

# A detail preserving neural network model for Monte Carlo denoising

Weiheng Lin<sup>1</sup>, Beibei Wang<sup>1</sup> (✉), Lu Wang<sup>2</sup> (✉), and Nicolas Holzschuch<sup>3</sup>

© The Author(s) 2020.

**Abstract** Monte Carlo based methods such as path tracing are widely used in movie production. To achieve low noise, they require many samples per pixel, resulting in long rendering time. To reduce the cost, one solution is Monte Carlo denoising, which renders the image with fewer samples per pixel (as little as 128) and then denoises the resulting image. Many Monte Carlo denoising methods rely on deep learning: they use convolutional neural networks to learn the relationship between noisy images and reference images, using auxiliary features such as position and normal together with image color as inputs. The network predicts kernels which are then applied to the noisy input. These methods show powerful denoising ability, but tend to lose geometric or lighting details and to blur sharp features during denoising.

In this paper, we solve this issue by proposing a novel network structure, a new input feature—light transport covariance from path space—and an improved loss function. Our network separates feature buffers from the color buffer to enhance detail effects. The features are extracted separately and then integrated into a shallow kernel predictor. Our loss function considers perceptual loss, which also improves detail preservation. In addition, we use a light transport covariance feature in path space as one of the features, which helps to preserve illumination details. Our method denoises Monte Carlo path traced images while preserving details much better than previous methods.

**Keywords** deep learning; light transport covariance; perceptual loss; Monte Carlo denoising

## 1 Introduction

Monte Carlo based methods are widely used for rendering in movie production [1], as they are physically based and can produce unbiased results. However, they require many samples per pixel to produce noise-free results. To reduce rendering costs, one solution is generate a noisy image with fewer samples and use denoising methods to remove the noise. This is called *Monte Carlo rendering denoising*.

Several Monte Carlo rendering denoising methods use deep learning. Bako et al. [2] used a convolutional neural network (CNN) to predict final denoised pixel values as a highly non-linear combination of input features. More precisely, they decouple diffuse and specular lighting in the rendered image and use two networks for learning. Instead of learning the denoised pixel value directly, they learn a kernel for each pixel and apply the kernel to neighbors of each pixel to reconstruct the denoised color. Vogels et al. [3] further improved on this work, using residual blocks to accelerate the convergence of the network. They consider the rendering sources of the images, e.g., different renderers, different filtering methods to avoid limitations of inputs. They also solve the temporal coherency issue between different images.

These methods very efficiently denoise Monte Carlo rendered images, but they tend to remove details (see Fig. 1), decreasing the quality of the resulting image. Details can come from geometry (see Fig. 1) or from lighting effects (see Fig. 9). Existing denoising algorithms capture details by extracting features from the color buffer and auxiliary buffers containing, e.g., positions and normals. However, details may only

1 School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China. E-mail: W. Lin, 442920398@qq.com; B. Wang, beibei.wang@njust.edu.cn (✉).

2 School of Computer Science and Technology, Shandong University, Jinan 250100, China. E-mail: luwang-hcivr@sdu.edu.cn (✉).

3 Univ. Grenoble-Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France. E-mail: Nicolas.Holzschuch@inria.fr.

Manuscript received: 2020-02-06; accepted: 2020-02-22



**Fig. 1** Comparison of output of our network and the Kernel Predicting Convolutional Network (KPCN) using the same dataset for training. Our model preserves details better, due to the novel network structure, a new feature (light transport covariance in path space), and the perceptual loss function. Quantitative error metrics (RelMSE and DSSIM) also confirm the higher quality of our method.

be obvious in a subset of the features; for example, complex lighting might show obvious differences in the color buffer, but have no discontinuities in the position or normal buffer, while complex geometry would show the opposite tendency. Training on all features together results in excessive blurring.

In this paper, we solve this issue by separating auxiliary feature buffers and the color buffer to enhance details. We extract their features separately, and then integrate them into a shallow kernel predictor. Our loss function considers perceptual loss, which also improves detail preservation. In addition, we introduce a light transport covariance feature in path space as one of the features. The covariance matrix represents frequency of light transport in the path space, which captures complex lighting details. Overall, our model preserves geometric and lighting details much better than previous methods.

In the next section, we review some previous work on Monte Carlo denoising and deep neural networks. Then, we review KPCN [2] and covariance tracing [4] in Section 3. Section 4 presents our method, and we explain implementation details in Section 5. We present our results, compare them with those from other methods, and analyze performance in Section 6. We conclude in Section 7.

## 2 Previous work

### 2.1 Machine learning based Monte Carlo denoising

Kalantari et al. [5] introduced neural networks for Monte Carlo denoising. Their algorithm learns the relationship between noisy images and ideal filter parameters with a multilayer perceptual neural

network, and then uses the learned model for new scenes for a wide range of distributed effects. Bako et al. [2] introduced a convolutional neural network (CNN) model to predict local weighting kernels to filter pixels from their neighbors. Their method is called *KPCN*. They decompose input into diffuse and specular components for which they train separate CNN models. The KPCN method is more efficient than earlier Monte Carlo denoisers. Vogels et al. [3] further improved denoising by combining KPCN with a number of task-specific modules, e.g., a source-aware encoder, and optimizing the assembly using an asymmetric loss, resulting in a more robust solution.

Chaitanya et al. [6] proposed a recurrent neural network (RNN) model for interactive rendering which considers temporal coherency.

Gharbi et al. [7] applied learning directly between samples and kernel parameters, instead of starting with noisy images. Since samples include more information, it produces higher quality even with only a few samples.

Yang et al. [8] proposed a dual-encoder network. The method first fuses feature buffers using a sub-network, then separately encodes the fused feature buffers and color buffer, and finally reconstructs a clean image using a decoder network. In comparison, our method does not first fuse auxiliary feature buffers, and adds a light transport covariance buffer which represents the frequency of the light transport. We use a residual network to filter the color buffer and auxiliary feature buffers separately, and then integrate their feature maps for a shallow kernel predictor network. Hence our algorithm is based on kernel prediction rather than being an end-to-end method.

## 2.2 Image space Monte Carlo denoising

Another avenue of work denoises Monte Carlo rendered images only in image space. It achieves high-quality results at reduced sampling rate [9].

Zero-order linear regression model based methods [10–13] use a non-local means filter in a joint filtering scheme, and combine color and auxiliary feature buffers robustly for denoising. These methods have well-chosen weighting kernels and can yield good performance, but are limited by their explicit filters, which make their filter kernel less flexible.

First-order [14, 15] or higher-order models [16] for Monte Carlo denoising are less constrained. They directly exploit the correlation between the auxiliary buffer and the color buffer, allowing for better use of neighborhood data. First order methods have problems dealing with low frequency noise, but higher-order methods can suffer from over-fitting.

Boughida et al. [17] proposed a non-local Bayesian collaborative filter, which globally produces high quality denoising, especially in dark areas.

## 3 Background

### 3.1 Problem statement

The problem of denoising Monte Carlo rendering can be formulated as

$$\hat{c} = \Phi(x; \theta) \quad (1)$$

where  $\hat{c}$  is the denoised result,  $\Phi$  is a filter for denoising,  $x$  is the noisy input data, and  $\theta$  are parameters controlling  $\Phi$ .  $x = [c, f]$  consists of the average RGB color  $c$  and optional auxiliary feature buffers  $f$  obtained from a renderer.

Following the previous deep learning based Monte Carlo denoising method, we chose a convolutional neural network as the filter  $\Phi$ . We formalize it as a supervised learning problem that uses a data set containing  $N$  example pairs of noisy inputs  $\{x^1, \dots, x^N\}$  and corresponding ground truth  $\{r^1, \dots, r^N\}$  to optimize the parameters of the network:

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} \sum_{n=1}^N l(r^n, \Phi(x^n; \theta)) \quad (2)$$

where  $l$  is an optional loss function giving the difference between the filtered color and ground truth. After training the network, the denoised result  $\hat{c}$  should be noise-free and preserve scene details.

### 3.2 Kernel prediction convolutional network

#### 3.2.1 Origins

Bako et al. proposed the first CNN based Monte Carlo denoising method. They decouple the rendered output into diffuse and specular components. The two components are preprocessed, and used to train individual CNN networks which output separate kernels used to obtain denoised diffuse and specular components. An inverse preprocessing transform then combines them to produce the final denoised result. Details can be found in Ref. [2].

#### 3.2.2 Input features

The renderer decomposes rendered outputs into diffuse and specular components. The rendered outputs include color buffers consisting of diffuse color (3 channels), specular color (3 channels), and their color variances, and auxiliary feature buffers consisting of normals (3 channels), depth (1 channel), albedo (3 channels), and their feature variances. Variances are converted to a single channel using luminance.

#### 3.2.3 Network architecture

KPCN uses a plain 9-layer CNN. In the first eight layers, the network applies a linear convolution to the previous layer's output, adds a constant bias, and then applies a Relu activation function. In the last layer, it outputs a  $K \times K$  kernel of scalar weights instead of directly outputting a denoised pixel.

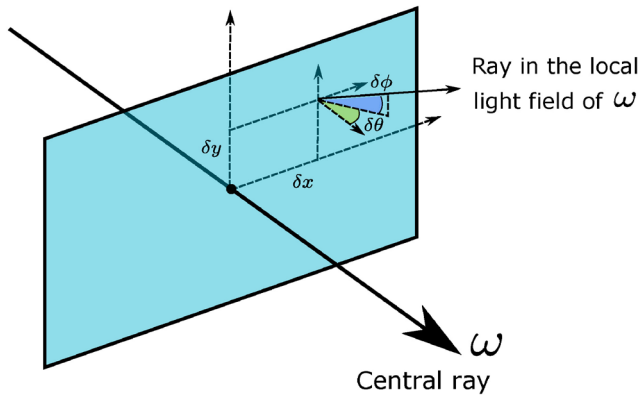
#### 3.2.4 Loss function

The loss function should represent the perceptual difference between the estimated and reference color well and be easy to optimize. KPCN uses  $L_1$  loss to optimize its network. The authors experimented with several loss functions, including  $L_1$ , relative (rel)  $L_1$ ,  $L_2$ , rel  $L_2$ , and SSIM (Structural Similarity). Their experimental results show that optimization of the  $L_1$  loss function is best:

$$l_1 = |c_{\text{denoised}} - c_{\text{reference}}| \quad (3)$$

### 3.3 Light transport covariance in path space

Durand et al. [19] introduced a framework for frequency analysis of light transport. They compute the frequency content of the local light field around a given ray. The local light field is defined as a 4D function, with two spatial dimensions and two angular dimensions (see Fig. 2). Standard operations on light transport, such as transport in free space or reflection,



**Fig. 2** The local light field is defined as a 4D function around the central ray ( $\omega$ ), parameterized by two spatial coordinates ( $\delta_x$  and  $\delta_y$ ) and two angular coordinates ( $\delta_\theta$  and  $\delta_\phi$ ) [18].

transform into operations on the Fourier spectrum of the local light field. Running computations with the full Fourier spectrum of the local light field is impractical. Belcour et al. [20] introduced an approximate representation for the Fourier spectrum of the local light field: the covariance matrix.

The key idea of Belcour et al. is to compute the covariance matrix of the Fourier spectrum of the local light field using matrix operations corresponding to basic operations of light transport (transport in free space, reflection, occlusion). See Ref. [4] for the detailed computation of these operations.

## 4 Method

### 4.1 Network architecture

Our network consists of four parts (see Fig. 3(a)): data preprocessing (see Section 5.1), feature extraction, shallow kernel prediction, and reconstruction.

In preprocessing, we separate features into diffuse and specular components, as in Bako et al. [2]: factoring out albedo from the diffuse component, applying a logarithmic transform to the specular component, scaling depth to the range  $[0, 1]$ , and taking gradients for all buffers including diffuse, specular, normal, albedo, and depth, with the addition of a light transport covariance feature (see Section 4.2).

Inspired by Simonyan and Andrew [21], in feature extraction, we first separate diffuse and specular components into a color component and a feature component respectively, to enhance detail capture. Each component is then sent to a feature extractor, which is a residual network (Fig. 3(b)). Our residual

network consists of eight residual blocks and two convolutional layers at the beginning and the end. As in Vogels et al. [3], the residual block has a two-layer network structure, with each layer containing a Relu activation function and a convolution layer. At the end of the residual block, the output of the convolutional layer and the input of the residual block are summed. Then the filtered color component and feature component are concatenated and fed into the next part of the framework. We use a residual network rather than a CNN, because a convolutional network with too many hidden layers may result in vanishing and exploding gradient, while a residual network protects data integrity by directly passing input data to the output (via skip connections) and the network only needs to learn the difference between inputs and outputs to simplify learning objectives.

The third part of our framework is a shallow kernel prediction network (Fig. 3(c)), which consists of only four traditional convolutional layers. Two kernel predictors output two  $21 \times 21$  kernels to denoise diffuse and specular buffers separately. We use a shallow network rather than a deep network, as a deep network makes optimization of the feature extractor more difficult, leading to degradation of the quality after training.

Finally, the inverse of the preprocessing transform is applied to the denoised data (i.e., multiplying irradiance by the albedo and applying an exponential transform to the specular component), and then the denoised diffuse and specular images are combined to obtain the overall denoised image.

### 4.2 Light transport covariance feature

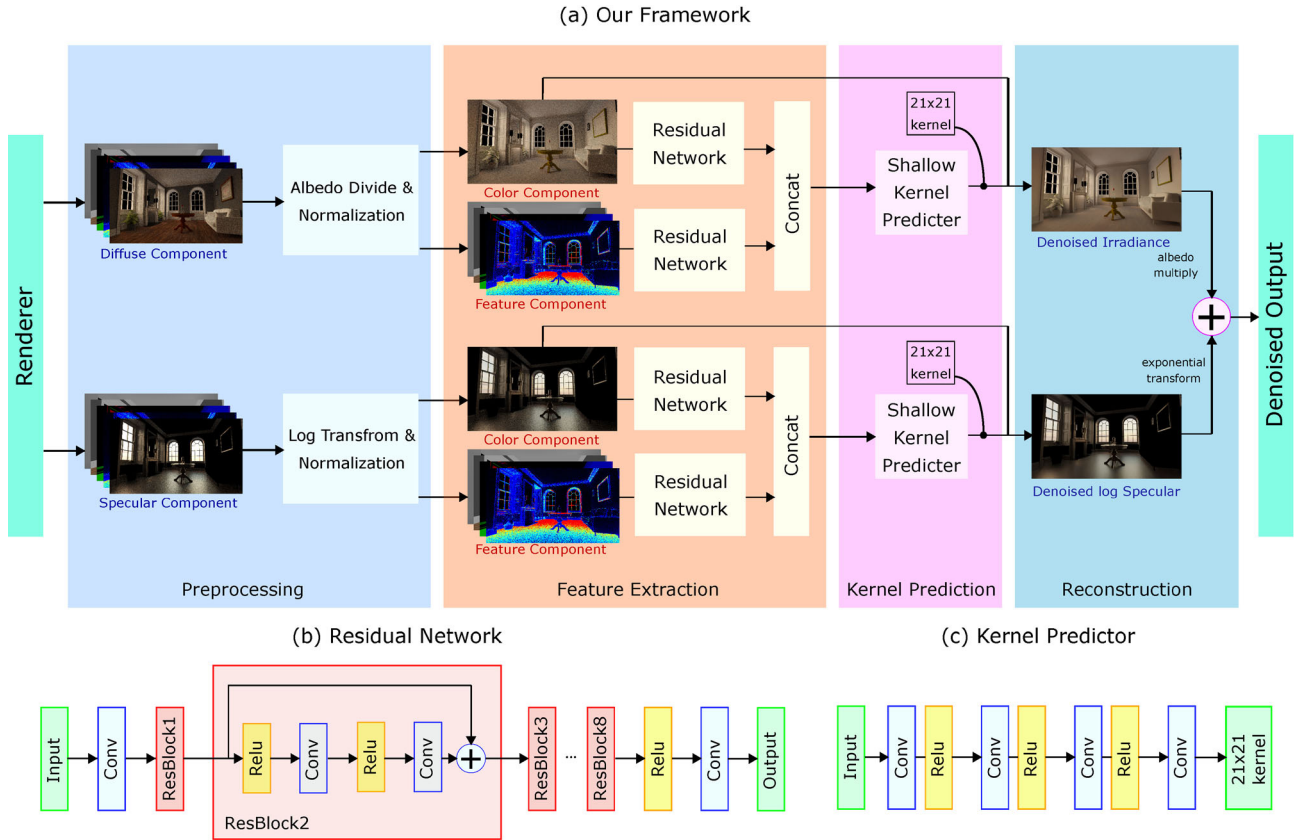
We introduce light transport covariance defined by Belcour et al. [4] as one of the input features, as it can represent the frequency of the light transport to help detail preservation.

The covariance matrix is denoted  $\Sigma$ . For a function  $f$  defined over a 4D domain, this  $4 \times 4$  matrix is defined by

$$\Sigma_{i,j} = \frac{\int_{x \in \Omega} (x \cdot e_i)(x \cdot e_j) f(x) dx}{\int_{\Omega} f} \quad (4)$$

where  $e_i$  is the  $i$ th basis vector of the 4D space  $\Omega$  and  $x \cdot y$  denotes the dot product of vectors  $x$  and  $y$ .

The eigenvectors of the covariance matrix indicate in which directions function  $f$  spreads most and least; its eigenvalues are the variance of the function in all 4 principal directions.

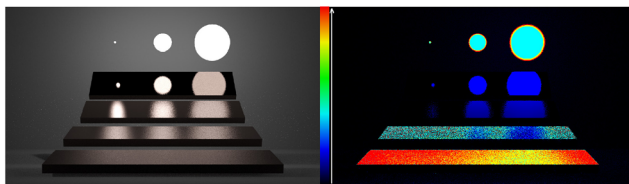


**Fig. 3** (a) Framework. The renderer decomposes rendered outputs into diffuse and specular components which are preprocessed independently. In both, their features are separated into a color component and a feature component. These are fed into a residual network receptively to extract features and the extracted features are concatenated. Next, two kernel predictor networks filter the extracted features and output two  $21 \times 21$  kernels, which are used to denoise preprocessed diffuse and specular buffers. Finally, these denoised results are combined to obtain the full denoised image. (b) The residual network architecture, with eight residual blocks. (c) The kernel predictor architecture, with four convolutional layers.

We next compute the determinant  $\eta$  of the covariance matrix, by

$$\eta = \sqrt{|\Sigma|} \tag{5}$$

$\eta$  lies in  $[0, 1]$ . The higher the value of  $\eta$ , the larger the frequency content at this location.  $\eta = 0$  corresponds to a uniform, constant distribution (low frequency), and  $\eta = 1$  corresponds to a Dirac delta function (high frequency). We use the determinant of the covariance matrix as a feature for training. This feature benefits complex lighting detail preservation (see Fig. 9). Figure 4 visualizes this feature.



**Fig. 4** A light transport covariance buffer. Left: the full color buffer. Right: the corresponding light transport covariance buffer.

### 4.3 Loss function

Our loss function is defined as

$$l = l_s + l_p \tag{6}$$

where  $l_s$  is the symmetric mean absolute percentage error (SMAPE), which has good stability in HDR images:

$$l_s = \frac{|c_{\text{denoised}} - c_{\text{reference}}|}{(|c_{\text{denoised}}| + |c_{\text{reference}}| + \varepsilon)/2} \tag{7}$$

where  $\varepsilon$  is a small number, taken to be  $10^{-8}$  in our implementation.

We also include the perceptual loss  $l_p$ :

$$l_p = \frac{1}{whd} \|\phi(c_{\text{denoised}}) - \phi(c_{\text{reference}})\|_2 \tag{8}$$

where  $\phi$  is a feature extractor, and  $w$ ,  $h$ , and  $d$  represent the width, height, and depth of the denoised image respectively. Following Ref. [22], we use pre-trained VGG-19 [23] as the feature extractor  $\phi$ , as VGG-19 can provide high-dimensional feature information for an image. Using the perceptual loss

helps to preserve more details in the denoised image (see Fig. 11).

## 5 Data creation and training

### 5.1 Data creation

For training, we rendered images and buffers with the Tungsten renderer [24] to give our dataset. Training a neural network requires a large and representative dataset to avoid overfitting. Thus, in order to generate sufficient data, we modified publicly available scenes [25] (see Fig. 5) by varying camera parameters, materials, and light sources. The noisy images were rendered with 32 samples per pixel (spp) or 128 spp, and the reference images were rendered with 8192 spp. The resolution of these images was  $1280 \times 720$ . We rendered about 220 scenes as our training set and about 20 scenes as our validation set.

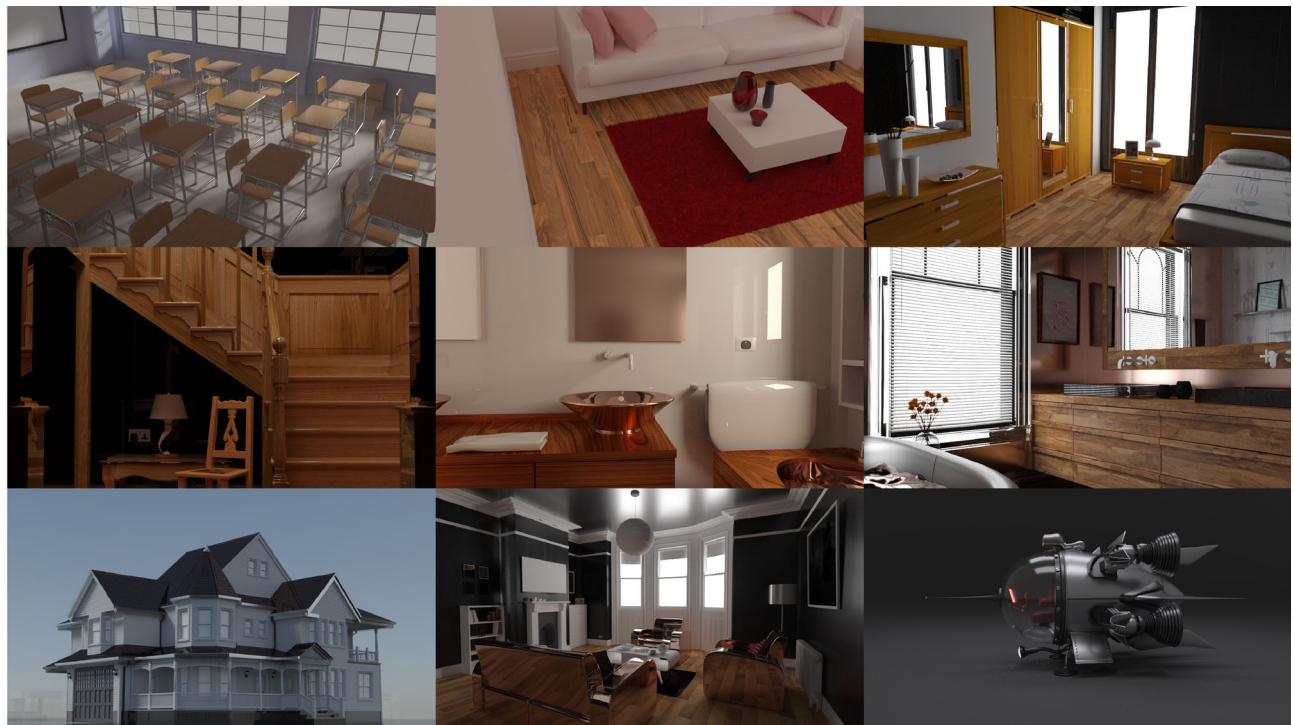
Following Bako et al. [2], we decompose rendered outputs into diffuse and specular buffers. In addition to the feature buffer used in KPCN, we add a light transport covariance feature buffer (see Fig. 4) (1 channel). The renderer outputs 20 channels in total (diffuse, specular, albedo, normal, depth, light transport covariance, and their corresponding

variances). We factor out the albedo from the diffuse channel and apply a logarithmic transform to the specular channel. We take gradients in both  $x$  and  $y$  directions for all buffers, and linearly scale the depth and light transport covariance buffer to the range  $[0, 1]$  for each frame.

### 5.2 Implementation and training

We implemented our network in TensorFlow [26] and used the ADAM [27] optimizer to optimize the parameters. Weights were initialized using the Xavier method [28].

To perform training, we split the processed data into  $128 \times 128$  patches, and then shuffled them and fed them into the network. The corresponding networks for diffuse and specular denoising pipelines were trained independently. The loss for the diffuse denoising pipeline network is computed between the denoised irradiance and the reference irradiance, while the loss for the specular denoising network is computed in the log domain. For each 500 iterations, we use 10 patches to train the network with learning rate,  $\eta = 10^{-4}$ . The process of selecting patches follows Bako et al. [2]. Each network is trained for approximately 50k iterations during 1.5 days on a Tesla K80 GPU.



**Fig. 5** Example images from our dataset. We modify camera, materials, and light sources of some publicly available scenes to enrich our dataset.

## 6 Results

We compare our results to those produced by four state-of-the-art methods: NFOR [15], KPCN [2], BCD [17], DEMC [8], and to reference images. We use DSSIM (structural dissimilarity) and RelMSE (relative mean squared error) as metrics to evaluate quality of the results. The input images were rendered with 32–128 spp, and the references were rendered with 8912–20000 spp.

### 6.1 Model validation



















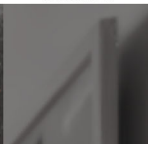
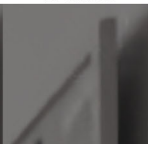
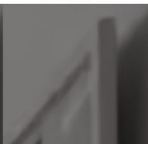
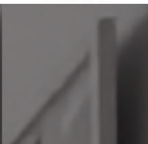
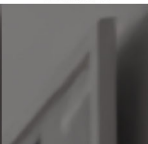
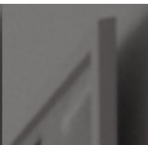

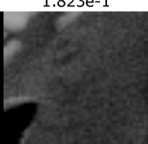

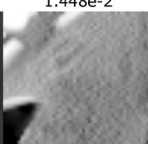





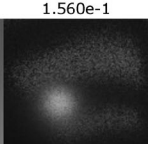
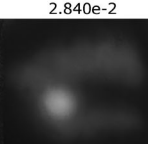
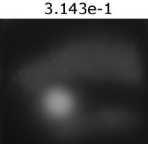
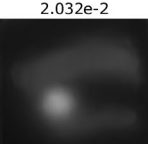

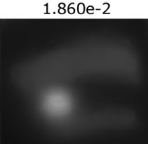

Figure 6 compares results from our model with the other four methods and reference images. The error metrics show that our model produces higher quality results and preserves details better. NFOR blurs the details of textures and lighting, and produces artifacts with low frequency noise. BCD still has

some noise in many geometric details. Compared to NFOR and BCD, KPCN has better overall denoising effects, but it has blurring and aliasing in some tiny details. DEMC is better than KPCN in preserving geometric details on some scenes, but it is not as good as our method in processing high-frequency lighting details.

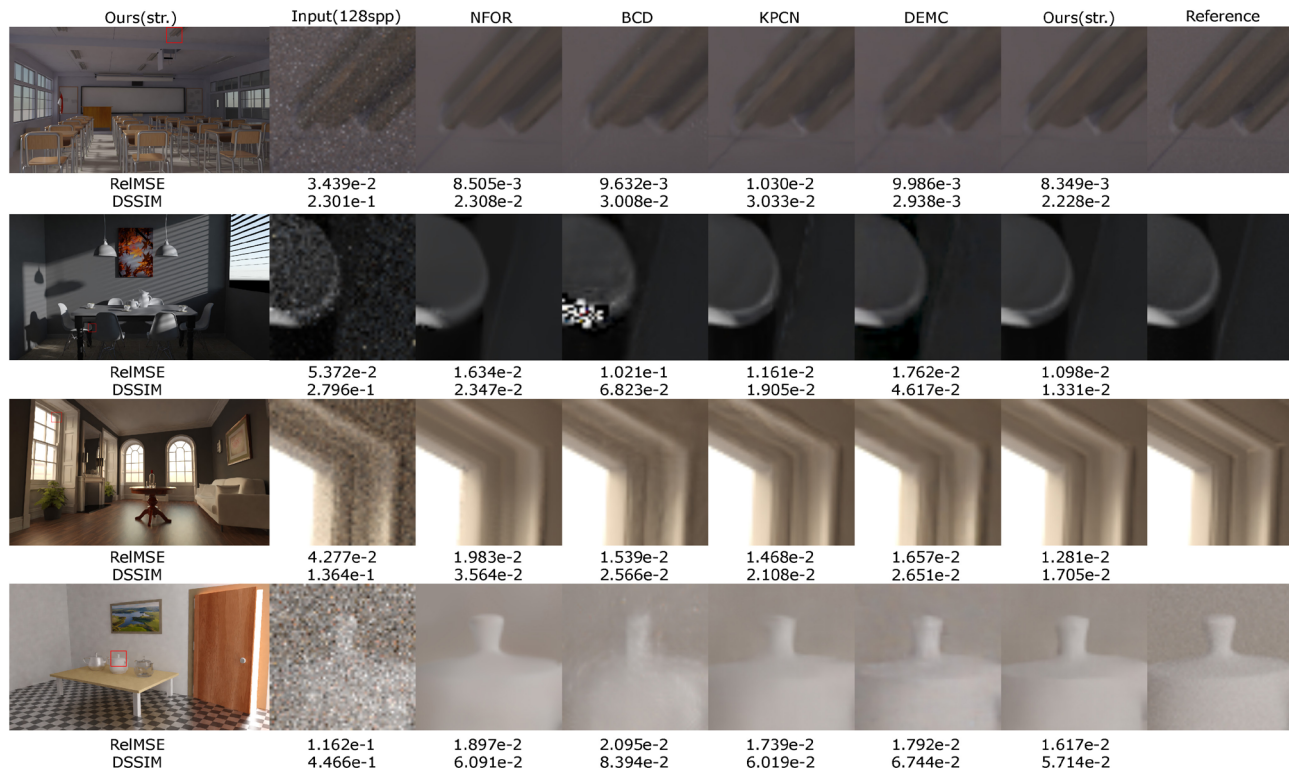
In Fig. 8, we show the error as a function of number of iterations for KPCN and our model. From 1k iterations, we perform validation every 2k iterations and calculate RelMSE. Our method consistently has smaller error than KPCN.

### 6.2 Model structure validation

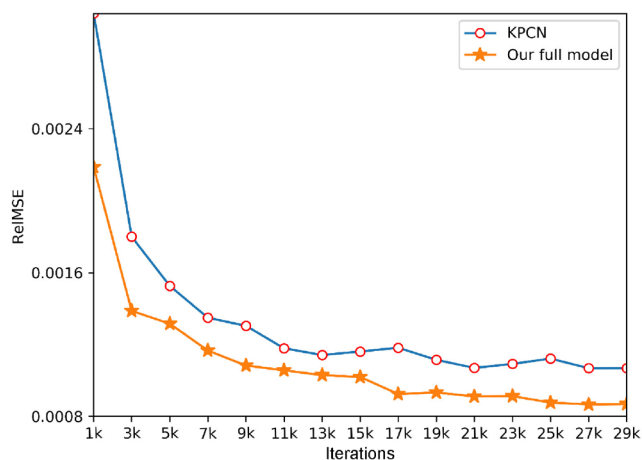
In Fig. 7, we focus on network structure, and disable light transport covariance and perceptual loss for network training in our model. We compare our model without these features to the state-of-the-art

Ours	Input(128spp)	NFOR	BCD	KPCN	DEMC	Ours	Reference
							
RelMSE	2.572e-2	9.666e-3	9.717e-3	1.003e-2	8.946e-3	7.467e-3	
DSSIM	1.486e-1	1.582e-2	2.024e-2	2.222e-2	1.798e-2	1.341e-2	
							
RelMSE	2.467e-2	1.138e-2	9.630e-3	1.039e-2	1.189e-2	8.608e-3	
DSSIM	1.395e-1	3.198e-2	2.487e-2	2.788e-2	3.332e-2	1.860e-2	
							
RelMSE	3.190e-2	9.762e-3	9.386e-3	8.649e-3	8.569e-3	7.146e-3	
DSSIM	1.823e-1	1.436e-2	1.448e-2	1.362e-2	1.343e-2	1.055e-2	
							
RelMSE	2.869e-2	1.309e-2	3.429e-1	1.222e-2	1.326e-2	1.112e-2	
DSSIM	1.560e-1	2.840e-2	3.143e-1	2.032e-2	2.475e-2	1.860e-2	
							
RelMSE	3.151e-2	7.452e-3	8.628e-3	5.057e-3	1.082e-2	4.988e-3	
DSSIM	1.553e-1	7.661e-3	6.730e-3	4.462e-3	9.817e-3	3.665e-3	

**Fig. 6** Comparison of our method to four other state-of-the-art methods NFOR [15], KPCN [2], BCD [17], DEMC [8], and reference images. KPCN's input features and loss function are as in the original paper (Section 3.2). Our model includes light transport covariance, besides the features of KPCN, and is trained with the loss function in Section 4.3. KPCN, DEMC, and our model have identical other training settings (see Section 5.2) and used the same dataset for training.



**Fig. 7** Network structure comparison between our model (str. means network structure only) and previous works. To validate the effect of network structure, our network training does not use light transport covariance and perceptual loss. KPCN, DEMC, and our method have the same training settings (see Section 5.2) other than the network structure. Even without light transport covariance and perceptual loss, our method provides a better result.



**Fig. 8** ReIMSE as a function of training iterations for our method and KPCN. Our method is consistently better than KPCN.

methods. The error metrics show that our model produces higher quality results which preserve details better, while KPCN has aliasing or blurring in some details.

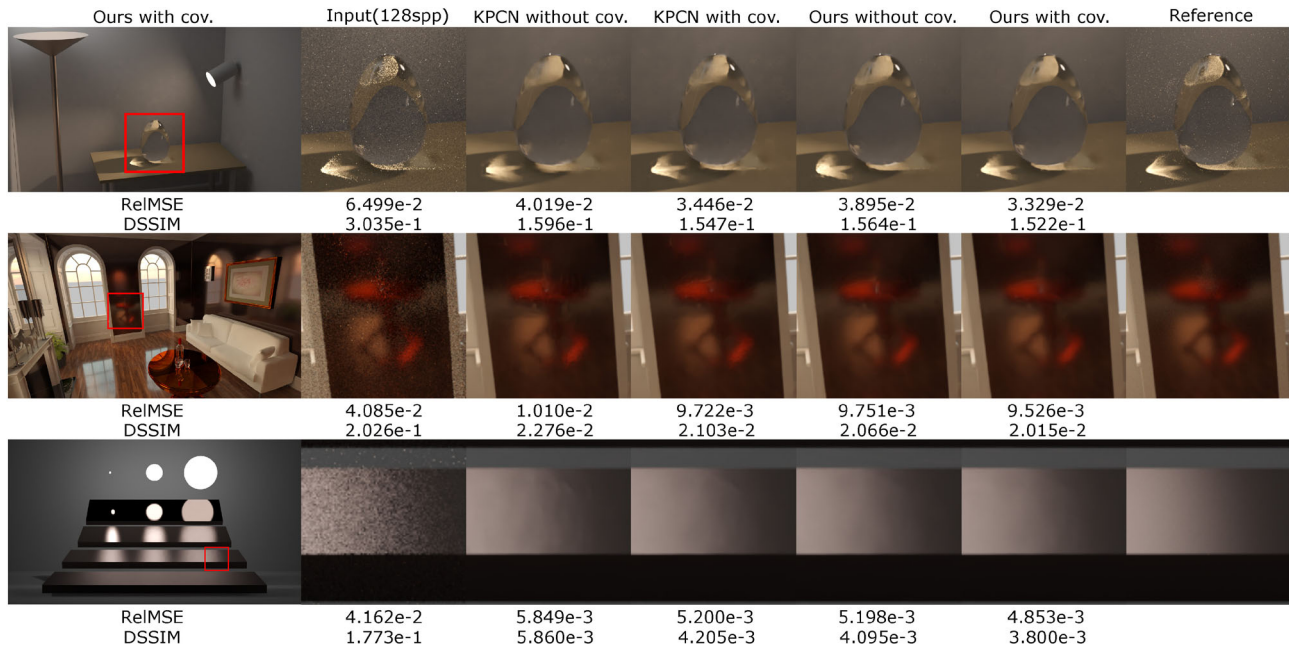
### 6.3 Light transport covariance buffer validation

We validate the impact of the light transport covariance buffer for scenes with complex lighting.

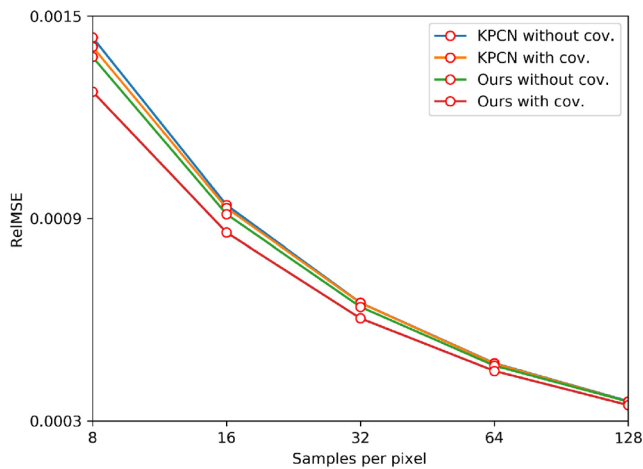
In Fig. 9, we show the impact of adding the light transport covariance buffer to the training, for both KPCN and our model. In both cases, adding light transport covariance significantly improves the handling of high frequency details. Light transport covariance can represent the frequency of the light transport so that neural networks can learn more features of high frequency light details. As shown in Fig. 9, the caustics, glossy, and specular details are preserved better when using the light transport covariance buffer.

In Fig. 10, we show the impact of the number of samples per pixel (spp) on denoising quality when using the light transport covariance buffer. Our networks are trained with only SMAPE loss, to validate the effects of light transport covariance. The test data consists of several scenes rendered at different sample counts (8, 16, 32, 64, and 128 spp), and filtered with four models (our model training with/without light transport covariance, KPCN training with/without light transport covariance). The error between the filtered results and the references is calculated and averaged. Our method





**Fig. 9** Comparison of training with and without light transport covariance. We use the feature buffer with and without light transport covariance to train KPCN and our method. Specially selected scenes show the advantages of training with the light transport covariance (cov. means light transport covariance).



**Fig. 10** RelMSE comparison between our method (with light transport covariance), our method (without light transport covariance), KPCN (without light transport covariance), and KPCN (with light transport covariance) for varying sample counts.

with covariance produces the best result for any level of input noise. In addition, light transport covariance can help improve the denoising quality for both our method and KPCN, especially for low numbers of samples per pixel.

### 6.4 Loss function validation

To validate the effects of perceptual loss used in our loss function, we compare our method with and without perceptual loss in Fig. 11. With perceptual

**Table 1** Cost of using light transport covariance. We implemented light transport covariance in the Tungsten renderer and experimented with four scenes, rendered at 128 spp and  $512 \times 512$  resolution

Scene	Time without cov.	Time with cov.	Cost
Bathroom	3 m 07 s	3 m 32 s	+13.37%
Classroom	2 m 53 s	3 m 15 s	+12.72%
Kitchen	3 m 30 s	3 m 50 s	+9.52%
Living-room	2 m 56 s	3 m 17 s	+11.93%

loss, geometric details are further restored, giving results closer to the reference than denoised results when only trained with SMAPE. Training with perceptual loss helps the denoising result to be more similar to the reference for high-level features, making geometric details sharper.

### 6.5 Shallow kernel predictor validation

We used a shallow network (4 layers) for our kernel predictor. We compare this shallow network with a deep network (10 layers) in Fig. 12. The shallow network works better than a deep network. The latter makes optimization of the feature extractor more difficult, leading to degradation of the training quality. Therefore, we use a shallow network for kernel prediction, for better performance and to reducing the number of network parameters.

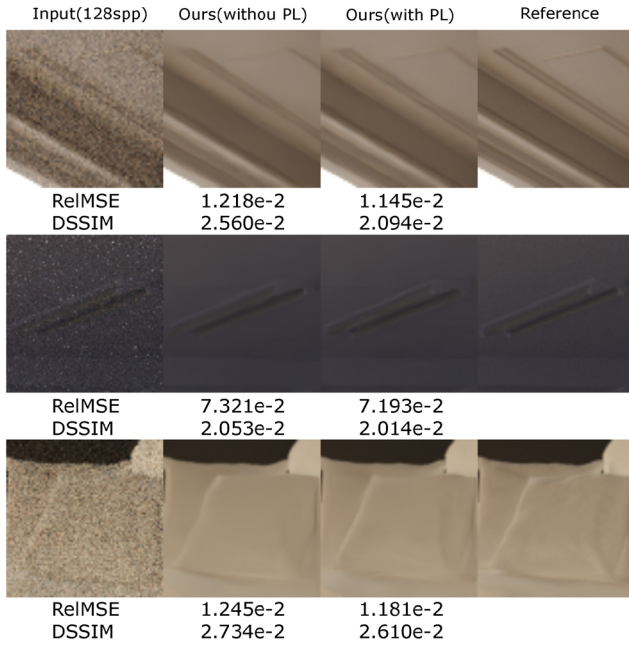


Fig. 11 Our method with and without perceptual loss (PL).

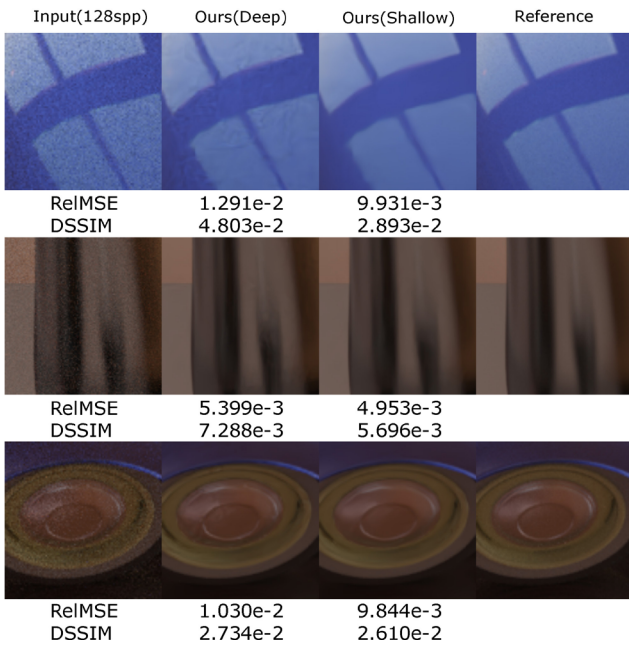


Fig. 12 Comparison of our method with shallow and deep kernel predictors.

### 6.6 Validation of separating color and auxiliary features

To validate the impact of separating color and auxiliary feature, we trained a network whose feature extraction uses only one residual network to process color and auxiliary features. Otherwise, the remaining network parameters and training settings were the same as in our full model. Figure 13 shows

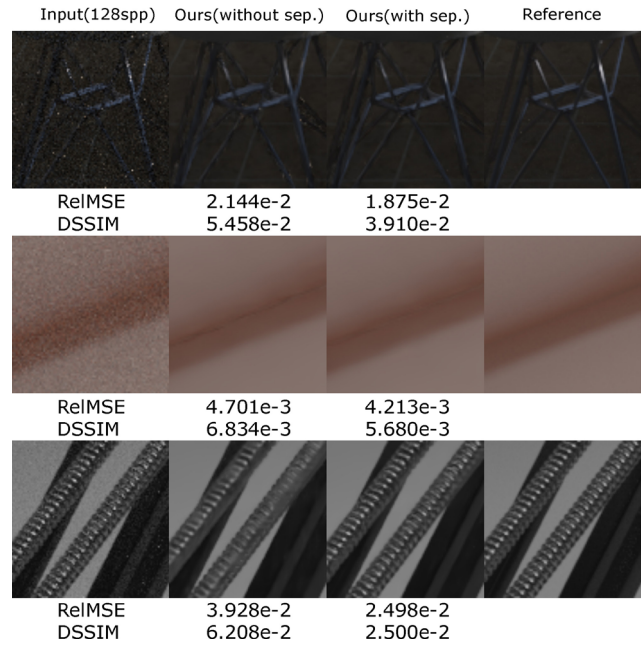


Fig. 13 Comparison of our method with and without separation (sep.) of color and auxiliary features.

that separating color and auxiliary features leads to smoother denoising results and preserve more structural details. The RelMSE and DSSIM measures also show that separating color and auxiliary features leads to better performance: this can help the network to learn more information from the auxiliary feature buffer.

### 6.7 Limitations

We used perceptual loss for training, so that the network can learn the relationship between the denoising result and the reference in terms of high-dimensional features, which can help preserve the sharpness of some geometric details. However there are also some limitations in our method. As shown in Fig. 14, using perceptual loss for training can sometimes make some details of the denoising results too sharp and can result in some artifacts. In future work, we will try to solve this problem by

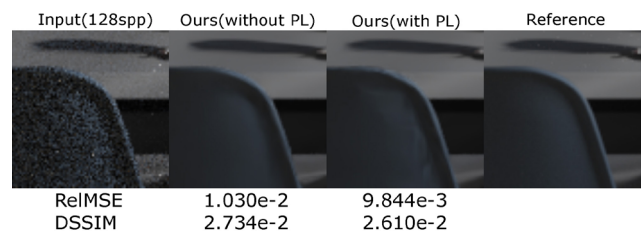


Fig. 14 Limitations of training with perceptual loss.

choosing a more robust perceptual loss function and by controlling its impact with a variable parameter.

## 7 Conclusions

We have presented a novel network for Monte Carlo rendering denoising. Our network decouples features and color, extracts features from them separately, and integrates them into high-dimensional feature information. We add an extra feature for training, based on the covariance of light transport in path space, and a perceptual loss function to preserve details. We then use a shallow neural network to learn kernels, and apply these kernels to produce the denoised picture. Our new algorithm outperforms the state of the art; it is better at preserving details while reducing noise in the picture.

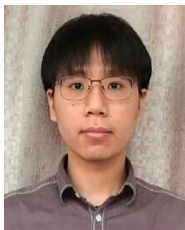
In this paper, we have only considered surface rendering denoising. It would be an interesting research direction to also consider volume denoising. In addition, our model can be exploited for other detail preserving applications, such as edge preservation.

## References

- [1] Keller, A.; Fascione, L.; Fajardo, M.; Georgiev, I.; Christensen, P.; Hanika, J.; Eisenacher, C.; Nichols, G. The path tracing revolution in the movie industry. In: Proceedings of the ACM SIGGRAPH 2015 Courses, Article No. 24, 2015.
- [2] Bako, S.; Vogels, T.; McWilliams, B.; Meyer, M.; Novák, J.; Harvill, A.; Sen, P.; Derose, T.; Rousselle, F. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Transactions on Graphics* Vol. 36, No. 4, Article No. 97, 2017.
- [3] Vogels, T.; Rousselle, F.; McWilliams, B.; Röthlin, G.; Harvill, A.; Adler, D.; Meyer, M.; Novák, J. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics* Vol. 37, No. 4, Article No. 124, 2018.
- [4] Belcour, L.; Bala, K.; Soler, C. A local frequency analysis of light scattering and absorption. *ACM Transactions on Graphics* Vol. 33, No. 5, Article No. 163, 2014.
- [5] Kalantari, N. K.; Bako, S.; Sen, P. A machine learning approach for filtering Monte Carlo noise. *ACM Transactions on Graphics* Vol. 34, No. 4, Article No. 122, 2015.
- [6] Chaitanya, C. R. A.; Kaplanyan, A. S.; Schied, C.; Salvi, M.; Lefohn, A.; Nowrouzezahrai, D.; Aila, T. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics* Vol. 36, No. 4, Article No. 98, 2017.
- [7] Gharbi, M.; Li, T. M.; Aittala, M.; Lehtinen, J.; Durand, F. Sample-based Monte Carlo denoising using a kernel-splatting network. *ACM Transactions on Graphics* Vol. 38, No. 4, Article No. 125, 2019.
- [8] Yang, X.; Wang, D.; Hu, W.; Zhao, L.-J.; Yin, B.-C.; Zhang, Q.; Wei, X.-P.; Fu, H. DEMC: A deep dual-encoder network for denoising Monte Carlo rendering. *Journal of Computer Science and Technology* Vol. 34, 1123–1135, 2019.
- [9] Sen, P.; Zwicker, M.; Rousselle, F.; Yoon, S.-E.; Kalantari, N. Denoising your Monte Carlo renders: Recent advances in image-space adaptive sampling and reconstruction. In: Proceedings of the ACM SIGGRAPH 2015 Courses, Article No. 11, 2015.
- [10] Sen, P.; Darabi, S. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Transactions on Graphics* Vol. 31, No. 3, Article No. 18, 2012.
- [11] Rousselle, F.; Manzi, M.; Zwicker, M. Robust denoising using feature and color information. *Computer Graphics Forum* Vol. 32, No. 7, 121–130, 2013.
- [12] Moon, B.; Jun, J. Y.; Lee, J.; Kim, K.; Hachisuka, T.; Yoon, S. E. Robust image denoising using a virtual flash image for Monte Carlo ray tracing. *Computer Graphics Forum* Vol. 32, No. 1, 139–151, 2013.
- [13] Zimmer, H.; Rousselle, F.; Jakob, W.; Wang, O.; Adler, D.; Jarosz, W.; Sorkine-Hornung, O.; Sorkine-Hornung, A. Path-space motion estimation and decomposition for robust animation filtering. *Computer Graphics Forum* Vol. 34, No. 4, 131–142, 2015.
- [14] Moon, B.; Carr, N.; Yoon, S.-E. Adaptive rendering based on weighted local regression. *ACM Transactions on Graphics* Vol. 33, No. 5, Article No. 170, 2014.
- [15] Bitterli, B.; Rousselle, F.; Moon, B.; Iglesias-Gutián, J. A.; Adler, D.; Mitchell, K.; Jarosz, W.; Novák, J. Nonlinearly weighted first-order regression for denoising Monte Carlo renderings. *Computer Graphics Forum* Vol. 35, No. 4, 107–117, 2016.
- [16] Moon, B.; McDonagh, S.; Mitchell, K.; Gross, M. Adaptive polynomial rendering. *ACM Transactions on Graphics* Vol. 35, No. 4, Article No. 40, 2016.
- [17] Boughida, M.; Boubekeur, T. Bayesian collaborative denoising for Monte Carlo rendering. *Computer Graphics Forum* Vol. 36, No. 4, 137–153, 2017.
- [18] Liang, Y. L.; Wang, B. B.; Wang, L.; Holzschuch, N. Fast computation of single scattering in participating

media with refractive boundaries using frequency analysis. *IEEE Transactions on Visualization and Computer Graphics* doi: 10.1109/TVCG.2019.2909875, 2019.

- [19] Durand, F.; Holzschuch, N.; Soler, C.; Chan, E.; Sillion, F. X. A frequency analysis of light transport. *ACM Transactions on Graphics* Vol. 24, No. 3, 1115–1126, 2005.
- [20] Belcour, L.; Soler, C.; Subr, K.; Holzschuch, N.; Durand, F. 5D covariance tracing for efficient defocus and motion blur. *ACM Transactions on Graphics* Vol. 32, No. 3, Article No. 31, 2013.
- [21] Simonyan, K.; Zisserman, A. Two-stream convolutional networks for action recognition in videos. In: *Proceedings of the Advances in Neural Information Processing Systems 27*, 568–576, 2014.
- [22] Yang, Q. S.; Yan, P. K.; Zhang, Y. B.; Yu, H. Y.; Shi, Y. Y.; Mou, X. Q.; Kalra, M. K.; Zhang, Y.; Sun, L.; Wang, G. Low-dose CT image denoising using a generative adversarial network with Wasserstein distance and perceptual loss. *IEEE Transactions on Medical Imaging* Vol. 37, No. 6, 1348–1357, 2018.
- [23] Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [24] Bitterli, B. Tungsten renderer. Available at <http://noobody.org/tungsten.html>.
- [25] Bitterli, B. Rendering resources. 2016. Available at <https://benediktbitterli.me/resources/>.
- [26] Abadi, M.; Agarwal, A.; Barham, P. Tensorflow: Large scale machine learning on heterogeneous systems. 2015. Available at <http://tensorflow.org/>.
- [27] Kingma, D. P.; Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [28] Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 249–256, 2010.



**Weiheng Lin** is a master candidate in the School of Computer Science and Engineering, Nanjing University of Science and Technology (NJUST). He received his bachelor degree from NJUST in 2018. His research interests include rendering and machine learning.



development.

**Beibei Wang** is an associate professor at NJUST. She received her Ph.D. degree from Shandong University in 2014 and visited Telecom ParisTech from 2012 to 2014. She worked as a postdoc in INRIA from 2015 to 2017. She joined NJUST in March 2017. Her research interests include rendering and game



**Lu Wang** is a professor at the School of Software, Shandong University. She received her Ph.D. degree from Shandong University in 2009. Her research interests include photorealistic rendering and high performance rendering.



**Nicolas Holzschuch** is a senior researcher at INRIA Grenoble Rhône-Alpes, and the scientific leader of the MAVERICK research team. He received his Ph.D. degree from Grenoble University in 1996 and his habilitation in 2007. He joined INRIA in 1997. His research interests include photorealistic rendering and real-time rendering, with an emphasis on material models and participating media.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made.

The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Other papers from this open access journal are available free of charge from <http://www.springer.com/journal/41095>. To submit a manuscript, please go to <https://www.editorialmanager.com/cvmj>.