A Detail Preserving Neural Network Model for Monte Carlo Denoising

Weiheng Lin, Beibei Wang, Lu Wang, Nicolas Holzschuch

Computational Visual Media
DOI 10.1007/s41095-xxx-xxxx-x Vol. x, No. x, month year, xx–xx

Article Type (Research/Review)

A Detail Preserving Neural Network Model for Monte Carlo Denoising

Weiheng Lin[1], **Beibei Wang**[1](✉), **Lu Wang**[2](✉), **and Nicolas Holzschuch**[3]

Abstract Monte Carlo based methods such as path tracing are widely used in movie production. To achieve noise-free quality, they tend to require a large number of samples per pixel, resulting in longer rendering time. To reduce that cost, one solution is Monte Carlo denoising: render the image with fewer samples per pixel (as little as 128) and then denoise 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 have shown powerful denoising ability. However, they tend to lose geometric details or lighting details and over 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. In our network, we separate feature buffers with color buffer to enhance detail effects. Their features are extracted separately and then are integrated to a shallow kernel predictor. Our loss function considers perceptual loss, which also improves the detail preserving. In addition, we present the light transport covariance feature in path space as one of the features, which is used to preserve illumination details. Our method denoises Monte Carlo path traced images while preserving details much better than previous work.

Keywords Deep learning, light transport covariance, perceptual loss.

1 Introduction

Monte Carlo based methods are widely used for rendering in movie production [14], as they are physically based and are able to produce unbiased results. However, they require a large number of samples per pixel to produce noise-free results. To save the rendering cost, one solution is generate a noisy image with only a few 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] use a convolutional neural network (CNN) to predict the final denoised pixel value as a highly non-linear combination of the 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, they learn a kernel for each pixel and apply the kernel to neighbors of each pixel to reconstruct the denoised color. Vogels et al. [25] further improve 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, etc., to avoid limitations of inputs. They also solve the temporal coherency issue between different images.

These methods are very efficient for denoising Monte Carlo rendered images, but they tend to remove details (see Figure 1), decreasing the quality of the resulting

1 School of Computer Science and Engineering, Nanjing University of Science and Technology, 210094, Nanjing. E-mail: 442920398@qq.com, beibei.wang@njust.edu.cn.
2 School of Computer Science and Technology, Shandong University, 250100, Jinan. 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: 2020-2-6; accepted: 2020-02-23.

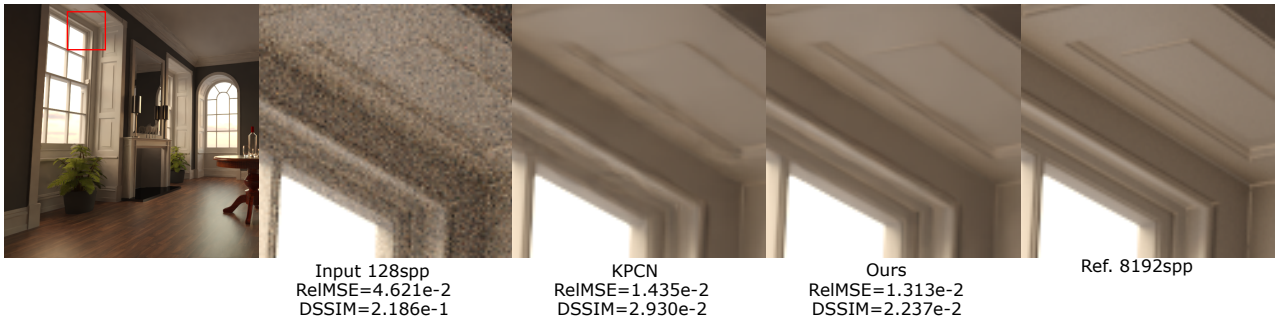


Fig. 1 Comparison between our network and Kernel Predicting Convolutional Network (KPCN) [2]. KPCN and our model use the same dataset for training. Our model preserves details better, due to the novel network structure, a new feature (light transport covariance in path space) and the perceptual loss function. The error metrics (RelMSE and DSSIM) also confirm the higher quality of our method.

image. Details can come from the geometry (see Figure 1) or from lighting effects (see Figure 9). Existing denoising algorithms capture details by extracting features from the color buffer and auxiliary buffers such as position and normals. However, details might only 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 and normal buffer, and complex geometry would have the opposite situation. Training on all the features together results in the over-blurring we observe.

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

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

2 Previous work

2.1 Machine learning based Monte Carlo denoising

Kalantari et al. [13] introduced neural network 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 the local weighting kernels to filter pixels from their neighbors. Their method is called *KPCN*. They decompose input into diffuse and specular components and train CNN models separately. The KPCN method is more efficient then earlier Monte Carlo denoisers. Vogel et al. [25] further improved denoising by combining KPCN with a number of task-specific modules, e.g. source-aware encoder, and optimizing the assembly using an asymmetric loss, resulting in a more robust solution.

Chaitanya et al. [9] proposed a recurrent neural network (RNN) model considering the temporal coherency for interactive renders.

Gharbi et al. [11] 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. [27] proposed a Dual-Encoder network. The method fuse feature buffers by a feature fusion sub-network firstly, then encode the fused feature buffers and color buffer separately, and finally reconstruct a clean image by a decoder network.

Compare to Yang et al. [27], our method does not fuse auxiliary feature buffers at first and add light transport covariance buffer which represent the frequency of the

light transport. We use residual network filter the color buffer and auxiliary feature buffers separately, then integrate their feature maps to a shallow kernel predictor network. Hence our algorithm is based on kernel predicting method instead of 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 [22].

Zero-order linear regression model based methods [21] [20] [17] [28] use 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 makes their filter kernel less flexible.

First-order models [16] [7] or high-order models [18] 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 neighboring data. First order methods have problem dealing with low frequency noise, and high-order methods might suffer from over-fitting.

Boughida et al. [8] propose a non-local Bayesian collaborative filter, which produces globally high denoising quality, especially in dark areas.

3 Background

3.1 Problem statement

The problem of denoising Monte Carlo rendering can be formulated as:

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

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

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

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

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

3.2 Kernel prediction convolutional network

Bako et al. [2] 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 trained with individual CNN network which outputs kernels separately. With the predicted kernel, the denoised diffuse and specular are obtained. And then they perform an inverse preprocess transform and combine them to produce the final denoised result. The details can be found in the original paper [2].

Input features. The renderer decomposes rendered outputs into diffuse and specular components. The rendered outputs includes 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.

Network architecture. KPCN uses a vanilla 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 Relu activation function. In the last layer, it outputs a $K \times K$ kernel of scalar weights instead of directly outputting a denoised pixel.

Loss function We know that the loss function should be able to get the perceptual difference between the estimated and reference color well and be easy to optimize. KPCN chose L_1 loss to optimize their network. They experimented with several loss functions, including L_1, relative (rel) L_1, L_2, rel L_2, and SSIM (Structural Similarity). The experimental results show that the optimization of the L_1 loss function is the best:

$$l_1 = |c_{\text{denoised}} - c_{\text{reference}}|. \tag{3}$$

3.3 Light transport covariance in path space

Durand et al.[10] 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 dimensions in space and two dimensions in angle (see Figure 2). Standard operations

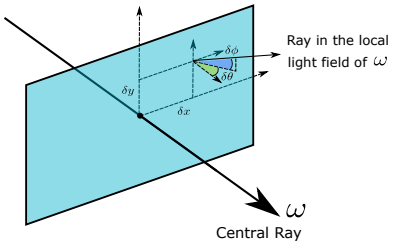


Fig. 2 The Local Light Field is defined as a 4D function around the center ray (ω), parameterized by two spatial coordinates (δ_x and δ_y) and two angular coordinates (δ_θ and δ_ϕ) [15]

on light transport, such as transport in free space or reflection, 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.[4] introduced an approximate representation for the Fourier spectrum of the local light field: the covariance matrix.

The key idea of Belcour et al.[3] 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 [3] for the detailed computation of these operations.

4 Our method

4.1 Network architecture

Our network consists of four parts (Figure 3(a)): data preprocessing (Sec. 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 diffuse, applying logarithmic transform to specular, scaling depth to the range[0,1] and taking gradient for all buffers including diffuse, specular, normal, albedo and depth, with the addition of a light transport covariance feature (see Sec. 4.2).

In feature extraction, we first separate diffuse and specular components into color component and feature component respectively, to enhance details capturing, inspired Simonyan and Andrew [23]. Then each component is sent to a feature extractor, which is a residual network (Figure 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.[25], 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 up. 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 residual network protects data integrity by directly passing input data to the output (skip connection) 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 (Figure 3(c)), which consists of only four traditional convolutional layers. Two kernel predictors output two 21×21 kernels to denoise diffuse and specular buffers separately. We use a shallow network rather than a deep network, as a deep network makes the optimization of feature extractor more difficult, leading to degradation of the training quality.

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

4.2 Light transport covariance feature

We introduce light transport covariance by Belcour et al.[3] as one of the input features, as it can represent the frequency of the light transport to help detail preserving.

The covariance matrix is denoted as Σ. For a function f defined over a 4D domain, it is a 4×4 matrix defined by:

$$\Sigma_{i,j} = \frac{1}{\int_\Omega f} \int_{x \in \Omega} (x \cdot e_i)(x \cdot e_j) f(x) dx, \forall (i,j) \in \{1, \cdots, 4\}^2, \tag{4}$$

where e_i is the i^{th} vector of the canonical basis of the 4D space Ω and $x \cdot y$ is the dot product of vectors x and y.

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

Then we compute the determinant of the covariance matrix, denoted as η, and defined by:

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

η goes from 0 to 1. The higher the value of η, the larger the frequency content at this location. $\eta = 0$ corresponds to a uniform, constant distribution (low frequency), $\eta = 1$ corresponds to a Dirac (high frequency). We use this determinant of the covariance matrix as a feature for training. This feature benefits

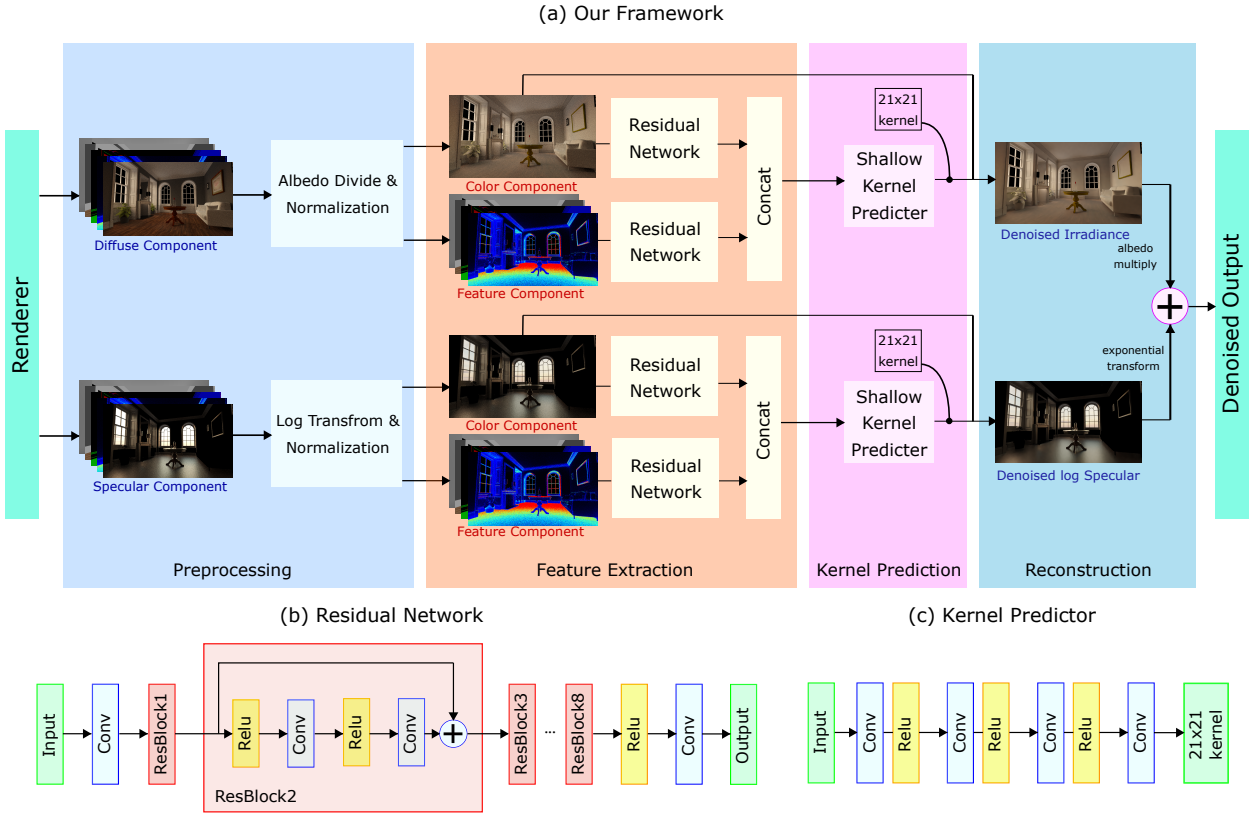


Fig. 3 (a) Our network framework. The renderer decomposes rendered outputs into diffuse and specular components. The two components are preprocessed independently. In both components, their features are separated into color component and feature component. These two components are fed into a residual network receptively to extract features and then the extracted features are concatenated. In the next step, two kernel predictor networks filter the extracted features and output two 21×21 kernels, which are used to denoise preprocessed diffuse and specular buffers. Finally, the denoised diffuse / specular 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.

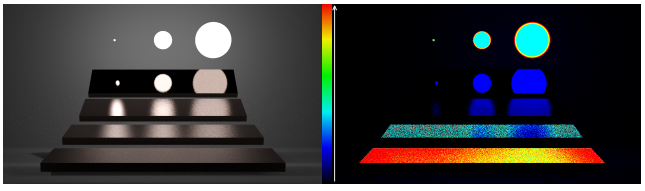


Fig. 4 An example of light transport covariance buffer. The left image is the full color buffer, and the right image is the corresponding light transport covariance buffer.

the complex lighting detail preservation (see Figure 9). Figure 4 shows a visualization of this feature.

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 ε is a small number, which is 10^{-8} in our implementation.

We also include the perceptual loss l_p:

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

where ϕ is a feature extractor, w, h, and d represent the width, height and depth of the denoised image respectively. Similar to [26], we use pre-trained VGG-19 [24] as the feature extractor ϕ, as VGG-19 can get high-dimensional feature information of the image.

The perceptual loss helps in preserving more details in the denoised image (see Figure 11).

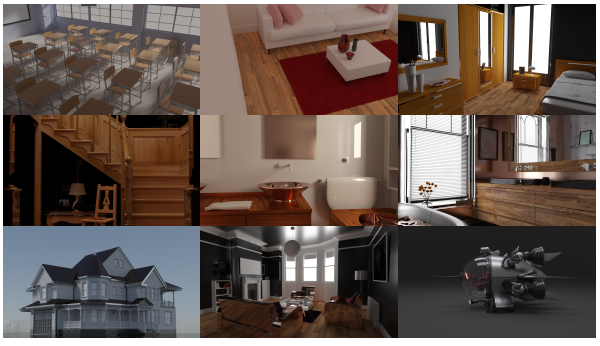


Fig. 5 Some example images from our dataset. We modify camera, materials, and light sources of some publicly available scenes to enrich our dataset.

5 Data Creation and Training

5.1 Data creation

For training, we rendered images and buffers with the Tungsten renderer [5], as our dataset. As known, training a neural network requires a large and representative dataset to avoid overfitting. So in order to generate a lot of data, we modify publicly available scenes [6] (see Figure 5) by varying camera parameters, materials, and light sources. The noisy images are rendered with 32 spp (samples per pixel) or 128 spp, and the reference images are rendered with 8192 spp. The resolution of these images is 1280×720. Finally, we rendered about 220 scenes as our training set and about 20 scenes as our validation set.

Similarly to Bako et al. [2], we decompose rendered outputs into diffuse and specular buffers. In addition to the feature buffer mentioned in KPCN, we add a light transport covariance feature buffer (see Figure 4) (1 channel). The renderer outputs 20 channels in total (diffuse, specular, albedo, normal, depth, light transport covariance and their corresponding variance). We factor out the albedo from the diffuse channel and apply a logarithmic transform to specular channel. We take the 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 implement our network in TensorFlow [1] and use ADAM [19] optimizer to optimize the parameters. Weights were initialized using the Xavier method [12].

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

6 Results

We compare our result to four state-of-the-art methods: NFOR [7], KPCN [2], BCD [8], DEMC [27] and reference images. We use DSSIM (Structural Dissimilarity) and RelMSE (relative Mean Squared Error) as metrics to evaluate the results quality. The input images were rendered with 32-128 spp, and the references were rendered with 8912-20000 spp.

6.1 Model validation

In Figure 6, we compare our model with four other methods representative of the state of the art: NFOR [7], KPCN [2], BCD [8], DEMC [27] and reference images. According to the error metrics, our model produces higher quality and preserves details better. NFOR blurs the details of textures and lighting, and produces artifacts at low frequency noise. BCD still has some noise in many geometric details. Compared to NFOR and BCD, KPCN has better overall denoising effect, 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 Figure 8, we show the error as a function of iterations for KPCN and our model. From 1K iteration, we perform validation every 2K iterations and calculate RelMSE. Our method has smaller error than KPCN all the time.

6.2 Model structure validation

In Figure 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 methods. According to the error metrics, our model produces higher quality and preserves details better, while KPCN has some aliasing or blurring in some details.

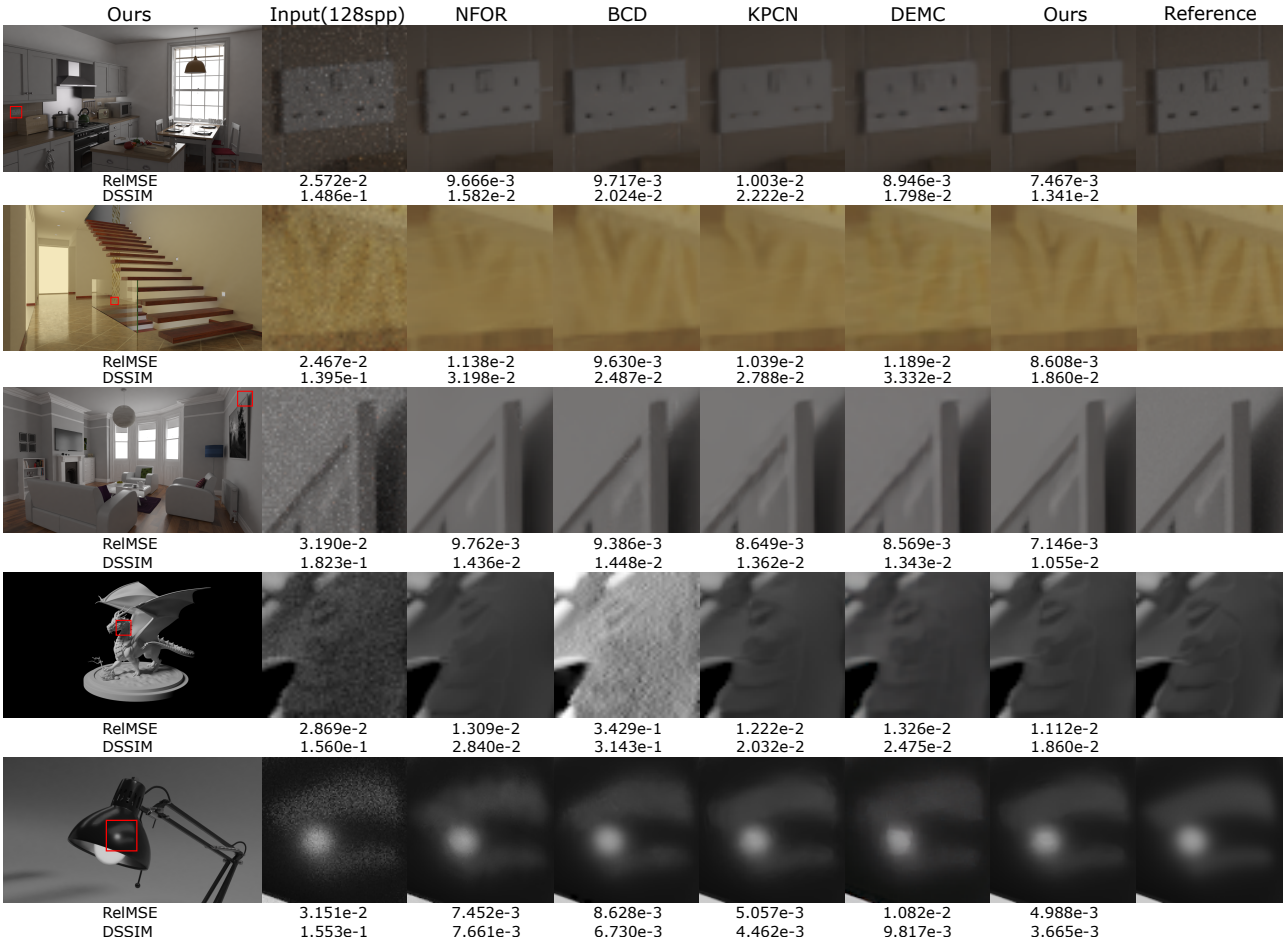


Fig. 6 Comparison between our method, other four state-of-the-art methods NFOR [7], KPCN [2], BCD [8], DEMC [27] and reference images. KPCN's input features and loss function are the same as the original paper (Sec. 3.2). Our model includes a light transport covariance, besides the features of KPCN, and is trained with loss function in Sec. 4.3. KPCN, DEMC and our model have same other training settings (see Sec. 5.2) and use the same dataset for training.

6.3 Light transport covariance buffer validation

We validate the impact of light transport covariance buffer for scenes with complex lighting. In Figure 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 the light transport covariance improves significantly in the handling of high frequency details. Light transport covariance can represent the frequency of the light transport so that neural network can learn more features of high frequency light details. As shown in Figure 9, the caustics, glossy and specular details are preserved better with the light transport covariance buffer.

In Figure 10, we show the impact of the number of samples per pixel (spp) on denoising quality with the light transport covariance buffer. Our networks are trained with only SMAPE loss, to validate the effect

Tab. 1 The cost of light transport covariance. We implement light transport covariance in Tungsten renderer and experimented with four scenes. These scenes are rendered with 128spp and 512x512 resolution.

Scene	Time without cov.	Time with cov.	Cost
Bathroom	3m07s	3m32s	+13.37%
Classroom	2m53s	3m15s	+12.72%
Kitchen	3m30s	3m50s	+9.52%
Living-room	2m56s	3m17s	+11.93%

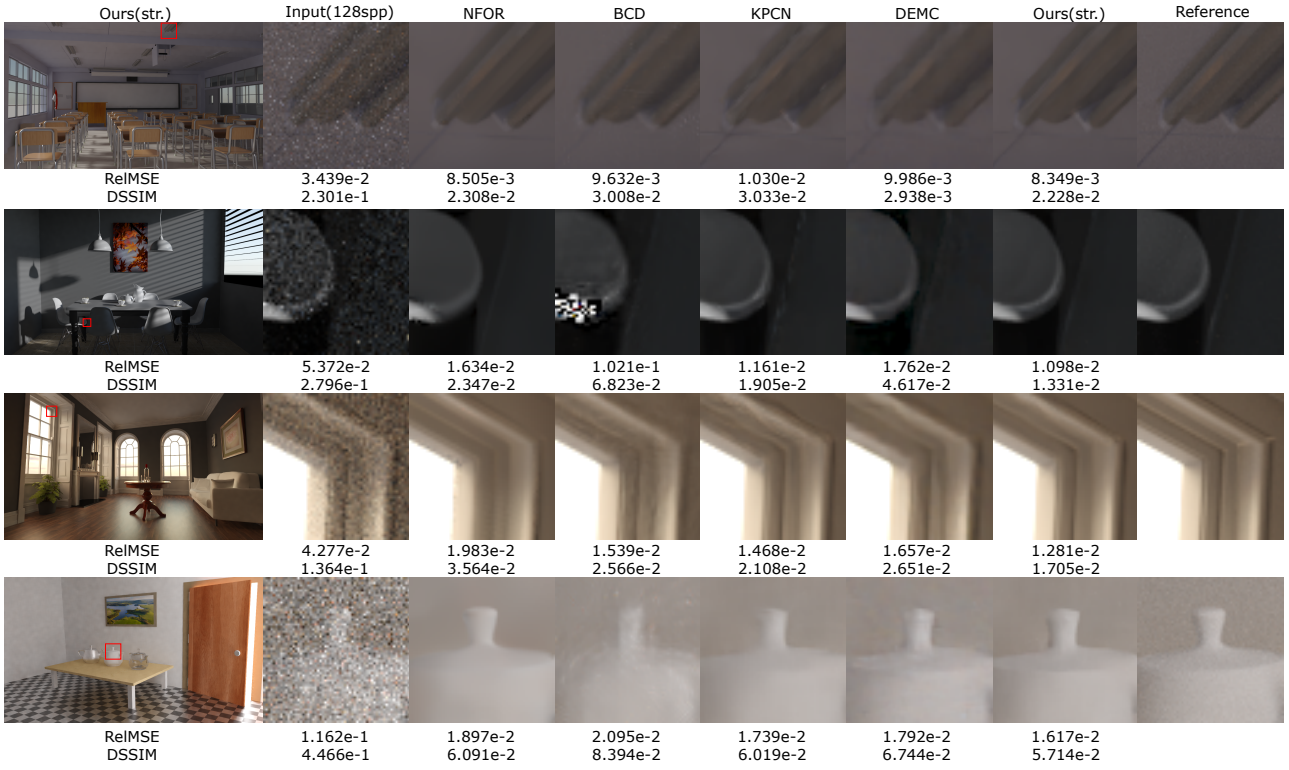


Fig. 7 Network structure comparison between our model(str. means network structure only) and previous works. To validate the effect of network structure, our network training does not use light transport covariance and perceptual loss. KPCN, DEMC and our method have the same training settings (See Sec.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

of light transport covariance. The test data consists of several scenes. We render the scenes at different sample count (8, 16, 32, 64 and 128 spp) and filter them with four models (our model training with / without light transport covariance, KPCN training with / without light transport covariance). Then the error between the filtered results and the references is calculated and averaged. Our method with covariance produces the best result for any level of noise input. In addition, light transport covariance can help improve the denoising quality for both our method and KPCN, especially in the case of low numbers of samples per pixel.

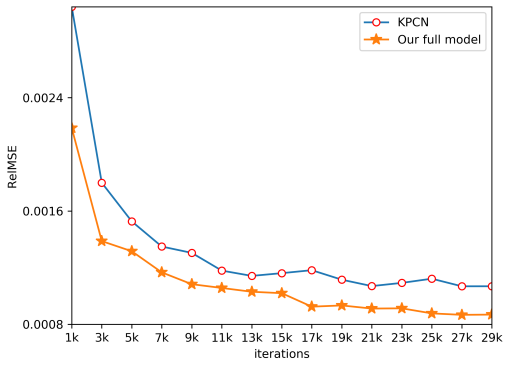


Fig. 8 Comparison of RelMSE as a function of training iterations of our method and KPCN. Our method is always better than KPCN at any iteration.

6.4 Loss function validation

To validate the effect of perceptual loss used in our loss function, we compare our method with and without perceptual loss in Figure 11. With the perceptual loss, the geometric details have been further restored, which is closer to the reference than the denoised results only trained with SMAPE. Training with perceptual loss can help the denoising result be similar to the reference on high-level features, so it can make some 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 Figure 12. The shallow network works better than deep network. A deep network makes the optimization of feature extractor more difficult, leading to degradation of the training

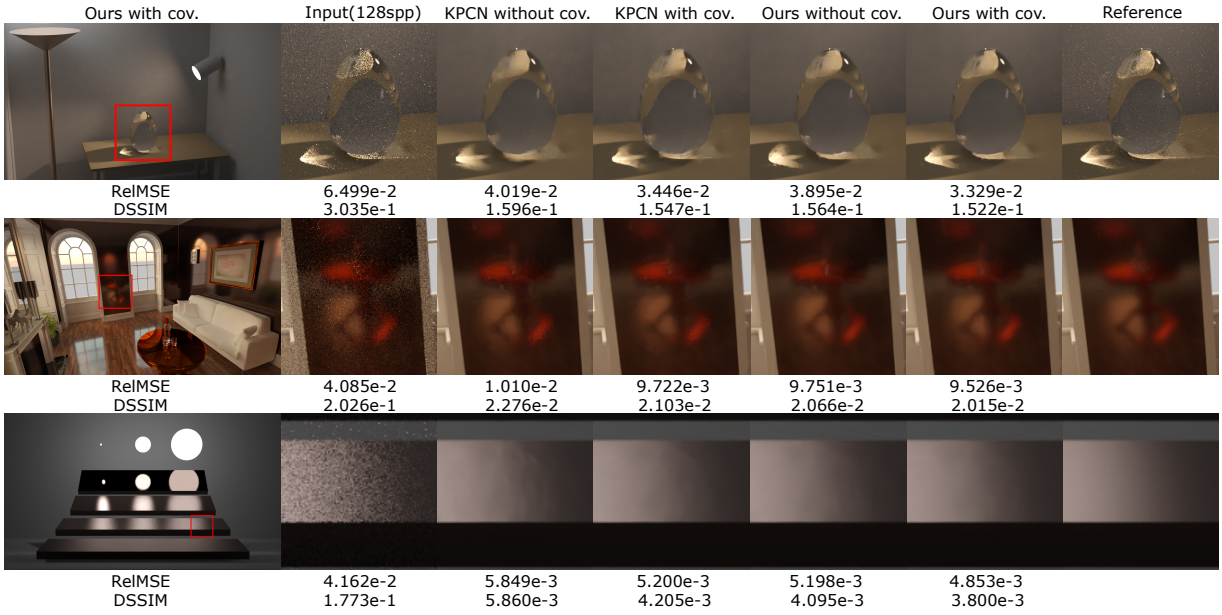


Fig. 9 Comparison of training with or without light transport covariance. We respectively use the feature buffer with and without light transport covariance to train KPCN and our method. Some scenes with special details are chosen to show the performance of training with the light transport covariance(cov. means light transport covariance).

quality. Therefore, we use a shallow network for kernel prediction, for better performance and reducing the amount of network parameters.

6.6 Separating color and auxiliary feature validation

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 feature. In addition, the remaining network parameters and training settings are the same as our full model. In Figure 13, training with separating color and auxiliary feature can make denoising result smoother and preserve more structure details. The result of RelMSE and DSSIM also shows that training with separating color and auxiliary feature have better performance. Thus separating color and auxiliary feature can help the network to learn more information from the auxiliary feature buffer.

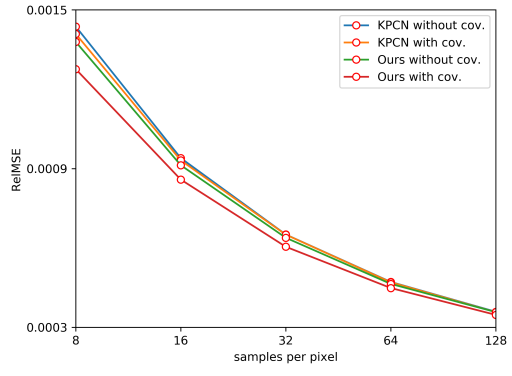


Fig. 10 RelMSE Comparison between our method (with light transport covariance), our method (without light transport covariance), KPCN (without light transport covariance) and KPCN (with light transport covariance) over varying sample count.

6.7 Limitation

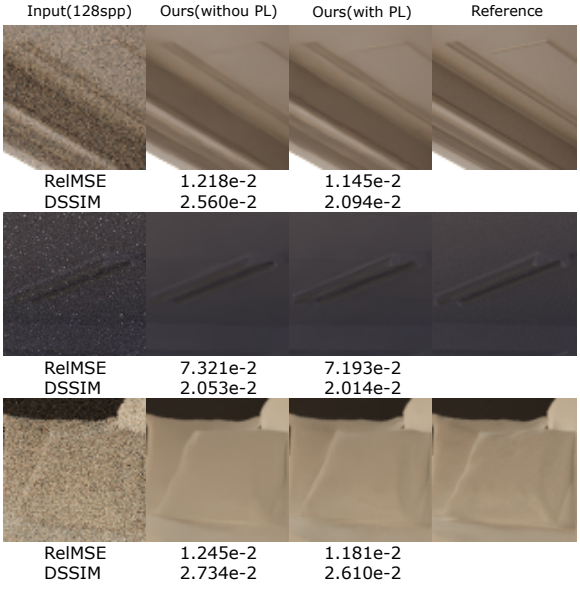


Fig. 11 Comparison of our method training with and without perceptual loss (PL means Perceptual Loss).

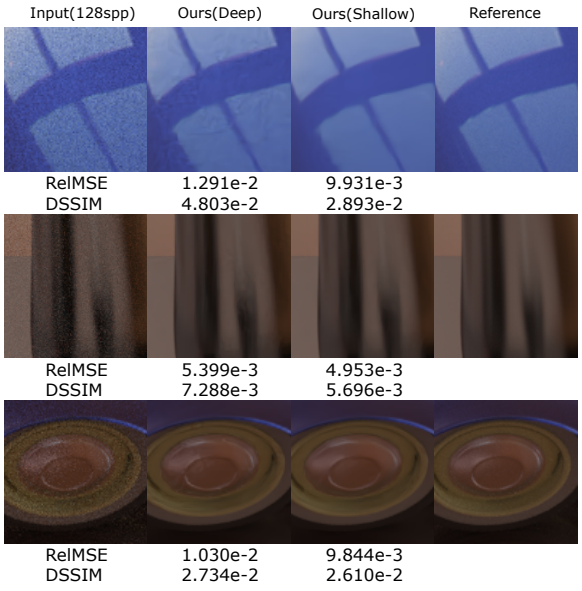


Fig. 12 Comparison of our method between shallow and deep kernel predictor.

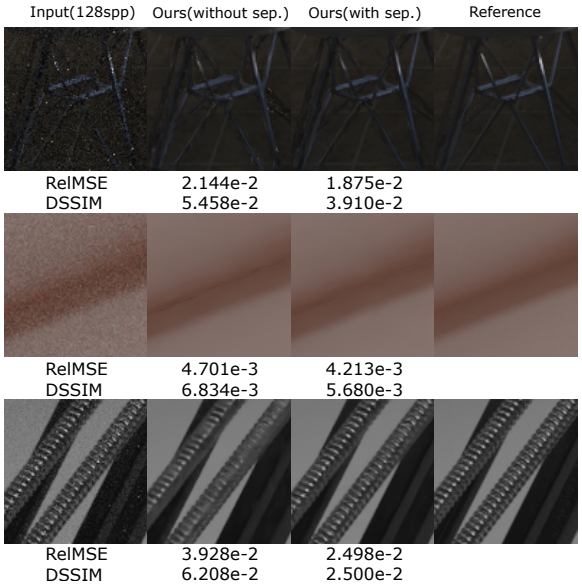


Fig. 13 Comparison of our method training with and without separating color and auxiliary feature (sep. means separating color and auxiliary feature).

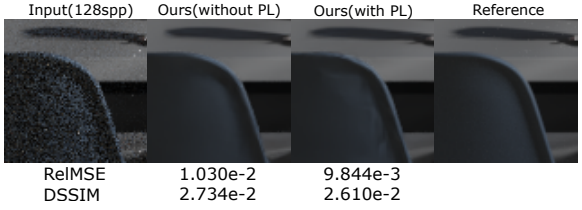


Fig. 14 The limitation of training with perceptual loss.

We used the perceptual loss for training, so that the network can learn the relationship between the denoising result and the reference on 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 Figure 14, using perceptual loss for training can sometimes make some details of the denoising results too sharp and resulting in some artifacts. In future work, we will try to solve this problem by choosing a more robust perceptual loss and controlling the impact of perceptual loss with a variable parameter.

7 Conclusion

We have presented a novel network for Monte Carlo rendering denoising. Our network decouples features and color, extract features from them separately, and integrates them into a 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 only considered surface rendering denoising. It's an interesting research direction to also consider volume denoisings. In addition, our model can be exploited for other detail preserving applications, such as edge preserving.

References

[1] M. Abadi, A. Agarwal, and P. Barham. Tensorflow: Large scale machine learning on heterogeneous systems. `http://tensorflow.org/`, 2015.

[2] S. Bako, T. Vogels, B. McWilliams, M. Meyer, J. Novák, A. Harvill, P. Sen, T. DeRose, and F. Rousselle. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2017)*, 36(4), July 2017.

[3] L. Belcour, K. Bala, and C. Soler. A local frequency analysis of light scattering and absorption. *ACM Transactions on Graphics (TOG)*, 33(5):163, 2014.

[4] L. Belcour, C. Soler, and K. Subr. 5d covariance tracing for efficient defocus and motion blur. *ACM Transactions on Graphics (TOG)*, 32(3):31, 2013.

[5] B. Bitterli. Tungsten renderer. `http://noobody.org/tungsten.html`.

[6] B. Bitterli. Rendering resources. `https://benediktbitterli.me/resources/`, 2016.

[7] 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*, 35(4):107–117, 2016.

[8] M. Boughida and T. Boubekeur. Bayesian collaborative denoising for Monte Carlo rendering. *Computer Graphics Forum (Proc. EGSR 2017)*, 36(4):137–153, 2017.

[9] C. R. Chaitanya, A. S.Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahra, and T. Aila. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.*, 36(4):98:1–98:12, July 2017.

[10] F. Durand, N. Holzschuch, C. Soler, E. Chan, and F. X.Sillion. A frequency analysis of light transport. *ACM Transactions on Graphics (TOG)*, 24(3):1115–1126, 2005.

[11] M. Gharbi, T.-M. Li, M. Aittala, J. Lehtinen, and F. Durand. Sample-based Monte Carlo denoising using a kernel-splatting network. *ACM Trans.Graph.*, 38(4):125:1–125:12, 2019.

[12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *In International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[13] N. K. Kalantari, S. Bako, and P. Sen. A machine learning approach for filtering Monte Carlo noise. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2015)*, 34(4), 2015.

[14] A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. Christensen, J. Hanika, C. Eisenacher, and G. Nichols. The path tracing revolution in the movie industry. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, pages 24:1–24:7, 2015.

[15] Y. Liang, B. Wang, L. Wang, and N. Holzschuch. Fast computation of single scattering in participating media with refractive boundaries using frequency analysis. *IEEE TVCG*, 2019.

[16] B. Moon, N. Carr, and S.-E. Yoon. Adaptive rendering based on weighted local regression. *ACM Trans. Graph*, 33(5):170:1–170:14, 2014.

[17] B. Moon, J. Y. Jun, J. Lee, K. Kim, T. Hachisuka, and S.-E. Yoon. Robust image denoising using a virtual flash image for Monte Carlo ray tracing. *Computer Graphics Forum*, 32(1):139–151, 2013.

[18] B. Moon, S. McDonagh, K. Mitchell, and M. Gross. Adaptive polynomial rendering. *ACM Trans. Graph*, page 10, 2014.

[19] D. P.Kingma and J. Ba. Adam:a method for stochastic optimization. `http://arxiv.org/abs/1412.6980`, 2014.

[20] F. Rousselle, M. Manzi, and M. Zwicker. Robust denoising using feature and color information. *Computer Graphics Forum*, 32(7):121–130, 2013.

[21] P. Sen and S. Darabi. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Transactionson Graphics*, 31(3):15, 2012.

[22] P. Sen, M. Zwicker, F. Rousselle, S.-E. Yoon, and N. Kalantari. Denoising your Monte Carlo renders: Recent advances in image-space adaptive sampling and reconstruction. *ACM SIGGRAPH 2015 Courses*, 2015.

[23] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, pages 568–576, 2014.

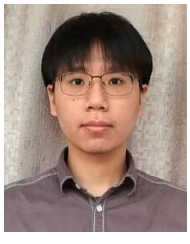
[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[25] T. Vogels, F. Rousselle, B. McWilliams, G. Röthlin, A. Harvill, D. Adler, M. Meyer, and J. Novák. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)*, 37(4):124:1–124:15, 2018.

[26] Q. Yang, P. Yan, Y. Zhang, H. Yu, Y. Shi, X. Mou, M. K.Kalra, Y. Zhang, L. Sun, and G. Wang. Low-dose ct image denoising using a generative adversarial network with wasserstein distance and perceptual loss. *IEEE transactions on medical imaging*, 37(6):1348–1357, 2018.

[27] X. Yang, D. Wang, B. Yin, X. Wei, W. Hu, L. Zhao, Q. Zhang, and H. Fu. Demc: A deep dual-encoder network for denoising monte carlo rendering. *arXiv preprint arXiv:1905.03908*, 2019.

[28] H. Zimmer, F. Rousselle, W. Jakob, O. Wang, D. Adler, W. Jarosz, O. Sorkine-Hornung, and A. Sorkine-Hornung. Path-space motion estimationand decomposition for robust animation filtering. *Computer Graphics Forum*, 34(4):131–142, 2015.



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

Beibei Wang Beibei Wang is an Associate Professor at Nanjing University of Science and Technology. She received her PhD 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 development.

Lu Wang Lu Wang is a Professor at School of Software, Shandong University. She received her PhD from Shandong University in 2009. Her research interests include photorealistic rendering and high performance rendering.

Nicolas Holzschuch Nicolas Holzschuch is a Senior Researcher at INRIA Grenoble Rhône-Alpes, and the scientific leader of the MAVERICK research team. He received his PhD 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.