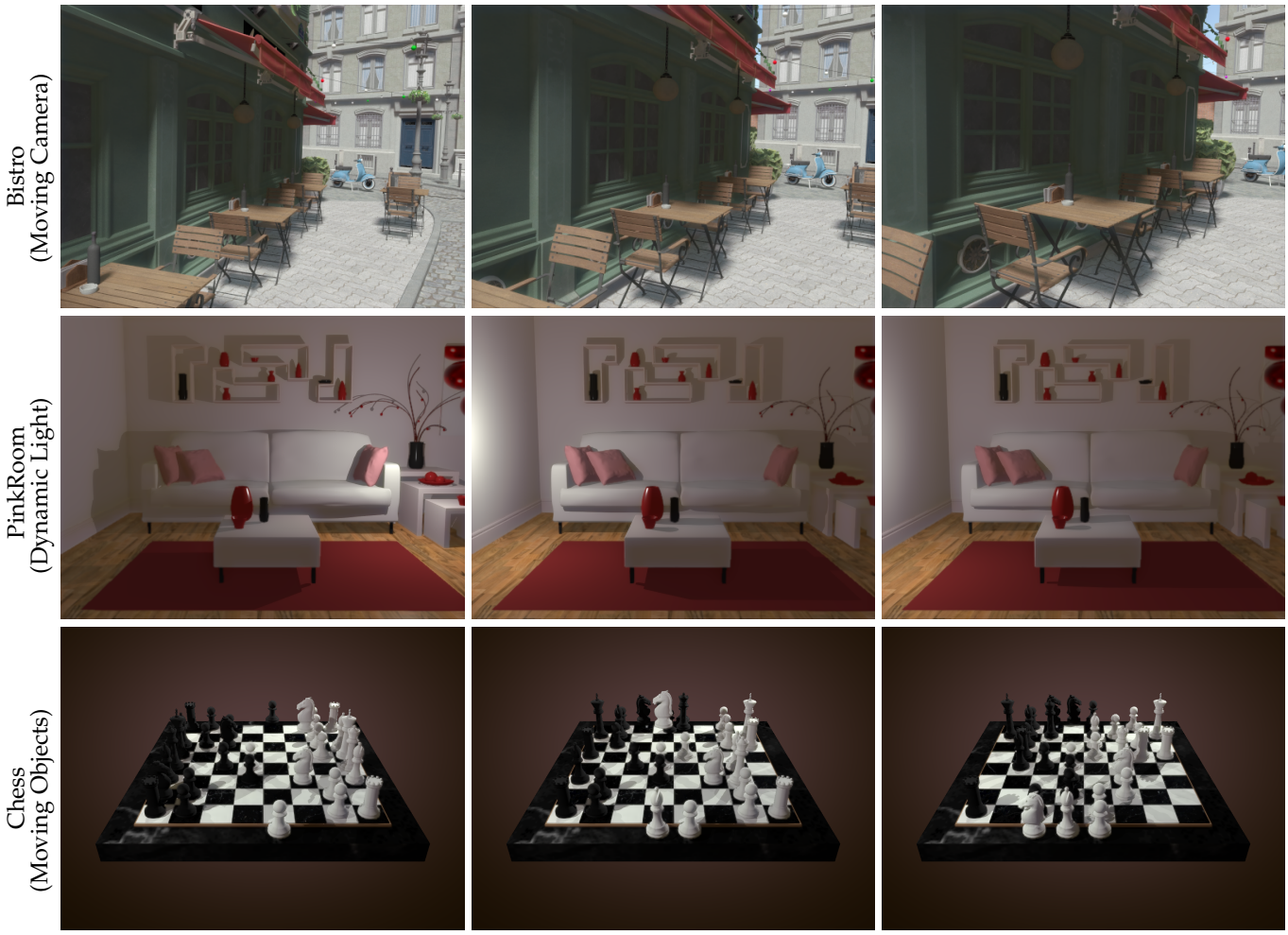


Fig. 9. Our method can produce high-quality global illumination for dynamic scenes under different settings. Top row: moving camera; middle row: dynamic light; bottom row: moving objects. For each scene, we pick up three frames from the rendered sequences.

8 Conclusion

We have proposed an interactive screen space method for single-bounce diffuse global illumination. The core of our method is a lightweight neural network. It takes full direct illumination and two auxiliary G-buffers of normals and 3D positions as input, and predicts single-bounce ndirect illumination. In order to explicitly take depth differences into account, we introduced bilateral convolutional layers in our networks. To maintain temporal coherence efficiently, we incorporated an optical flow based temporal loss during network training. We integrate our neural network into an OpenGL framework and demonstrate that we can produce high-quality global illumination at a resolution of 1024×768 for dynamic scenes in interactive framerates.

As future works, we would like to extend our method from screen space to world space, so that hidden surfaces could also contribute to indirect illumination computations. We are also interested in extending our method to deal with participating media. Furthermore, handling highly glossy interreflections is another worthwhile future direction to explore.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Project Number 61822204, 61521002), a research grant from the Beijing Higher Institution Engineering Research Center.

References

[1] O. Nalbach, E. Arabadzhiyska, D. Mehta, H.-P. Seidel, and T. Ritschel, "Deep shading: Convolutional neural networks for screen space shading," *Comput. Graph. Forum*, vol. 36, no. 4, pp. 65–78, 2017.

[2] T. Ritschel, T. Grosch, and H.-P. Seidel, "Approximating dynamic global illumination in image space," in *Proceedings of I3D*, 2009, pp. 75–82.

[3] J. F. Blinn and M. E. Newell, "Texture and reflection in computer generated images," *Commun. ACM*, vol. 19, no. 10, pp. 542–547, 1976.

[4] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive indirect illumination using voxel cone tracing," *Computer Graphics Forum*, vol. 30, no. 7, pp. 1921–1930, 2011.

[5] id Software, "Quake," https://github.com/id-Software/Quake-III-Arena, 1999.

[6] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 527–536, 2002.

[7] X. Sun, K. Zhou, Y. Chen, S. Lin, J. Shi, and B. Guo, "Interactive relighting with dynamic brdfs," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 27:1–27:10, 2007.

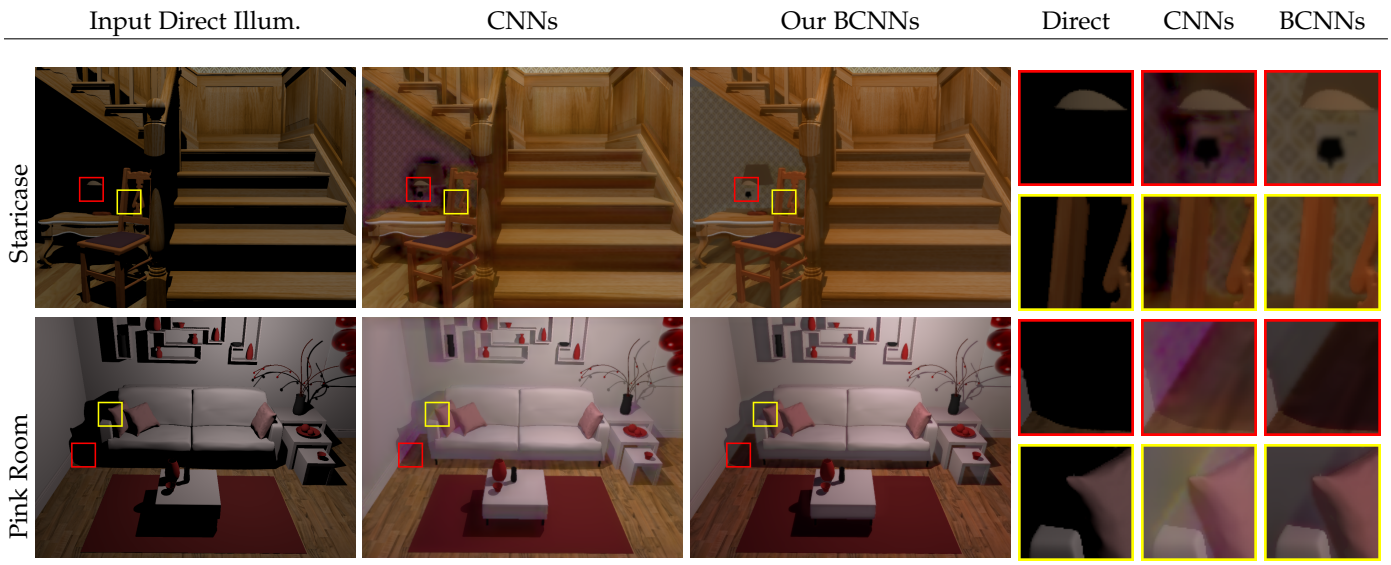


Fig. 10. Comparison the results of our BCNNs with traditional CNNs. Traditional CNNs easily produce noticeable artifacts, i.e., incorrect colors in shadow areas and blurring around object boundaries. By contrast, our BCNNs produce higher quality results without artifacts.

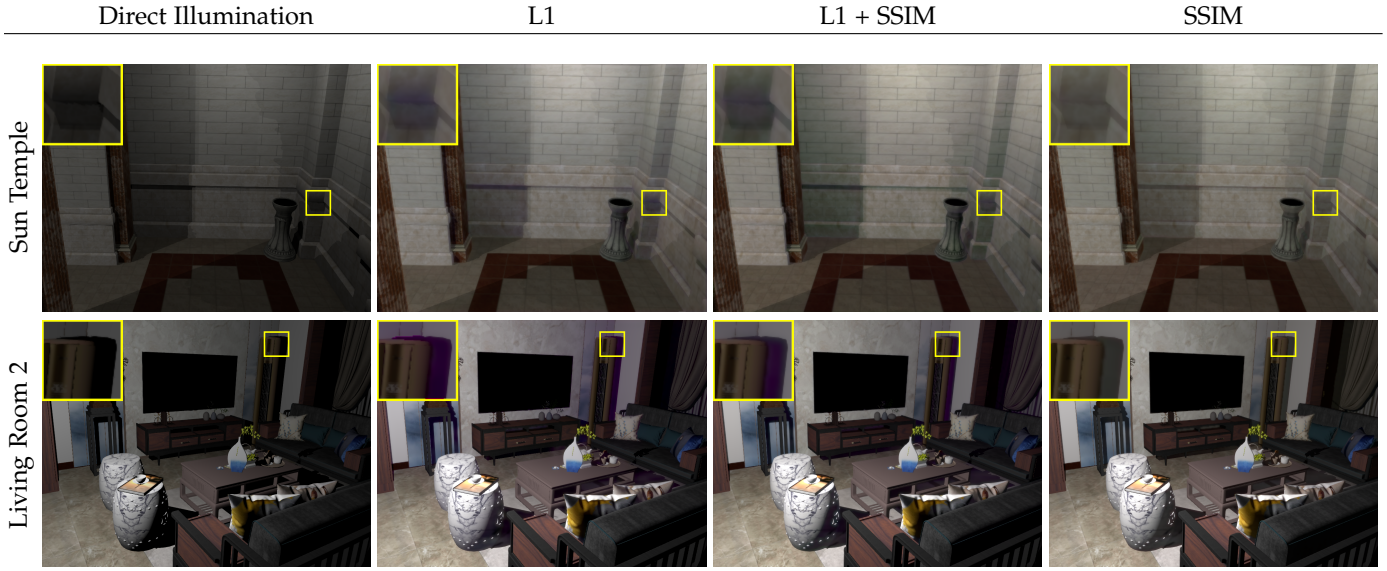


Fig. 11. Comparisons of different metrics in computing the content loss term (Equation 4). 'Using L1 only' or 'using L1+SSIM' are easy to introduce color-shifting artifacts in dark areas. 'Using SSIM only' generates results with the best visual quality.

[8] P. Ren, J. Wang, M. Gong, S. Lin, X. Tong, and B. Guo, "Global illumination with radiance regression functions," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 130:1–130:12, 2013.
[9] M. Hašan, F. Pellacini, and K. Bala, "Direct-to-indirect transfer for cinematic relighting," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1089–1097, 2006.
[10] A. Silvennoinen and J. Lehtinen, "Real-time global illumination by precomputed local reconstruction from sparse radiance probes," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 230:1 – 230:13, 2017.
[11] L.-Q. Yan, S. U. Mehta, R. Ramamoorthi, and F. Durand, "Fast 4d sheared filtering for interactive rendering of distribution effects," *ACM Trans. Graph.*, vol. 35, no. 1, pp. 7:1–7:13, 2015.
[12] C. Schied, A. Kaplanyan, C. Wyman, A. Patney, C. R. A. Chaitanya, J. Burgess, S. Liu, C. Dachsbacher, A. Lefohn, and M. Salvi, "Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination," in *Proceedings of High Performance Graphics*, 2017, pp. 2:1–2:12.
[13] S. Bako, T. Vogels, B. Mcwilliams, M. Meyer, and F. Rousselle, "Kernel-predicting convolutional networks for denoising monte carlo renderings," *Acm Trans. Graph.*, vol. 36, no. 4, pp. 97:1–97:14, 2017.
[14] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, and T. Aila, "Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder," *Acm Trans. Graph.*, vol. 36, no. 4, pp. 98:1–98:12, 2017.
[15] H. Huang, H. Wang, W. Luo, L. Ma, W. Jiang, X. Zhu, Z. Li, and W. Liu, "Real-time neural style transfer for videos," in *Proceedings of IEEE CVPR*, 2017, pp. 7044–7052.
[16] S. Kallweit, T. Müller, B. McWilliams, M. Gross, and J. Novák, "Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 23:1–23:11, 2017.
[17] W. Yang, J. Feng, J. Yang, F. Zhao, J. Liu, Z. Guo, and S. Yan, "Deep edge guided recurrent residual learning for image super-resolution," *IEEE Transactions on Image Processing*, vol. 26, no. 12, pp. 5895–5907, 2017.
[18] Nvidia, "Deep learning super sampling," https://developer.nvidia.com/rtx/ngx, 2018.
[19] C. Dachsbacher and M. Stamminger, "Reflective shadow maps," in *Proceedings of I3D*, 2005, pp. 203–231.
[20] P. Shanmugam and O. Arikan, "Hardware accelerated ambient occlusion techniques on gpus," in *Proceedings of I3D*, 2007, pp. 73–

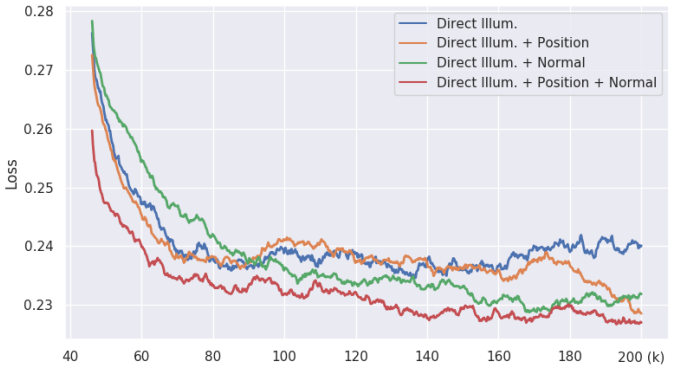


Fig. 12. Training loss concerning the number of training iterations when different combinations of auxiliary G-buffers are taken as input. Blue curve: without any auxiliary maps; orange curve: only with 3D positions; green curve: only with normals; red curve: with both normals and 3D positions. The training loss decreases fastest when both normals and 3D positions are taken as input.

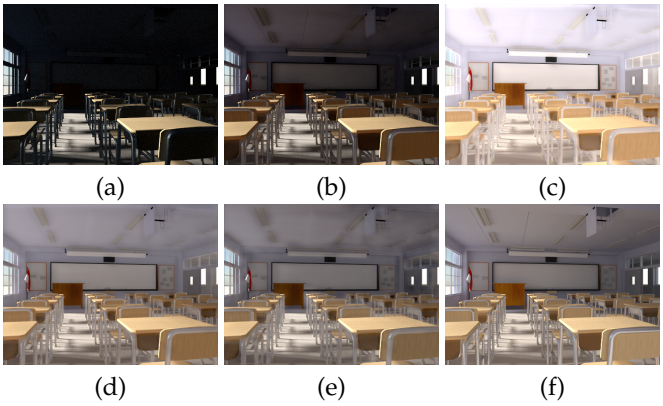


Fig. 13. Our method could be extended to handle multiple bounces. (a) direct illumination; (b) single-bounce indirect illumination; (c) 4-bounce indirect illumination without attenuation; (d) 4-bounce indirect illumination with attenuation; (e) re-trained with multi-bounce dataset; (f) multi-bounce reference.

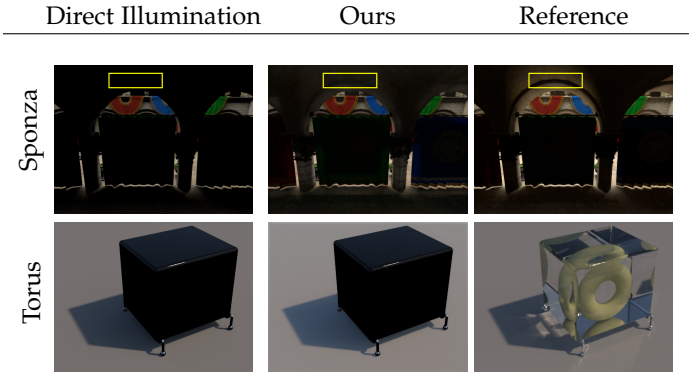
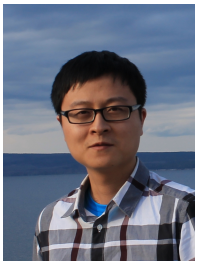


Fig. 14. Failure cases of our method.

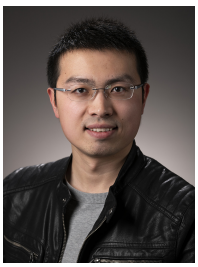
80.
[21] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of IEEE CVPR*, 2018, pp. 7794–7803.
[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[23] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.
[24] S. U. Mehta, B. Wang, R. Ramamoorthi, and F. Durand, "Axis-aligned filtering for interactive physically-based diffuse indirect lighting," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 96:1–96:12, 2013.
[25] B. Moon, J. A. Iglesias-Guitian, S.-E. Yoon, and K. Mitchell, "Adaptive rendering with linear predictions," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 121:1–121:11, 2015.
[26] J. Munkberg, J. Hasselgren, P. Clarberg, M. Andersson, and T. Akenine-Möller, "Texture space caching and reconstruction for ray tracing," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 249:1–249:13, 2016.
[27] Z. Majercik, J.-P. Guertin, D. Nowrouzezahrai, and M. McGuire, "Dynamic diffuse global illumination with ray-traced irradiance fields," *Journal of Computer Graphics Techniques (JCGT)*, vol. 8, no. 2, pp. 1–30, 2019.
[28] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *Proceedings of IEEE CVPR*, vol. 2. IEEE, 2005, pp. 60–65.
[29] B. Bitterli, F. Rousselle, B. Moon, J. A. Iglesias-Guitián, D. Adler, K. Mitchell, W. Jarosz, and J. Novák, "Nonlinearly weighted first-order regression for denoising monte carlo renderings," *Computer Graphics Forum*, vol. 35, no. 4, pp. 107–117, 2016.
[30] C. Schied, A. Kaplanyan, C. Wyman, A. Patney, C. R. A. Chaitanya, J. Burgess, S. Liu, C. Dachsbacher, A. Lefohn, and M. Salvi, "Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination," in *Proceedings of High Performance Graphics*, 2017, pp. 2:1–2:12.
[31] Y. Li, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep joint image filtering," in *Proceedings of ECCV*, 2016, pp. 154–169.
[32] M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand, "Deep bilateral learning for real-time image enhancement," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 118:1–118:12, 2017.
[33] R. Gadde, V. Jampani, M. Kiefel, D. Kappler, and P. V. Gehler, "Superpixel convolutional networks using bilateral inceptions," in *Proceedings of ECCV*, 2016, pp. 597–613.
[34] V. Jampani, M. Kiefel, and P. V. Gehler, "Learning sparse high dimensional filters: Image filtering, dense crfs and bilateral neural networks," in *Proceedings of IEEE CVPR*, 2016, pp. 4452–4461.
[35] J. Kontkanen, J. Räsänen, and A. Keller, "Irradiance filtering for monte carlo ray tracing," in *Monte Carlo and Quasi-Monte Carlo Methods*, 2004, pp. 259–272.
[36] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of ICCV*, 1998, pp. 839–846.
[37] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 664–672, 2004.
[38] A. Maas, A. Hannun, and A. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of ICML*, 2013, p. 3.
[39] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime tv-l1 optical flow," in *Joint Pattern Recognition Symposium*, 2007, pp. 214–223.
[40] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: A general purpose ray tracing engine," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 66:1–66:13, 2010.
[41] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, vol. 9, 2010, pp. 249–256.
[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.
[43] R. Fernando, "Percentage-closer soft shadows," in *ACM SIGGRAPH 2005 Sketches*, 2005, p. 35.
[44] P.-P. Sloan, N. K. Govindaraju, D. Nowrouzezahrai, and J. Snyder, "Image-based proxy accumulation for real-time soft global illumination," in *Proceedings of PG*, 2007, pp. 97–105.
[45] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 96:1–96:6, 2007.
[46] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of IEEE CVPR*, 2016, pp. 1646–1654.
[47] B. Karis, "High-quality temporal supersampling," in *Advances in Real-Time Rendering in Games, SIGGRAPH Courses*, 2014.
[48] B. Bitterli, "Rendering resources," 2016, https://benedikt-bitterli.me/resources/.
[49] M. McGuire. (2017, July) Computer graphics archive. [Online]. Available: https://casual-effects.com/data

Hanggao Xin is a PhD student in the Department of Computer Science and Technology, Tsinghua University. He received his bachelor degree from the same university in 2018. His research interests include realistic rendering and real-time ray tracing.

Shaokun Zheng is a senior undergraduate student in the Department of Computer Science and Technology, Tsinghua University. His research interests include realistic rendering and real-time ray tracing.



Kun Xu is an associate professor in the Department of Computer Science and Technology, Tsinghua University. Before that, he received his bachelor and doctor degrees from the same university in 2005 and 2009, respectively. His research interests include realistic rendering and image/video editing.



Ling-Qi Yan is an assistant professor of Computer Science at UC Santa Barbara, co-director of the MIRAGE Lab, and affiliated faculty in the Four Eyes Lab. Before that, he received his doctor degree from the Department of Electrical Engineering and Computer Sciences at UC Berkeley and obtained his bachelor degree in Computer Science from Tsinghua University. His research interests include physically-based rendering, real-time ray tracing and realistic appearance modeling.