Computational Visual Media
https://doi.org/10.1007/s41095-019-0142-3 Vol. 5, No. 3, September 2019, 239–255

Research Article

Deep residual learning for denoising Monte Carlo renderings

Kin-Ming Wong[1] (✉), **Tien-Tsin Wong**[2]

Abstract Learning-based techniques have recently been shown to be effective for denoising Monte Carlo rendering methods. However, there remains a quality gap to state-of-the-art handcrafted denoisers. In this paper, we propose a deep residual learning based method that outperforms both state-of-the-art handcrafted denoisers and learning-based denoisers. Unlike the *indirect* nature of existing learning-based methods (which e.g., estimate the parameters and kernel weights of an explicit feature based filter), we *directly* map the noisy input pixels to the smoothed output. Using this direct mapping formulation, we demonstrate that even a simple-and-standard ResNet and three common auxiliary features (depth, normal, and albedo) are sufficient to achieve high-quality denoising. This minimal requirement on auxiliary data simplifies both training and integration of our method into most production rendering pipelines. We have evaluated our method on unseen images created by a different renderer. Consistently superior quality denoising is obtained in all cases.

Keywords Monte Carlo rendering; denoising; deep learning; deep residual learning; filter-free denoising

1 Introduction

Monte Carlo rendering methods have become the mainstream photo-realistic image synthesis technique because of their generality, fast start-up, and progressive nature. Unfortunately, such methods take a prohibitive amount of time to obtain a noise-free image. While light transport techniques [1–3] can accelerate the integration process, noise-free images remain computationally expensive. Image-based denoising techniques have matured quickly in recent years. They take less processing time and are often easy to integrate into existing rendering pipelines. Several post-processing image denoisers have been proposed which achieve high-quality results [4–6].

Recently, learning-based approaches [7, 8] have been demonstrated to provide an effective means to denoising. However, their current results do not show a significant improvement in quality over state-of-the-art handcrafted denoisers. We believe that their mildly incremental improvements are due to the joint filtering model commonly found in mainstream image-based denoisers. The passive roles of their neural networks in filter kernel estimation fail to unleash the full power of deep learning. While the state-of-the-art deep learning method proposed by Ref. [7] requires a large number of auxiliary features, their true benefit is dubious.

In this paper, we propose a *filter-free direct denoising* method based on supervised learning using a *standard-and-simple* deep residual network (ResNet) [9]. Unlike previous learning-based methods which require a large number of auxiliary features, ours requires only *three*: depth, view-space normal, and albedo. We train our simple network to map the noisy inputs directly to high-quality noise-free outputs using our own dataset. The training takes less than 36 hours. Nevertheless, our network outperforms both state-of-the-art learning-based denoisers and carefully handcrafted image denoisers, in terms of visual quality. Figure 1 compares our denoising results with those from two leading denoisers, NFOR [5] and KPCN [7].

1 Artixels, Hong Kong S.A.R., China. E-mail: mwkm@artixels.com (✉).

2 Department of Computer Science and Engineering, the Chinese University of Hong Kong, Hong Kong S.A.R., China. E-mail: ttwong@cse.cuhk.edu.hk.

Manuscript received: 2019-03-12; accepted: 2019-04-14

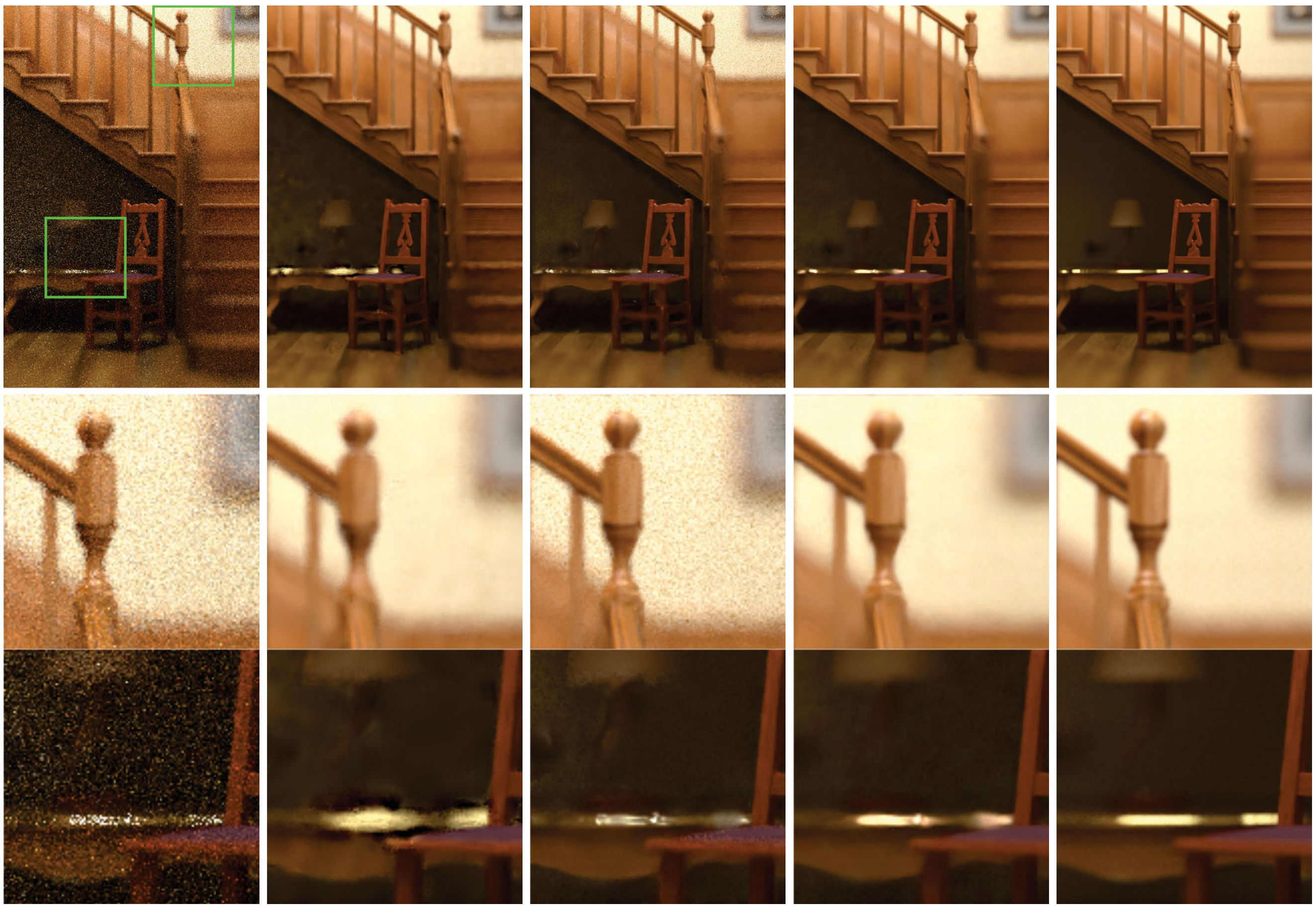


Fig. 1 We propose a *filter-free direct denoising* method based on supervised learning with a deep residual network [9, 10]. Our network takes the noisy image together with depth, screen-space normal, and albedo as input (9 channels total), and it directly outputs a noise-free result *with no intermediate filtering step.* Both our network and KPCN [7] are trained with our own dataset (rendered using RenderMan) which covers diverse shading and distributed effects found in modern rendering methods. The above images compare the denoising performance of our network with other leading methods using a noisy image with *depth of field effect rendered at 8 samples per pixel* using the Tungsten renderer (*The Wooden Staircase* scene by Wig42 from Ref. [11])

The key to achieving such high quality using a simple ResNet and just three auxiliary features is the notion of deep residual learning; its unique design forces the network to learn the difference between its input and the expected output, i.e., the residual. In a supervised learning setting, the network learns to map the differences between the noisy input and the corresponding ground truth. Furthermore, the shortcut connections of ResNet allow reuse of upstream features to establish a multi-scale alike mapping capability. All these features make ResNet a perfect candidate for our denoising task. In addition, the batch normalization [12] layers in ResNet make it resilient to high dynamic range data (typical of our noisy color inputs), and scale well in depth.

To validate our method, we have tested it on a rich variety of scenes (not included in our training data) rendered by a different renderer. Extensive experiments and quantitative evaluation show that our method consistently outperforms other state-of-the-art denoising methods. In short, our contributions are:

- A deep learning based single image denoising method for Monte Carlo rendering. Our simple ResNet generalizes well, and utilizes only three standard auxiliary features as additional input. It integrates transparently into existing rendering pipelines without need for adaptation.
- We identify and demonstrate the benefits of residual learning for high-quality denoising of the output of Monte Carlo rendering methods.

- We demonstrate the advantage of direct denoising using a deep neural network, and identify the importance of auxiliary feature selection and its balance with network capacity.

2 Related work

In this section, we review selected image-based denoising methods and deep learning techniques closely related to our work. A thorough review of denoising Monte Carlo rendering output, including *a priori* methods which study origins of sampling noise in the rendering process, is available in Ref. [13]. Given the wide spectrum of deep learning techniques and applications, even a brief review is beyond the scope of this paper, and we refer to Refs. [14–16] for comprehensive reviews.

2.1 Joint filtering methods

Auxiliary features, such as depth, normal, and albedo computed during most Monte Carlo rendering methods, possess strong correlations with image structures and details seen in the rendered color image. More importantly, such feature data are often considerably less noisy than the rendered image itself, even when the sampling rate is low. Many successful denoising techniques adapt various edge-aware non-linear image filters to leverage this correlation to produce powerful joint filtering methods.

McCool [17] leverages the feature data to produce a coherence map which controls an anisotropic diffusion [18] filtering process. Dammertz et al. [19] adapts auxiliary features as edge-stopping functions in their edge-avoiding wavelet [20] framework. Sen and Darabi [21] adapt the kernel weights of a cross-bilateral filter [22–24] using mutual information, which helps to suppress the influence of random inputs. Li et al. [25] propose use of an SURE estimator [26, 27] to select the best per-pixel result among a set of cross-bilateral filtered candidates created with different bandwidths. Rousselle et al. [6] use both cross-bilateral and non-local means filtering [28] in their framework to produce a set of improved candidates for selection via SURE estimation. First-order regression-based methods using local regression and linear models have been proposed by Refs. [4, 29]. Bitterli et al. [5] use a holistic first-order regression approach, which is considered to be the state-of-the-art joint filtering method.

2.2 Learning-based filtering methods

Regression-based joint filters aim to produce smooth results from the noisy inputs; there is always a risk of over-fitting to the noise. It is known they do not handle highly noisy inputs well. Kalantari et al. [8] propose the first supervised machine learning method to estimate the ideal parameters of their cross-bilateral filter model. Unlike traditional regression-based approaches, a supervised learning model is trained with a large number of noisy and ground truth image pairs. A neural network such as the multilayer perceptron used in Ref. [8] can learn the complex relationship between the noisy inputs and the ground truth. Bako et al. [7] recognize the potential benefits of deep convolutional neural networks, and further delegate the task of determining the ideal filter kernel (bandwidth is preassigned) to the neural network. They also report the difficulties faced by their direct CNN denoising attempt including slow training convergence and potential color shifts.

2.3 Deep learning for inverse problems

Deep convolutional neural networks have demonstrated their great feature extraction power in many difficult image classification problems [30–32]. Supervised learning methods using CNN have also shown impressive results in image denoising [33], and many inverse problems such as inpainting [34], deconvolution [35], and super-resolution [36, 37]. These inverse problems share a common challenge, to reconstruct an output based on inputs with incomplete information. The capability of a CNN is known to directly depend on its depth [38] but it is not a simple matter of stacking more layers to improve capability. The various training difficulties related to deep neural networks have been studied and several practical means [12, 39] exist to tackle them. As a result, denoising Monte Carlo rendering outputs with a deep neural network is likely to meet the challenges of training convergence, the high dynamic range of the image data (both reported in Ref. [7]), and selection of auxiliary features as additional inputs to the network.

3 Direct denoising using a deep residual network

In the following, we present the details of our deep learning based direct denoising approach, and the key design considerations that govern our network architecture and selection of auxiliary features.

3.1 Filter-free direct denoising model

Most regression-based joint filtering methods for denoising Monte Carlo rendering outputs share a generic model which reconstructs the noise-free image by filtering the noisy input colors. They compute the filtered color $\hat{\boldsymbol{c}}_i$ of pixel i as a weighted sum of the colors of pixels in a neighborhood $\boldsymbol{N}(i)$ centered at pixel i:

$$\hat{\boldsymbol{c}}_i = \sum_{j \in \boldsymbol{N}(i)} \boldsymbol{\omega}_{i,j}\, \boldsymbol{c}_j \tag{1}$$

where $\boldsymbol{\omega}_{i,j}$ is the normalized contribution weight of color $\boldsymbol{c}_j$ of pixel j to the result; the exact expression of this weight is determined by the filter model used in a specific method.

Xu and Pattanaik [41] propose one of the earliest joint filters for denoising Monte Carlo rendering output. Their method augments the bilateral filter using pre-filtered pixels for range filtering. The general strategy of joint filtering is to exploit the correlations between various auxiliary features and the color input [42, 43]. In all cases in both the latest techniques [6, 25, 44] which apply error estimation to select filter parameters, and the state-of-the-art regression-based approaches [4, 5], the joint filter variants can be expressed in the following form:

$$\hat{\boldsymbol{c}}_i = \sum_{j \in N(i)} \boldsymbol{F}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{\theta}_{i,j})\, \boldsymbol{c}_j \tag{2}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the inputs based on the selected auxiliary features and $\boldsymbol{c}_j$ is the color of pixel j. $\boldsymbol{\theta}_{i,j}$ represents the filter specific parameters of the kernel function $\boldsymbol{F}(\cdot)$ defined by the method. Recent supervised learning techniques estimate the joint filter parameters [8] or even predict the per-pixel kernel function [7]. However, no matter how sophisticated the joint filter design is, its fundamental filtering formulation stays unchanged and is ultimately dependent on the noisy input colors.

This joint filtering approach unfortunately limits the solution space. One obvious potential consequence is the difficulty of producing good results when the color inputs are very noisy. This is especially common in high dynamic range Monte Carlo renderings as the non-converged samples often exhibit high variance.

In order to take full advantage of deep learning (especially its unparalleled non-linear mapping capability), we propose to solve the Monte Carlo denoising problem by learning a *direct* mapping from the noisy rendered images to the corresponding high-quality noise-free results. Our *filter-free direct denoising model* is expressed as follows:

$$\hat{\boldsymbol{c}}_i = \boldsymbol{G}(\boldsymbol{x}_i, \boldsymbol{X}), \quad \boldsymbol{X} = \{\boldsymbol{x}_j : j \in \boldsymbol{N}(i)\} \tag{3}$$

where $\boldsymbol{G}(\cdot)$ is the mapping to be learned by our method, $\boldsymbol{N}(i)$ is the neighborhood centered at pixel i, and the input is $\boldsymbol{x}_i$. In our method, the feature vector comprises:

$$\boldsymbol{x}_i = \{\boldsymbol{c}_i(r,g,b),\ \boldsymbol{z}_i,\ \boldsymbol{n}_i(x,y),\ \boldsymbol{a}_i(r,g,b)\} \tag{4}$$

with $\boldsymbol{c}_i$, color, $\boldsymbol{z}_i$, depth, $\boldsymbol{n}_i(x,y)$, view-space normal, and $\boldsymbol{a}_i(r,g,b)$, albedo, of a noisy input pixel i. In order to learn such a challenging mapping, we need a neural network which is easy to train, and has the capacity to realize that highly non-linear mapping. In the next section, we present a network architecture which has a proven record of dealing with natural image inverse problems of a similar nature to our denoising problem. We then discuss the rationale behind the selection of features in Section 3.3.

3.2 Network architecture

We consider the key characteristics of our problem relevant to network architecture and training. The two primary concerns identified are as follows:

1. The network is expected to map from the noisy inputs directly to the corresponding noise-free results while exploiting the correlations between the auxiliary features and noisy color inputs.
2. Monte Carlo rendered image data have high dynamic range, which have the potential to cause instability during training.

The first concern indicates the need for a network capable of learning a complex mapping, and is good at denoising-like tasks. The second concern over stability during training is due to the potentially large changes in inputs during training. Overall, we need a network which scales well with depth [31, 38] in terms of both learning capacity and stability of training.

We propose to use a deep residual learning (ResNet) [9] based architecture for our method, as shown in Fig. 2. This type of network has shown good performance on several inverse problems [37, 40, 45]. Deep residual learning pioneered by He et al. [9] set several records in image recognition challenges. The depth of a ResNet can reach over one hundred

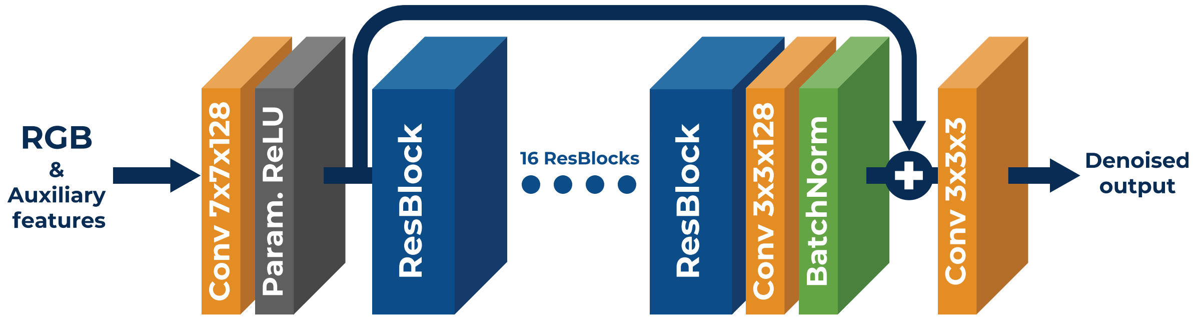


Fig. 2 Our network is based on the ResNet architecture [9]. We use 16 *ResBlocks* (basic ResNet building blocks). Various *ResBlock* variants exist [10, 37, 40]. Experiments showed that our current choice (see Fig. 3) performs best for our application in terms of both training efficiency and quality of output. There are a total of 35 convolutional layers, and 19,592,704 trainable parameters in our network.

convolutional layers with continued improvement in performance.

The basic building block of ResNet is a small unit of stacked convolutional layers with a shortcut connection, shown as a *ResBlock* in Fig. 3. If we expect a *ResBlock* to learn a mapping $\boldsymbol{H}(\boldsymbol{x})$ with an input $\boldsymbol{x}$, the stacked layers inside are now effectively learning a residual mapping $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{H}(\boldsymbol{x}) - \boldsymbol{x}$. This recasting makes certain mappings easier to learn, for example an identity mapping, i.e., $\boldsymbol{H}(\boldsymbol{x}) = \boldsymbol{x}$, and $\boldsymbol{F}(\boldsymbol{x}) = 0$. In this case the *ResBlock* has to learn nothing. An identity mapping can be hard to learn for an ordinary convolutional network.

This shortcut connection in practice allows an upstream block to share its input data with any downstream block via a coordinated identity mapping if desired. This unique feature is actually the key which makes ResNet a powerful mapping learner because data are free to flow across the network instead of in a layer-by-layer fashion.

The residual learning capability also makes ResNet an ideal candidate for the denoising task in a supervised learning setting as it can focus on learning

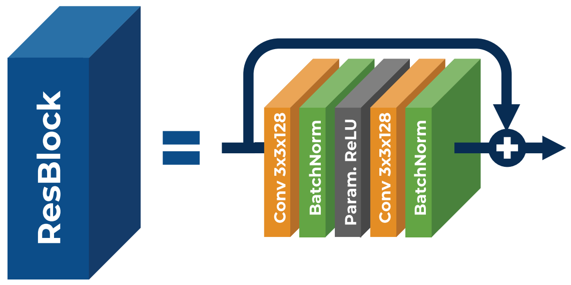


Fig. 3 Inside our *ResBlock*, there are 2 convolutional layers, each having 128 filters of size 3×3. Each convolutional layer is followed by a batch normalization layer [12] to form a sub-unit, and a parametric rectifier [32] is sandwiched between these two sub-units.

and mapping the differences between the noisy input and the corresponding ground truth at different scales. Each *ResBlock* includes a batch normalization layer [12] which improves training stability by suppressing the internal covariate shift [12] caused by large changes in inputs between layers. Although He et al. [10] propose an improved *ResBlock*, we found that the one proposed in Ref. [37], which includes the use of a parametric rectifier [32] as an activation unit, performed best in our experiments. We also include a network wide skip connection [10, 37, 46] for added mapping flexibility.

3.3 Auxiliary feature selection and pre-processing

In our filter-free model, there is no predefined filter which governs the choice of auxiliary features to be learned by the neural network, unlike in Ref. [8]. Bako et al. [7] use a 27-channel input to each of their filter pipelines, presumably to maximize the potential benefits of using more auxiliary features. However, the overall benefits of auxiliary feature inputs depend naturally on the learning capacity of the network, and we suspect it may be incapable of learning such a complicated mapping. We performed a simple experiment using the diffuse filter pipeline of KPCN [7]. The color, depth, and normal related auxiliary features were selected to form an 11-channel subset from their original input to train the same network. Figure 4 shows that the KPCN network trained with the subset input achieves a lower training loss, and the 11-channel trained network delivers similar and a few better denoising results, but we did not pursue this further. This simple experiment reflects the importance of evaluating a network's capacity relative to its input.

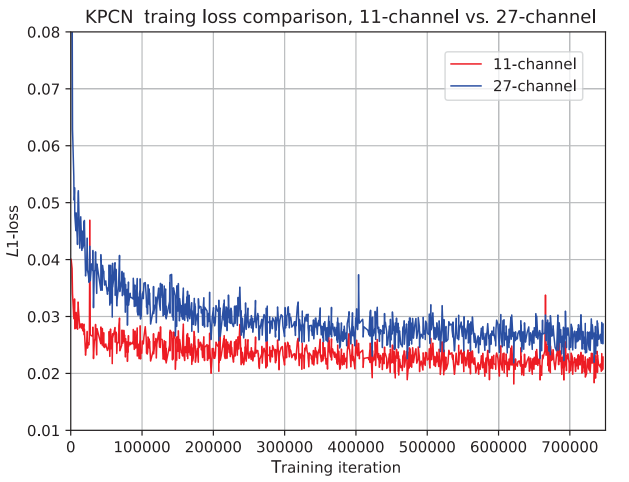


Fig. 4 Comparison of the KPCN (diffuse pipeline) [7] training loss for an 11-channel subset of the input (color, depth, and normal related only) and the original 27-channel input.

In order to verify the candidate auxiliary features we planned to use, we relied on a smaller *8-ResBlock* ResNet (2 convolutional layers of 64 filters of size 3×3 per block) to evaluate their usefulness. Figure 5 shows the impact of selected combinations of auxiliary features on the $L1$ training loss. Our candidate auxiliary features, depth, view-space normals, and albedo are shown to be useful but albedo provides only marginal improvement to the loss. Furthermore, the inclusion of view-space normals and albedo seems to accelerate training convergence.

From a practical point of view, the selection of auxiliary features should also consider the ease of adoption as mentioned in Ref. [5], so we chose the most common ones which are readily available from most renderers. In a supervised learning setting, the ground truth images already provide much information, and we believe the per-pixel statistical information may not be as useful to us as they are in the regression-based methods. In addition, the feature extraction power of CNNs is well recognized [30, 47], so we chose to include only primary auxiliary features.

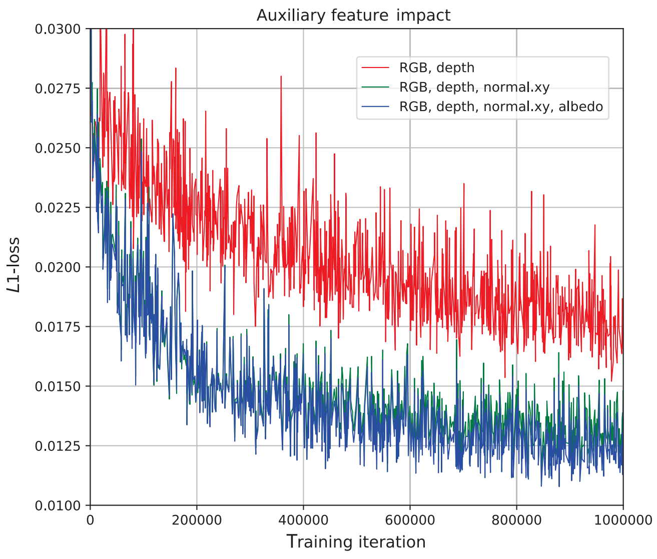


Fig. 5 Impact of auxiliary features on a smaller ResNet similar to our network.

Lastly, we follow Ref. [7] to apply range compression via logarithmic transform as a pre-processing step to our high dynamic range inputs, i.e., color and depth. This compression step improves the results in terms of smoothness. In short, our 9-channel input for training is as follows:

- noisy color (3 channels, range compressed),
- depth (1 channel, range compressed),
- view-space normals (2 channels),
- albedo (3 channels).

4 Implementation and network training

In this section, we present details of training data preparation, and implementation of our network model. We also discuss the impact of loss functions and concerns about the capacity of our network.

4.1 Training data preparation

To the best of our knowledge, there is no publicly available training dataset dedicated to the Monte Carlo rendering denoising task. Knowing the quality of data has an important impact on the trained denoiser's performance, so we have invested considerable time in carefully preparing a reasonably large dataset which covers a wide range of object scale, shading, and distributed effects seen in most modern renderings. Figure 6 shows selected ground truth images from our dataset.

We imported assets at different scales collected from various public resource archives (full credits are included in our Electronic Supplementary Material (ESM)) into Autodesk Maya to create our scenes. By applying different lighting (both analytical lights and image-based lighting), materials (mostly physically based materials), and cameras with different angles, aperture size, and focal lengths, the whole dataset was rendered with Pixar's non-commercial RenderMan RIS renderer. We authored over 50 different scenes

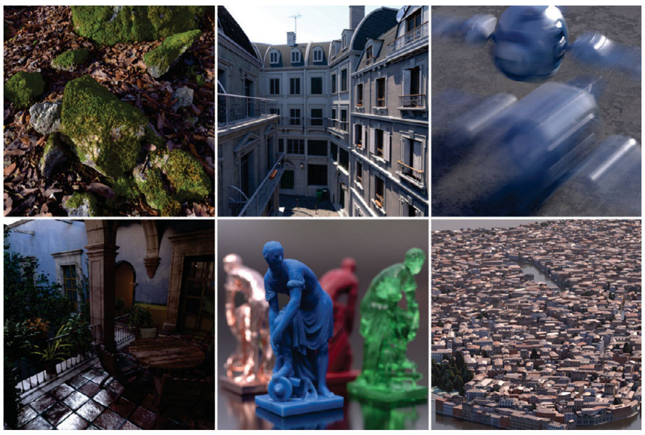
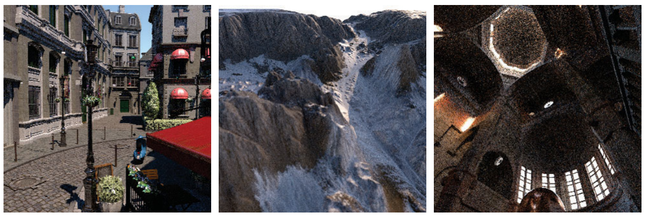


Fig. 6 Selected ground truth images from our dataset.

covering a wide range of genres and scales, including natural and procedural objects, interiors, sci-fi, automobiles, street scenes, cityscapes, etc. As our resources were very limited, and this dataset was to be used to train our own network and also KPCN for comparison, we were cautious and attempted to produce a dataset diverse enough for supervised learning of neural networks with good generalization.

We rendered our ground truth images at 1024×1024 resolution with 2048 samples per pixel (spp) to achieve a perceptually noise-free quality, with a few exceptions rendered at 512 and 8192 spp. The noisy counterparts were rendered at 8–32 spp according to the level of perceived noise which often depended on the material response and lighting conditions rather than a particular sampling rate. Figure 7 shows a few noisy training samples from our dataset.

There were in total 256 multichannel high resolution images in the final dataset. For training purposes, we further extracted small patches from them. We relied on the color variance channel and blue noise sampling for patch selection so as to collect noisy data rather than unhelpful smooth data which might be collected by sampling uniformly.



(a) 8 spp (b) 16 spp (c) 32 spp

Fig. 7 Selected noisy images rendered at different sample rates from our dataset.

We extracted 256 unique patches of size 64 × 64 from each image, giving a dataset of 65,536 multi-channel image patches. For training efficiency, we also created network model specific datasets with the unused image channels removed or simple statistics precomputed for overall improved I/O performance.

4.2 Model implementation and training

We implemented our *16-ResBlock* (2 convolutional layers of 128 filters of size 3×3 per block) ResNet (see Fig. 2) using the Python API of the Cognitive Toolkit (CNTK) [48, 49]. Training data were stored in OpenEXR high dynamic range image files, and we used the OpenImageIO [50] library to read and serve the image patches as NumPy [51] arrays to our CNTK network. Image patches were pre-shuffled to ensure a good mix of patches from different scenes in each mini-batch. In addition, 660 image patches were randomly selected, and reserved for in-training validation use.

All weight parameters were initialized with the He initializer [32] which is designed to be used together with parametric ReLU activation units. As the batch normalization [12] units have built-in bias parameters, there was no need to include bias in the convolutional layers. Our network was optimized using the ADAM [52] optimizer available in CNTK with a momentum value set at 0.9, and gradient clip threshold set at 1.0. $L1$-loss was used as the loss function for our final network (choice of loss function is discussed in the next section). Training used a mini-batch size of 10, and ran for 10^6 iterations. The corresponding learning rate schedule was as follows:

- 0.01 for the first 1000 iterations,
- 0.001 for the second 1000 iterations,
- 10^{-4} for the rest.

Our network has a total of 13,847,296 trainable parameters, and the training took less than 36 hours to complete on an nVIDIA GeForce GTX 1080 Ti GPU (98% GPU load). Figure 8 shows the $L1$-loss and errors during the training session. The training loss converged quickly, with stable progression. The validation error also decreased steadily without any sign of over-fitting.

4.3 Loss functions

In many recent CNN-based denoising applications [7, 53, 54], the $L1$-loss function has been found to be a consistently good performer. It is inexpensive

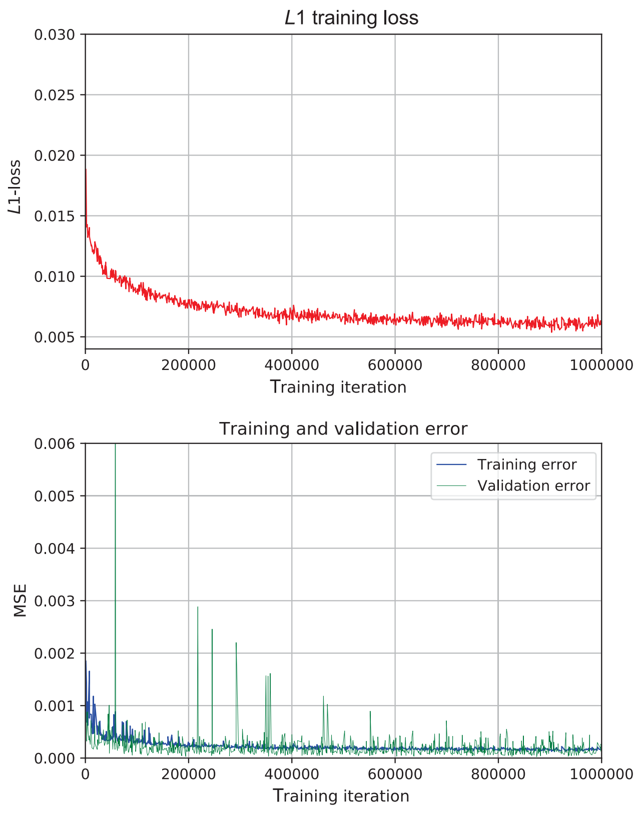


Fig. 8 Training loss (top), and training and validation errors (bottom) for our *16-ResBlock* network during a training session with 10^6 iterations.

to compute, and often surpasses metric-specific loss functions such as MSE loss. We evaluated a combination of potentially useful loss functions for our denoising task before training the final network. We evaluated combinations of $L1$ and $L2$ loss functions, and also the VGG-network [38] based perceptual loss [55]. The VGG-perceptual loss is expensive as it requires inference through the VGG network, and the average sum of multiple feature maps. Although this perceptual loss function is known to improve sharpness in some inverse problems when coupled with MSE loss, our experiments showed that $L1$-loss remains the best loss function, especially if cost effectiveness or fast convergence is a concern.

4.4 Capacity of our network

One common phenomenon related to deep neural networks is called *diminishing feature reuse* [56]: some parts of a deep network end up not learning anything, and it can be understood due to over-capacity. ResNet is especially prone to this issue because the gradient information can basically flow freely to any block during training because of the shortcut connections. Huang et al. [57] propose use of a stochastic training strategy which randomly shuts down different layers to form a virtually shallower network during training. Zagoruyko and Komodakis [58] propose use of a widened (more filters per layer) version of ResNet with reduced depth. Some wide residual networks have been reported to perform better than deep ones for some applications [58]. As a result, we built a shallower but wider, *8-ResBlock* version of our network (with 256 filters of size 3×3 in each convolutional layer). We trained this wide version using the same training setup as for the *16-ResBlock*, and Fig. 9 shows the corresponding progression of training loss and errors.

This wide version has a total of 19,592,704 trainable parameters, and the 10^6 iterations for training took approximately 52 hours to complete on an nVIDIA GeForce GTX 1080 Ti GPU. This wide ResNet has considerably more trainable parameters but only achieved similar training loss and error as our proposed *16-ResBlock* ResNet, while its denoising performance was consistently inferior to the *16-ResBlock* version in our tests. This suggests that

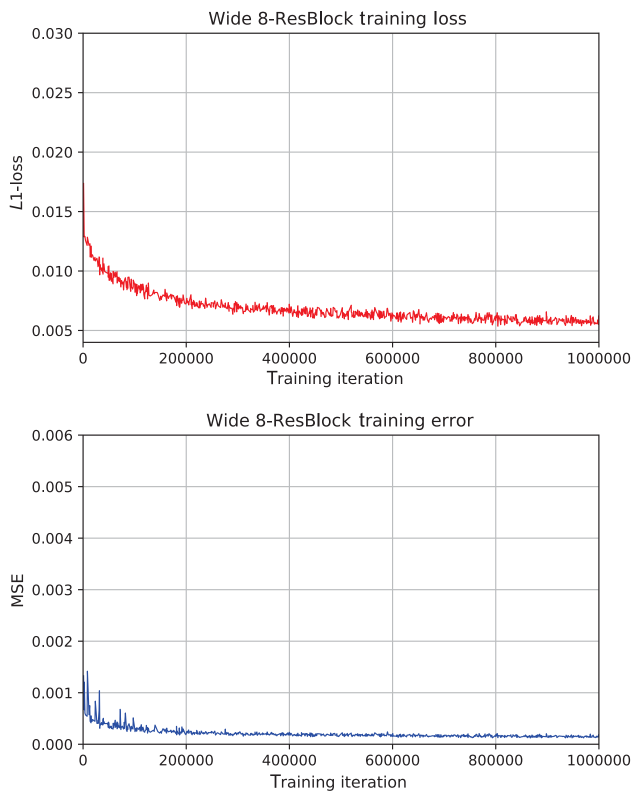


Fig. 9 Training loss (top) and training error (bottom) of an alternative wide *8-ResBlock* network during a training session with 10^6 iterations.

the proposed *16-ResBlock* ResNet is making proper use of its capacity, while greater depth allows more sophisticated mapping at least empirically.

5 Results and evaluation

5.1 Outline

We now compare the results of our *filter-free direct denoising* neural network with current state-of-the-art denoisers based on joint filtering, and learning-based approaches. We compare with the denoisers NFOR [5] and KPCN [7] using low sample rate noisy images rendered with scenes curated by Ref. [11]; they have diverse lighting, materials, and level of detail.

For conciseness, we report only the SSIM [59] and relative MSE [44] in the figures. A fuller report with additional quality metrics is available in the ESM, and we recommend close inspection of the full-resolution images available in it.

The learning-based denoiser KPCN [7] requires some additional preparation. We followed the details in their paper and source code to re-implement and train their network model using CNTK [49]. We pre-processed image patches from our dataset to produce the 27-channel data inputs required by each diffuse and specular KPCN filter pipeline in order to minimize data processing during training. We trained the networks with an nVIDIA GeForce GTX 1080 Ti; the training time for each filter pipeline took approximately 14 hours over 750k iterations each, while the 250k iterations of joint fine-tuning took another 10 hours. For the NFOR denoiser, we used the publicly available implementation provided by the Tungsten software package. We now discuss the results.

5.2 Kitchen (close-up) scene comparison (16 spp, dynamic range 0.0–4.0)

Figure 10 shows the denoising results for a scene populated with objects with different glossiness. NFOR (see Fig. 10(b)) denoises the kitchen countertop reasonably but leaves some splotches. There are also very subtle splotches left on the stainless steel wall panel but the overall denoising result is clean. KPCN (see Fig. 10(c)) removes most noise on the countertop, but with noticeable smear marks, and the texture of the countertop is not well

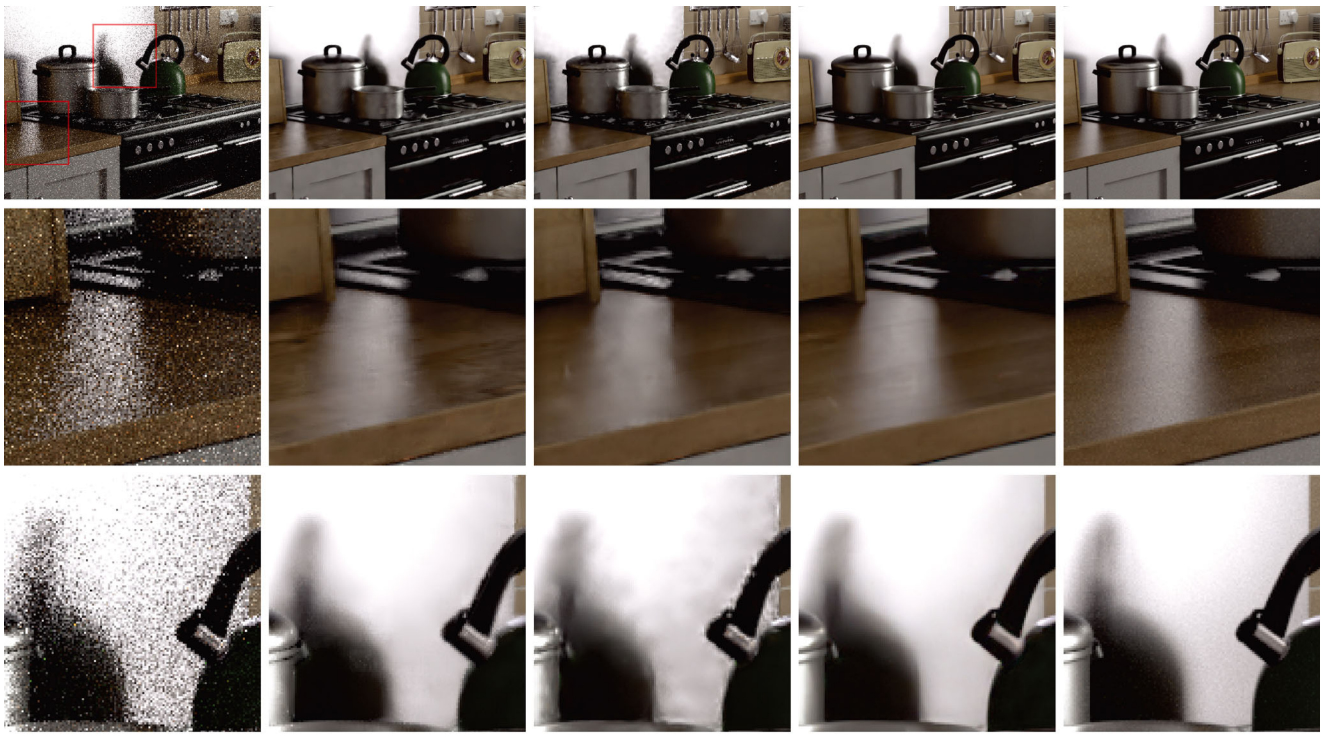

(a) Input, SSIM=0.7109, RelMSE=161.62 (b) NFOR, SSIM=0.9645, RelMSE=9.784 (c) KPCN, SSIM=0.9693, RelMSE=7.067 (d) Ours, SSIM=0.9784, RelMSE=4.184 (e) Reference (2048 spp), dynamic range: 0.0 – 4.0

Fig. 10 Kitchen (close-up). Input (16 spp) and reference (2048 spp) images rendered by Tungsten. Quality metrics refer to the top row of images. RelMSE (10^{-3}).

recovered. The denoising result for the stainless steel panel is unsatisfactory, and the silhouette of the kettle has room for improvement. Our method (see Fig. 10(d)) denoises both countertop and stainless steel panel with good results. The silhouette of the kettle is very sharp and the shading on the stainless steel panel closely matches the reference.

5.3 Bedroom scene comparison (16 spp, dynamic range 0.0–16.49)

Figure 11 shows denoising results with a rather low sampling rate input. The high intensity and directional light setup is challenging for a regular path tracer, and the input is seemingly under-sampled. NFOR (see Fig. 11(b)) recovers the corrupted ceiling lamp nicely but it leaves visually distracting splotches on most diffuse surfaces, and the black details on the decorative plant are also softened. The splotches could be a consequence of the relatively high local sensitivity of a first-order method. KPCN (see Fig. 11(c)) successfully denoises most diffuse surfaces but it has difficulty in recovering the ceiling lamp, with some artifacts on the silhouette. We suspect this could be caused by the diffuse/specular decomposed pipeline or the occasional inability of KPCN to generalize as reported in the original paper (re-training is required for improvement); we return to discuss this in detail later. Our method (see Fig. 11(d)) denoises most diffuse surfaces properly and recovers the ceiling lamp with the best results of any denoiser.

5.4 Car scene comparison (32 spp, dynamic range 0.0–3.60)

Figure 12 shows denoising results for a scene with a depth of field effect which requires high sample rate to obtain noise-free results. NFOR (see Fig. 12(b)) denoises the out-of-focus area but the reconstruction looks somewhat splotchy and the smoothness could be improved. We note that although the floor should not be difficult to denoise, NFOR leaves some visually distracting splotches on the floor and it seems that all image metrics fail to capture these artifacts. KPCN (see Fig. 12(c)) performs unsatisfactorily in this test although it was trained by the same dataset, we suspect this could be related to the choice of providing feature information such as depth in gradient form; this requires the network to learn re-integrating the gradients in order to extract the correct relative difference of depth between distant pixels. Ours

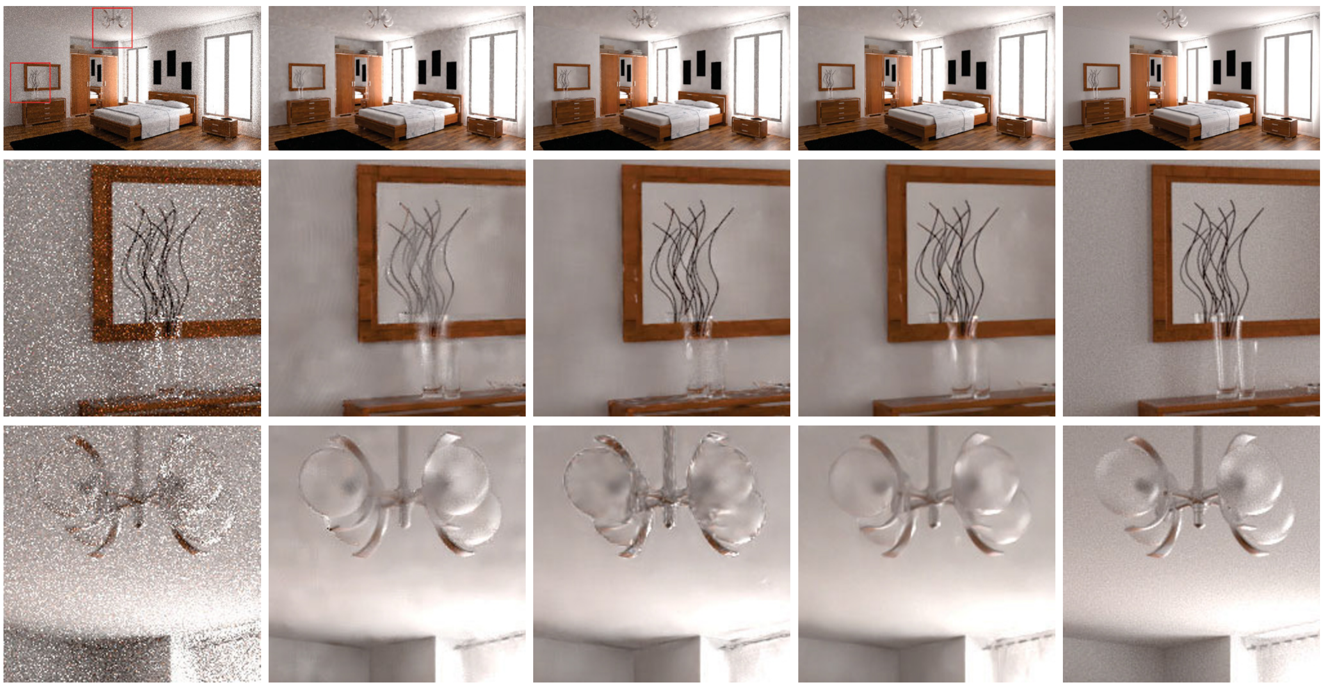
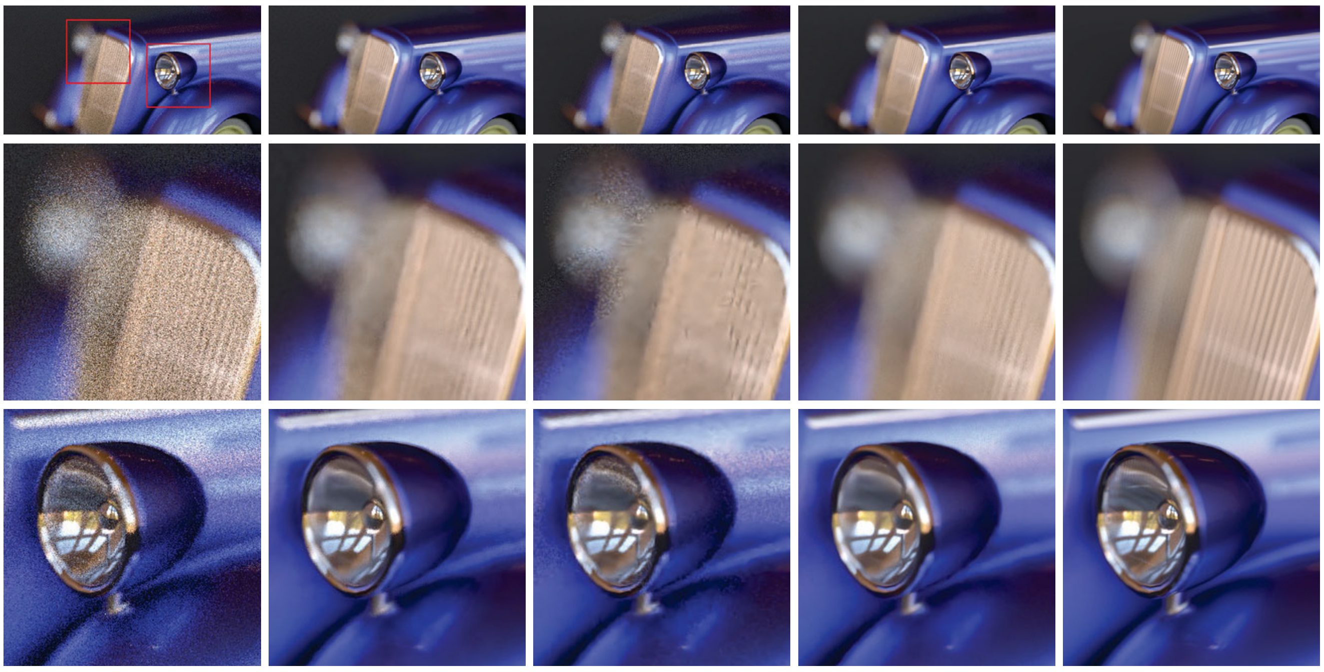

(a) Input, SSIM=0.7838, RelMSE=132.506 (b) NFOR, SSIM=0.9716, RelMSE=6.562 (c) KPCN, SSIM=0.9688, RelMSE=4.761 (d) Ours, SSIM=0.9746, RelMSE=3.142 (e) Reference. Dynamic range: $0.0 - 16.49$

Fig. 11 Bedroom scene. Input (16 spp) and reference (2048 spp) images rendered by Tungsten. Quality metrics refer to the top row images. RelMSE (10^{-3}).



(a) Input, SSIM=0.8991, RelMSE=14.544 (b) NFOR, SSIM=0.9948, RelMSE=0.704 (c) KPCN, SSIM=0.9907, RelMSE=1.094 (d) Ours, SSIM=0.9950, RelMSE=0.552 (e) Reference. Dynamic range: $0.0-3.60$

Fig. 12 Car scene. Input (32 spp) and reference (8192 spp) images rendered by Tungsten. Quality metrics refer to the top row images. RelMSE (10^{-3}).

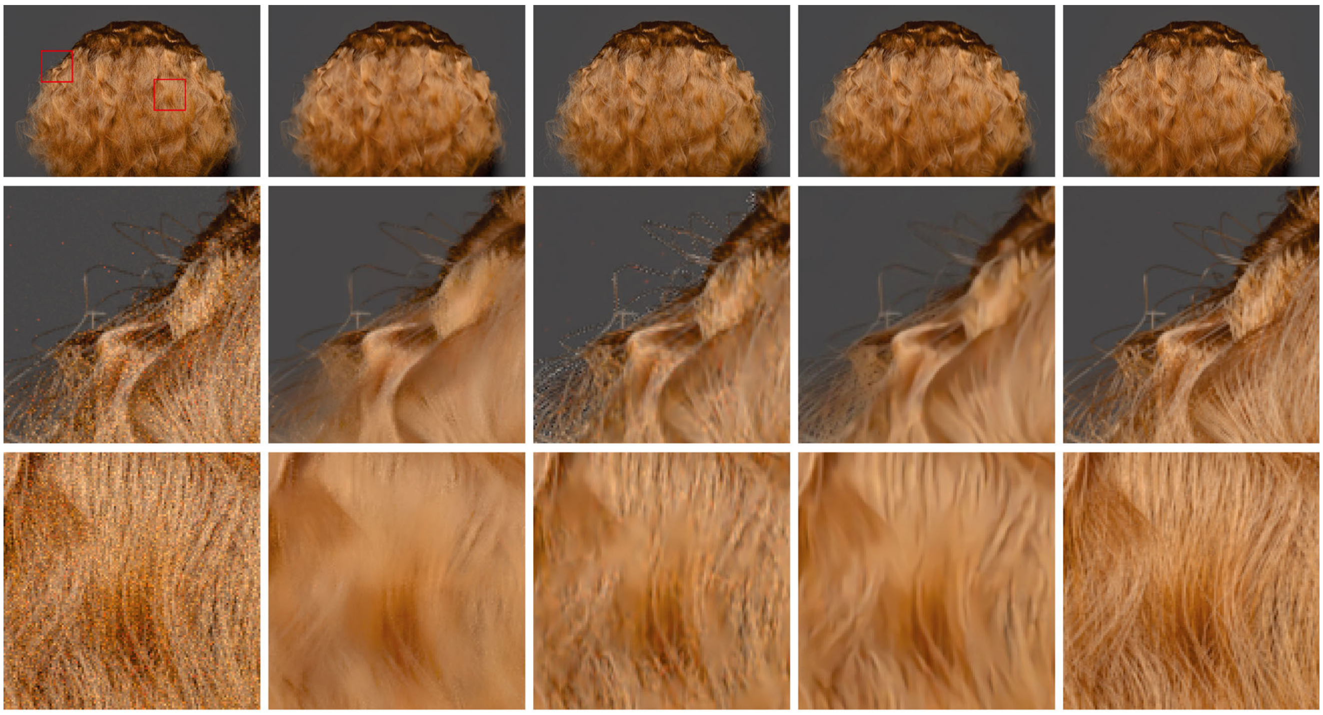
(Fig. 12(d)) shows that the network has successfully learned from the dataset how to map from noisy depth of field pixels to their noise-free counterparts, and its result is rated highest quantitatively.

5.5 Hair scene comparison (32 spp, dynamic range 0.0–1.06)

Figure 13 shows denoising results for a fairly challenging scene. Hair and fur are challenging objects to sample and render, as the naturally high frequency details exhibit complicated noise patterns when under-sampled spatially or in terms of shading. NFOR (see Fig. 13(b)) seems to aggressively smooth everything. KPCN (see Fig. 13(c)) attempts to maintain fine features but the residual shading noise gives an impression of incomplete filtering. Our method (see Fig. 13(d)) removes the shading noise more successfully while maintaining a reasonable amount of fine detail, but the result is not particularly impressive even it is judged best by the quality metrics. We have only a few hair-related images in our training dataset, and it seems denoising such fine objects might require more specialized training. Such images remain a challenge to most denoisers.

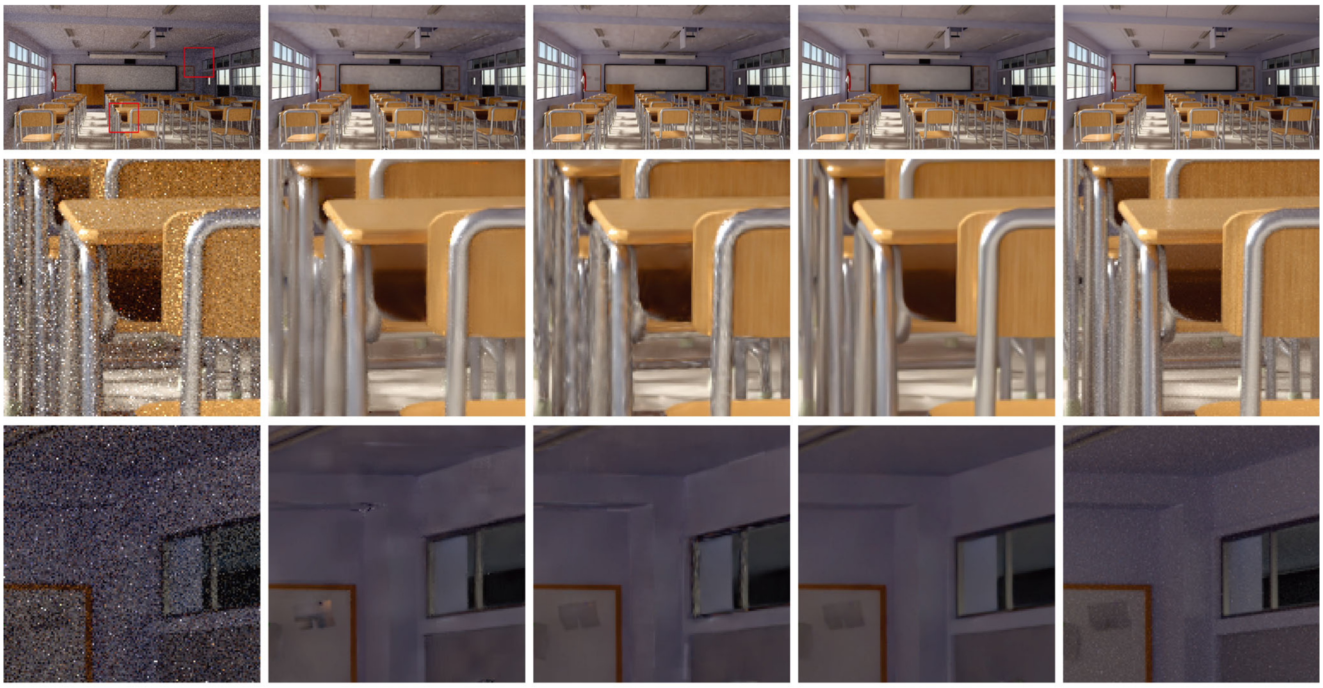
5.6 Classroom scene comparison (32 spp, dynamic range 0.0–36.34)

Figure 14 shows denoising results of another challenging scene. The lighting conditions are similar to those for the bedroom scene (see Fig. 11) but it has an even higher dynamic range, more details in the dark area, and glossy materials on thin objects (the chair frame). NFOR (see Fig. 14(b)) follows its own pattern of leaving distracting splotches on the diffuse walls and ceilings. For this scene, it fails to handle dark areas corrupted by outliers and leaves unpleasant artifacts in those areas. NFOR handles the noise on the chair frames properly but there are still subtle splotches on them. KPCN (see Fig. 14(c)) denoises the diffuse areas properly but in the dark areas which are corrupted high intensity outliers, it leaves some unexpected edges. As for the ceiling lamp case in the bedroom scene (see Fig. 11), KCPN has difficulty in maintaining a smooth boundary between glossy and diffuse areas, and leaves an impression of aliasing. Our method (see Fig. 14(d)) seems to denoise consistently well in both bright and dark areas, and the glossy shading on the chair frames is smoother than the reference which still exhibits residual noise. Our denoiser works remarkably well in this scene.



(a) Input, SSIM=0.8878, RelMSE=13.936 (b) NFOR, SSIM=0.9425, RelMSE=4.334 (c) KPCN, SSIM=0.9532, RelMSE=4.187 (d) Ours, SSIM=0.9561, RelMSE=3.266 (e) Reference. Dynamic range: 0.0 − 1.06

Fig. 13 Hair scene. Input (32 spp) and reference (2048 spp) images rendered by Tungsten. Quality metrics refer to the top row images. RelMSE (10^{-3}).



(a) Input, SSIM=0.9913, RelMSE=126.433 (b) NFOR, SSIM=0.9985, RelMSE=9.271 (c) KPCN, SSIM=0.9987, RelMSE=5.578 (d) Ours, SSIM=0.9993, RelMSE=3.288 (e) Reference. Dynamic range: 0.0 − 36.34

Fig. 14 Classroom scene. Input (32 spp) and reference (2048 spp) images rendered by Tungsten. Quality metrics refer to the top row images. RelMSE (10^{-3}).

5.7 Overall evaluation

In all the tests shown, our direct denoising network consistently outperformed the other two state-of-the-art solutions quantitatively. Additional test results can be found in the ESM; Tables 1 and 2 respectively summarize the SSIM and relative MSE results for the complete set of tests. NFOR performs better in a few scenes when judged by the pixel-space metric MSE, but in those cases, their results have obvious splotches on diffuse surfaces, and such artifacts cannot be captured by most pixel-space metrics. We have included all results in full resolution in the ESM, which provides for closer visual inspection.

Table 1 SSIM results

Scene name	NFOR	KPCN	Ours
Hair	0.942510	0.953195	**0.956061**
Kitchen (close-up)	0.964515	0.969311	**0.978701**
Staircase (dof)	0.973732	0.939727	**0.982566**
Bedroom	0.971588	0.968812	**0.974636**
Classroom	0.998524	0.998746	**0.999253**
Car (dof)	0.994781	0.990718	**0.994969**
Car	0.996666	0.996063	**0.997087**
Veach	0.935345	0.921266	**0.938137**
Staircase 2	0.977141	0.976496	**0.978526**
Spaceship	0.998996	0.998673	**0.999256**
Kitchen	0.995988	0.995664	**0.996677**
House	0.980789	0.983675	**0.987976**
Bathroom	**0.995053**	0.992926	0.994819
Staircase	0.993212	**0.997047**	0.996947

Table 2 Relative MSE results ($\times 10^{-3}$)

Scene name	NFOR	KPCN	Ours
Hair	4.334	4.187	**3.266**
Kitchen (close-up)	9.784	7.067	**4.184**
Staircase (dof)	5.947	3.184	**1.394**
Bedroom	6.562	4.761	**3.142**
Classroom	9.271	5.578	**3.288**
Car (dof)	0.704	1.094	**0.552**
Car	1.246	1.041	**0.828**
Veach	17.380	19.510	**12.270**
Staircase 2	3.614	3.843	**2.978**
Spaceship	1.630	1.171	**0.767**
Kitchen	4.139	2.807	**1.871**
House	1.347	0.997	**0.775**
Bathroom	22.210	16.620	**8.316**
Staircase	4.041	1.315	**1.247**

6 Discussion

6.1 Key findings

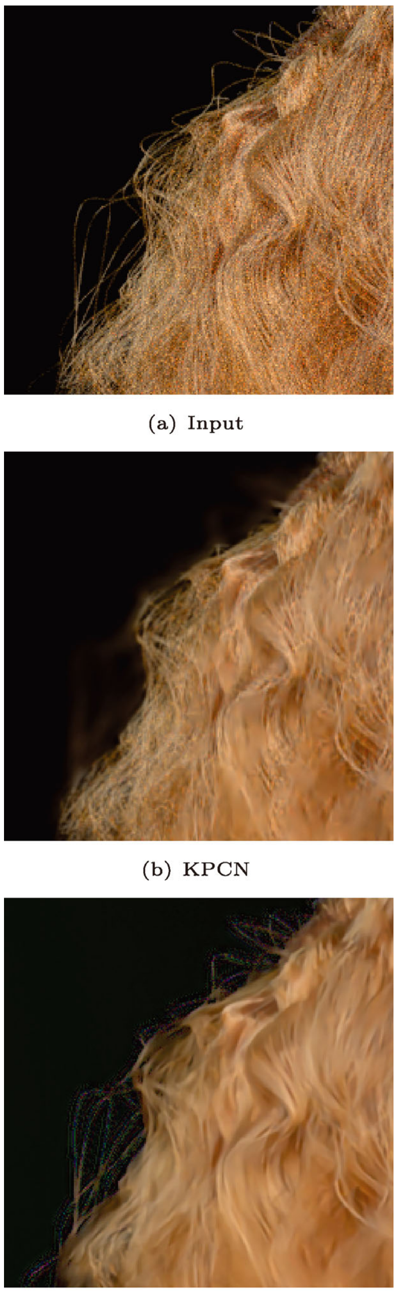
The denoising results in the last section proves the competitiveness of our *filter-free direct denoising* network as a practical solution for denoising Monte Carlo rendering output. We have to emphasize that the quality of most supervised learning methods relies heavily on the quality of the training data set, and we are pleased to see that our dataset helps to achieve very competitive denoising results. The unique architecture of ResNet [9] enables sophisticated mapping possibilities through the identity mapping. The freedom to allow reuse of upstream features repeatedly is very similar to many multi-scale algorithms, e.g., multigrid [60] as mentioned in the original paper [9]. This is also the main reason why we choose ResNet as our network solution. In addition, the choice of using a small set of primary auxiliary features and letting the network explore the solutions itself without overloading seems to be a good strategy.

6.2 Limitations

A fundamental limitation of all supervised learning methods is connected to the coverage of training set. Our approach relies on the samples in the training set to establish its non-linear mapping. For any cases that have not been included in the training set, there is a potential that our network may fail to deliver expected results. Our training set has no samples of fine objects such as hair on a blank background (zeros in all inputs). We used an untrained case to test both our network and KPCN (both trained with our dataset). Figure 15 shows the denoising artifacts arising in both methods. KPCN (see Fig. 15(b)) blurs all fine hairs on an empty background, while ours (see Fig. 15(c)) shows sparsely colored pixels. A common solution is to retrain the networks with additional desired training samples.

6.3 Joint filtering versus direct denoising

Carefully handcrafted joint filters, such as the state-of-the-art denoiser NFOR [5] can handle many denoising cases with outstanding results, but the underlying rigid regression-based formulation cannot adapt well to extremely noisy inputs. In order to improve these handcrafted models, researchers need to explore further for potential causes of noise or hidden correlations. In contrast, a direct denoising method based on a neural network relies on learning from training samples to establish powerful non-linear mappings. There is no difficulty in obtaining many noise-free images for training in most production studios, and a deep learning method seems to be a more natural choice. If a direct denoising network


(a) Input

(b) KPCN

(c) Ours

Fig. 15 Denoising images not covered by our training set. Our training set has no example pairs for fine objects with a background of blank pixels (zeros in color and auxiliary features). Both KPCN and our method show artifacts, which highlight the underlying differences between these methods.

faces a new challenge, it can be improved quickly by learning from additional examples in a matter of hours. KPCN [7] can be classified as a hybrid method, and its current formulation seems to have inherited the disadvantages of both joint filters and direct denoising, i.e., the solution space is limited to noisy input colors and the dependency on training set coverage. We observe that the choice of separating diffuse and specular components in KPCN as in Ref. [61] might not be a good decision. The original idea of such decomposition is to facilitate an analytical approach to handle specular paths under a light transport setting. The specular-only solution space for joint filtering can sometimes be very sparse, making good filtering even more difficult.

6.4 Runtime performance

To process a 1024×1024 image, a non-optimized implementation of our network running on an nVIDIA GTX 1080 Ti takes approximately 18 s to perform denoising. Our implementation of KPCN is not optimal, and we believe a competent implementation should take a similar or lower time to our method. The open source implementation of NFOR spends 89 s to denoise the same image on an Intel XEON E5-2683 V3 CPU.

7 Conclusions and future work

We have presented a filter-free direct denoising network solution for processing noisy Monte Carlo rendering output. Our ResNet [9] based network is able to establish a sophisticated mapping through supervised learning. Our network generalizes very well and is able to deliver high-quality denoising results from noisy images rendered by a different renderer.

Temporal stability is the first subject we want to explore next with our method, and also the possibility of including denoising level control, which is often desirable in a production environment.

Acknowledgements

This work was supported by the Research Grants Council of the Hong Kong Special Administrative Region, under RGC General Research Fund (Project No. CUHK14217516).

Electronic Supplementary Material Supplementary material is available in the online version of this article at `https://doi.org/10.1007/s41095-019-0142-3`.

References

[1] Jakob, W.; Marschner, S. Manifold exploration: A Markov Chain Monte Carlo technique for rendering scenes with difficult specular transport. *ACM Transactions on Graphics* Vol. 31, No. 4 Article No. 58, 2012.

[2] Hachisuka, T.; Pantaleoni, J.; Jensen, H. W. A path space extension for robust light transport simulation. *ACM Transactions on Graphics* Vol. 31, No. 6, Article No. 191, 2012.

[3] Georgiev, I.; Křivánek, J.; Davidovič, T.; Slusallek, P. Light transport simulation with vertex connection and merging. *ACM Transactions on Graphics* Vol. 31, No. 6, Article No. 192, 2012.

[4] 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.

[5] Bitterli, B.; Rousselle, F.; Moon, B.; Iglesias-Guitián, 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.

[6] Rousselle, F.; Manzi, M.; Zwicker, M. Robust denoising using feature and color information. *Computer Graphics Forum* Vol. 32, No. 7, 121–130, 2013.

[7] 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, 1–14, 2017.

[8] 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.

[9] He, K. M.; Zhang, X. Y.; Ren, S. Q.; Sun, J. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770–778, 2016.

[10] He, K. M.; Zhang, X. Y.; Ren, S. Q.; Sun, J. Identity mappings in deep residual networks. In: *Computer Vision – ECCV 2016. Lecture Notes in Computer Science, Vol 9908.* Leibe, B.; Matas, J.; Sebe, N.; Welling, M. Eds. Springer Cham, 630–645, 2016.

[11] Bitterli, B. Rendering resources 2016. Available at https://benedikt-bitterli.me/resources/.

[12] Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint* arXiv:1502.03167, 2015.

[13] Zwicker, M.; Jarosz, W.; Lehtinen, J.; Moon, B.; Ramamoorthi, R.; Rousselle, F.; Sen, P.; Soler, C.; Yoon, S.-E. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Computer Graphics Forum* Vol. 34, No. 2, 667–681, 2015.

[14] Goodfellow, I.; Bengio, Y.; Courville, A.; Bengio, Y. *Deep Learning, Vol. 1.* MIT Press, 2016.

[15] LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* Vol. 521, No. 7553, 436–444, 2015.

[16] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks* Vol. 61, 85–117, 2015.

[17] McCool, M. D. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Transactions on Graphics* Vol. 18, No. 2, 171–194, 1999.

[18] Perona, P.; Malik, J. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 12, No. 7, 629–639, 1990.

[19] Dammertz, H.; Sewtz, D.; Hanika, J.; Lensch, H. P. A. Edge-avoiding À-Trous wavelet transform for fast global illumination filtering. In: Proceedings of the Conference on High Performance Graphics, 67–75, 2010.

[20] Fattal, R. Edge-avoiding wavelets and their applications. *ACM Transactions on Graphics* Vol. 28, No. 3, Article No. 22, 2009.

[21] 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.

[22] Aurich, V.; Weule, J. Non-linear Gaussian filters performing edge preserving diffusion. In: *Mustererkennung 1995. Informatik aktuell.* Sagerer, G.; Posch, S.; Kummert, F. Eds Springer Berlin Heidelberg, 538–545 1995.

[23] Tomasi, C.; Manduchi, R. Bilateral filtering for gray and color images. In: Proceedings of the IEEE International Conference on Computer Vision, 839–846, 1998.

[24] Eisemann, E.; Durand, F. Flash photography enhancement via intrinsic relighting. *ACM Transactions on Graphics* Vol. 23, No. 3, 673–678, 2004.

[25] Li, T. M.; Wu, Y. T.; Chuang, Y. Y. SURE-based optimization for adaptive sampling and reconstruction. *ACM Transactions on Graphics* Vol. 31, No. 6, Article No. 194, 2012.

[26] Stein, C. M. Estimation of the mean of a multivariate normal distribution. *The Annals of Statistics* Vol. 9, No. 6, 1135–1151, 1981.

[27] Van de Ville, D.; Kocher, M. SURE-based non-local means. *IEEE Signal Processing Letters* Vol. 16, No. 11, 973–976, 2009.

[28] Buades, A.; Coll, B.; Morel, J. M. Nonlocal image and movie denoising. *International Journal of Computer Vision* Vol. 76, No. 2, 123–139, 2008.

[29] Moon, B.; Iglesias-Guitian, J. A.; Yoon, S. E.; Mitchell, K. Adaptive rendering with linear predictions. *ACM Transactions on Graphics* Vol. 34, No. 4, Article No. 121, 2015.

[30] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. ImageNet classification with deep convolutional neural networks. In: Proceedings of the Advances in Neural Information Processing Systems 25, 1097–1105, 2012.

[31] Szegedy, C.; Liu, W.; Jia, Y. Q.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1–9, 2015.

[32] He, K. M.; Zhang, X. Y.; Ren, S. Q.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In: Proceedings of the IEEE International Conference on Computer Vision, 1026–1034, 2015.

[33] Xie, J.; Xu, L.; Chen, E. Image denoising and inpainting with deep neural networks. In: Proceedings of the Advances in Neural Information Processing Systems 25, 341–349, 2012.

[34] Iizuka, S.; Simo-Serra, E.; Ishikawa, H. Globally and locally consistent image completion. *ACM Transactions on Graphics* Vol. 36, No. 4, Article No. 107, 2017.

[35] Xu, L.; Ren, J. S.; Liu, C.; Jia, J. Deep convolutional neural network for image deconvolution. In: Proceedings of the Advances in Neural Information Processing Systems 27, 1790–1798, 2014.

[36] Shi, W. Z.; Caballero, J.; Huszar, F.; Totz, J.; Aitken, A. P.; Bishop, R.; Rueckert, D.; Wang, Z. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1874–1883, 2016.

[37] Ledig, C.; Theis, L.; Huszar, F.; Caballero, J.; Cunningham, A.; Acosta, A.; Aitken, A.; Tejani, A.; Totz, J.; Wang, Z.; Shi, W. Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 105–114, 2017.

[38] Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint* arXiv:1409.1556, 2014.

[39] 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.

[40] Nah, S.; Kim, T. H.; Lee, K. M. Deep multi-scale convolutional neural network for dynamic scene deblurring. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 257–265, 2017.

[41] Xu, R. F.; Pattanaik, S. N. A novel Monte Carlo noise reduction operator. *IEEE Computer Graphics and Applications* Vol. 25, No. 2, 31–35, 2005.

[42] Shirley, P.; Timo, A. L.; Cohen, J.; Enderton, E.; Laine, S.; Luebke, D.; McGuire, M. A local image reconstruction algorithm for stochastic rendering. In: Proceedings of the Symposium on Interactive 3D Graphics and Games, 9–14, 2011.

[43] Bauszat, P.; Eisemann, M.; Magnor, M. Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum* Vol. 30, No. 4, 1361–1368, 2011.

[44] Rousselle, F.; Knaus, C.; Zwicker, M. Adaptive sampling and reconstruction using greedy error minimization. *ACM Transactions on Graphics* Vol. 30, No. 6, Article No. 159, 2011.

[45] Lim, B.; Son, S.; Kim, H.; Nah, S.; Lee, K. M. Enhanced deep residual networks for single image super-resolution. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 1132–1140, 2017.

[46] Mao, X.; Shen, C.; Yang, Y.-B. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In: Proceedings of the Advances in Neural Information Processing Systems 29, 2802–2810, 2016.

[47] Zeiler, M. D.; Fergus, R. Visualizing and understanding convolutional networks. In: *Computer Vision – ECCV 2014. Lecture Notes in Computer Science, Vol. 8689.* Fleet, D.; Pajdla, T.; Schiele, B.; Tuytelaars, T. Eds. Springer Cham, 818–833, 2014.

[48] Yu, D.; Eversole, A.; Seltzer, M. L.; Yao, K.; Huang, Z.; Guenter, B.; Kuchaiev, O.; Zhang, Y.; Seide, F.; Wang, H. et al. An introduction to computational networks and the computational network toolkit. Microsoft Technical Report MSR-TR-2014–112. 2014.

[49] CNTK. Microsoft cognitive toolkit. 2018.

[50] Gritz, L. Openimageio software. 2008.

[51] Van der Walt, S.; Colbert, S. C.; Varoquaux, G. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering* Vol. 13, No. 2, 22–30, 2011.

[52] Kingma, D. P.; Ba, J. Adam: A method for stochastic optimization. *arXiv preprint* arXiv:1412.6980, 2014.

[53] Zhao, H.; Gallo, O.; Frosio, I.; Kautz, J. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging* Vol. 3, No. 1, 47–57, 2017.

[54] 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.

[55] Johnson, J.; Alahi, A.; Li, F. F. Perceptual losses for real-time style transfer and super-resolution. In: *Computer Vision – ECCV 2016. Lecture Notes in Computer Science, Vol. 9906.* Leibe, B.; Matas, J.; Sebe, N.; Welling, M. Eds. Springer Cham, 694–711, 2016.

[56] Srivastava, R. K.; Greff, K.; Schmidhuber J. Highway networks. *arXiv preprint* arXiv:1505.00387, 2015.

[57] Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; Weinberger, K. Q. Deep networks with stochastic depth. In: *Computer Vision – ECCV 2016. Lecture Notes in Computer Science, Vol 9908*. Leibe, B.; Matas, J.; Sebe, N.; Welling, M. Eds. Springer Cham, 646–661, 2016.

[58] Zagoruyko, S.; Komodakis, N. Wide residual networks. *arXiv preprint* arXiv:1605.07146, 2016.

[59] Wang, Z.; Bovik, A. C.; Sheikh, H. R.; Simoncelli, E. P. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing* Vol. 13, No. 4, 600–612, 2004.

[60] Briggs, W. L.; McCormick, S. F. *A Multigrid Tutorial, Vol. 72.* Siam, 2000.

[61] 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.

Kin-Ming Wong is an award winning visual effects professional and the owner of artixels, a boutique visual effects software developer. He received his Ph.D. degree from the Chinese University of Hong Kong under the supervision of Prof. Tien-Tsin Wong. He works on photo-realistic rendering problems with a strong interest in high-performance sampling and filtering techniques. He is also a computational artist with works exhibited in SIGGRAPH and GRAPHITE (the predecessor of SIGGRAPH Asia).

Tien-Tsin Wong received his B.Sc., M.Phil., and Ph.D. degrees in computer science from the Chinese University of Hong Kong in 1992, 1994, and 1998 respectively. He is currently a professor in the Department of Computer Science and Engineering, the Chinese University of Hong Kong. His main research interests include computer graphics, computer vision, computational perception, computational manga, GPU techniques, and image-based rendering. He received the IEEE Transactions on Multimedia Prize Paper Award in 2005 and the Young Researcher Award in 2004.