Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder

CHAKRAVARTY R. ALLA CHAITANYA, NVIDIA, University of Montreal and McGill University
ANTON S. KAPLANYAN, NVIDIA
CHRISTOPH SCHIED, NVIDIA and Karlsruhe Institute of Technology
MARCO SALVI, NVIDIA
AARON LEFOHN, NVIDIA
DEREK NOWROUZEZAHRAI, McGill University
TIMO AILA, NVIDIA

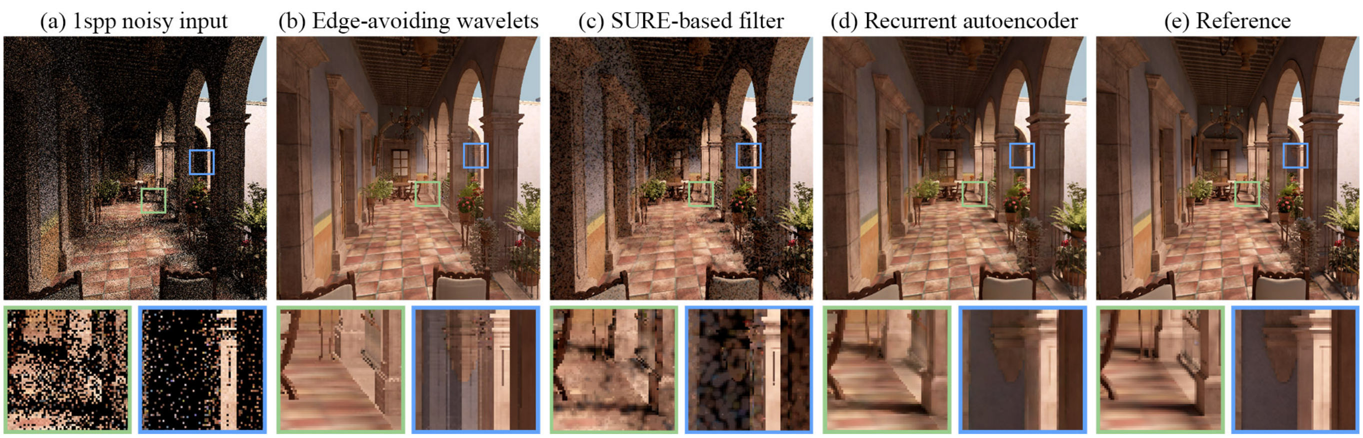


Fig. 1. Left to right: (a) noisy image generated using path-traced global illumination with one indirect inter-reflection and 1 sample/pixel; (b) edge-avoiding wavelet filter [Dammertz et al. 2010] (10.3ms at 720p, SSIM: 0.7737); (c) SURE-based filter [Li et al. 2012] (74.2ms, SSIM: 0.5960); (d) our recurrent denoising autoencoder (54.9ms, SSIM: 0.8438); (e) reference path-traced image with 4096 samples/pixel.

We describe a machine learning technique for reconstructing image sequences rendered using Monte Carlo methods. Our primary focus is on reconstruction of global illumination with extremely low sampling budgets at interactive rates. Motivated by recent advances in image restoration with deep convolutional networks, we propose a variant of these networks better suited to the class of noise present in Monte Carlo rendering. We allow for much larger pixel neighborhoods to be taken into account, while also improving execution speed by an order of magnitude. Our primary contribution is the addition of recurrent connections to the network in order to drastically improve temporal stability for sequences of sparsely sampled input images. Our method also has the desirable property of automatically modeling relationships based on auxiliary per-pixel input channels, such as depth and normals. We show significantly higher quality results compared to existing methods that run at comparable speeds, and furthermore argue a clear path for making our method run at realtime rates in the near future.

CCS Concepts: • **Computing methodologies → Ray tracing**; *Neural networks*; *Image processing*;

Additional Key Words and Phrases: Monte Carlo denoising, image reconstruction, interactive global illumination, machine learning

© 2017 ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *ACM Transactions on Graphics*, https://doi.org/http://dx.doi.org/10.1145/3072959.3073601.

ACM Reference format:
Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article 98 (July 2017), 12 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073601

1 INTRODUCTION

Ray and path tracing have recently emerged as the rendering algorithms of choice for visual effects [Keller et al. 2015]. This has encouraged the development of filtering and reconstruction techniques to reduce the noise inherent in Monte Carlo renderings [Zwicker et al. 2015], but the focus on film-quality results provides hundreds to thousands of samples per pixel prior to filtering.

Meanwhile, games have also recently migrated towards physically-based shading from more empirical models [Hill et al. 2015], but much of the potential increase in realism from this transition hinges on the possibility of sampling light transport paths more flexibly than rasterization allows. Unfortunately, even the fastest ray tracers can only trace a few rays per pixel at 1080p and 30Hz. While this number doubles every few years, the trend is (at least partially) countered by the move towards higher resolution displays and higher refresh rates. It therefore seems likely that a realistic sampling budget for games and other realtime applications will remain on the order of one (short) path per pixel for the foreseeable future. We present a new general reconstruction technique that significantly improves the state-of-the-art in this regime.

Reconstructing global illumination from a single path per pixel presents considerable challenges. Monte Carlo integration of indirect illumination leads to very noisy images at low sampling rates, so that much of the energy is concentrated in a small subset of paths or pixels (e.g. Figure 1, left). We therefore frame the problem as a *reconstruction* of the final image (rather than denoising) from these sparse samples since, at prohibitively low sampling rates, many image areas have almost only noise to begin with. This problem is further compounded when trying to obtain a temporally stable result in animation.

Motivated by recent work in single-image restoration using deep convolutional networks, we propose significant modifications to the structure of these networks in order to address the particular challenges of reconstructing rendered image sequences with extreme Monte Carlo noise. Our solution has the following novel aspects:

- recurrent connections in a deep autoencoder structure lead to increased temporal stability, and
- end-to-end training allows our network to automatically learn how to best utilize auxiliary pixel features, such as depth and normals, with no guidance from the user.

These advances allow us to interactively generate plausible image sequences with global illumination at extremely low sampling budgets, consistently outperforming the quality of the state-of-the-art in this regime.

2 RELATED WORK

A large body of work exists for image denoising and the reconstruction of images from a sparse set of samples. Our primary focus is on the denoising of images rendered using Monte Carlo methods, which has been an important research field for over two decades [Lee and Redner 1990; McCool 1999]. We first review the most relevant work in computer graphics literature and then draw connections to the machine learning-based image and video restoration methods. Recent surveys provide further information [Schmidhuber 2015; Schwenk 2013; Zwicker et al. 2015].

Offline Denoising for Monte Carlo Rendering. "Last-mile" image denoising is essential for making physically based rendering methods viable for production [Keller et al. 2015], due to long-tailed image convergence error with stochastic Monte Carlo (MC) methods. To address this, Jensen and Christensen [1995] apply non-linear image-space filters to indirect diffuse illumination. More recently, by looking at the frequency analysis of light transport, Egan et al. [2011a; 2011b; 2009] derive high quality sheared filters for specific effects. These filters are applied over individual ray samples in a 4D domain. Lehtinen et al. [2012] present high quality image reconstruction from a coarse light field with auxiliary information for each sample. Sen and Darabi [2012] estimate parameters for a cross bilateral filter based on mutual information, and track the noise parameters for each sample.

Many recent papers use auxiliary features from the rendering process. Li et al. [2012] use Stein's unbiased risk estimate [1981] to select from a filter bank in the process of minimizing the denoising error. Rousselle et al. [2013] employ additional features from the renderer for denoising. The challenge is how to select the filter parameters, and the influence of each feature. To address this, Kalantari et al. [2015] train the parameters of a non-local means filter using machine learning. Zimmer et al. [2015] further decompose path-space in multiple buffers and apply individual filters to each buffer. Image features have also been used to build local regression models [Bitterli et al. 2016; Moon et al. 2015, 2016].

These approaches efficiently reduce residual noise at a cost of higher execution times measured in seconds or minutes. In an offline setting, slow reconstruction times are often acceptable, representing a small fraction of the total image render time. Our proposed method also uses auxiliary features, but works well with low sample counts, and runs at interactive rates with plausible results. Next, we will review interactive approaches to denoising.

Interactive Denoising for Monte Carlo Rendering. Images produced with ray tracing are challenging to render interactively due to a high amount of noise with low number of rays traced per pixel [Bikker and van Schijndel 2013]. Robison and Shirley [2009] explore noise-reducing gathering for filtering an input image with a single sample per pixel. Schwenk [2013] proposes to filter the incident indirect radiance, as well as providing a good overview of interactive denoising methods.

A common theme for interactive MC denoising is to separate direct and indirect illumination and filter the latter using edge-avoiding filters. Examples include: edge-avoiding À-trous wavelets [Dammertz et al. 2010; Hanika et al. 2011], adaptive manifolds [Gastal and Oliveira 2012], guided image filters [Bauszat et al. 2011], filtering on adaptive manifolds [Bauszat et al. 2015], and in texture space [Munkberg et al. 2016]. These approaches are fast and produce convincing results, but lack the error estimates applied by many offline approaches. Therefore, local detail may be lost. Please refer to Moon et al. [2015] for an example comparison. Moreover, the amount of filtering is a user parameter. Our approach does not need to separate direct and indirect light and does not require user guidance.

Another class of methods are based on the frequency-space analysis of light transport. Axis-aligned filters have been proposed for interactive soft shadows [Mehta et al. 2012], diffuse indirect lighting [Mehta et al. 2013], and multiple distribution effects [Mehta et al. 2014]. These are faster but less accurate than the sheared filters discussed in the previous section. Yan et al. [2015] propose a fast implementation of a quantized 4D sheared filter, which is more accurate, but slower than the axis-aligned versions. Note that these are domain-specific filters tailored for particular light transport effects. Furthermore, the soft shadow filter is applied once (with different parameters) for each individual area light, which makes it hard to scale to larger scenes. In contrast, our approach is independent of the illumination in the scene.

In a concurrent work [Bako et al. 2017], a new denoising method based on machine learning is presented, which targets image denoising with high sample count.

Convolutional Neural Networks (CNN). CNNs have recently been used in a wide range of graphics-related tasks, including image classification [Krizhevsky et al. 2012], object localization [Girshick et al. 2014], colorization [Iizuka et al. 2016; Larsson et al. 2016], inpainting [Pathak et al. 2016], and view interpolation [Dosovitskiy et al. 2015; Flynn et al. 2016], with consistently good results. We refer an interested reader to the recent survey on this topic [Schmidhuber 2015].

CNNs consist of a sequence of layers that apply a set of convolution kernels to input from the preceding layer, followed by a non-linear mapping [LeCun et al. 1998]

$$\mathbf{a}_i^l = \sigma\left(\sum_j \mathbf{a}_j^{l-1} * \mathbf{k}_{ij}^l + \mathbf{b}_i^l\right). \tag{1}$$

Here $*$ denotes convolution, $\mathbf{a}_i^l$ is the scalar 2D image of the ith activation on the lth layer, $\mathbf{k}_{ij}^l$ and $\mathbf{b}_i^l$ are the jth 2D convolution kernel and bias term associated with output activation i at level l, respectively, and σ is typically a rectified linear unit, ReLU, i.e. $max(0, \cdot)$. The input RGB image is usually considered as a layer with three activations.

Our algorithm makes heavy use of *skip connections* that jump over a set of layers. Such shortcuts make the training of deep networks substantially easier [He et al. 2015a; Raiko et al. 2012]. We also use *recurrent connections* that link layers to themselves, so that the network becomes able to retain state between frames in an animation sequence [Schmidhuber 2015].

Image Restoration using Deep Learning. The denoising of images corrupted with Gaussian noise is an active research topic in image processing and neural networks. Currently the best published results come from a deep convolutional architecture that uses hierarchical skip connections [Mao et al. 2016]. The same architecture also yields state-of-the-art results for super-resolution, JPEG deblocking, and text removal. The network is trained using a large number of natural images, and it learns to complete or correct the images to look locally like the training image patches. We build on this approach, but in our application the noise has very different characteristics compared to the additive Gaussian noise. Some samples typically have a very high energy (well outside the dynamic range of an image), while most areas appear black. The input pixel values are therefore not even approximately correct, but we do know that they are correct on the average.

The task of filling in the missing pixel colors is closely related to image inpainting [Pathak et al. 2016] and single-image super-resolution [Ledig et al. 2016]. The key difference is that in our application we control the image generation process and have access to auxiliary information such as depth and normal buffers, in addition to the pixel colors. In a work closely related to ours, Kalantari et al. [2015] use a fully connected network to estimate the weights for the auxiliary parameters in a non-local means filter. In contrast, we do the entire filtering operation using CNNs.

Since our focus is on highly interactive—and ultimately realtime—rendering, temporal stability of the reconstructed frames is very important. In the context of video super-resolution, good results have been demonstrated using *recurrent neural networks* (RNNs) [Huang et al. 2015] and sub-pixel CNNs [Shi et al. 2016]. Pătrăucean et al. [2015] used a recurrent long short term memory block in the bottleneck of the autoencoder to improve temporal stability. While in super-resolution the (potential) temporal flickering is very limited spatially, in our applications it can affect large, variable-size areas in the images, and thus we choose to build on the general solution of hierarchical RNNs.

3 PATH TRACING

Despite the availability of advanced light transport methods, e.g. bidirectional path tracing and Metropolis light transport [Veach 1998], many industrial renderers continue to rely on optimized unidirectional path tracers [Hill et al. 2014]: they are simple to implement and, compared to bidirectional methods, generate a single (noisy) path integral estimate more quickly. We target interactive rendering, and so a 1-sample unidirectionally path-traced estimate is the most efficient way of generating the input to our technique.

We detail our path tracing implementation below, including the measures we take to reduce the variance of our estimate without incurring any substantial additional cost. Afterwards, we discuss the interaction between our renderer and our reconstruction algorithm.

3.1 Interactive Path Tracer

We use an optimized path tracer to produce our noisy input images. Traditional path tracers [Kajiya 1986] shoot rays through each pixel, stochastically scattering according to the profile of the intersected object's reflectance, and continuing recursively until striking a light source. We employ *next event estimation* to improve convergence by deterministically connecting each path vertex to a light.

To accelerate visible surface determination, we leverage GPUs to rasterize (instead of ray tracing) the first hit point from the camera and store its associated shading attributes in a G-Buffer [Deering et al. 1988]: we store the hit mesh ID, mesh primitive ID, triangle intersection barycentric coordinates, material ID, world-space position and shading normal, diffuse and specular albedo, and motion vectors. After this rasterization pass, we continue tracing the path using an NVIDIA OptiX-based GPU ray tracer [Parker et al. 2010].

We choose to not consider depth of field and motion blur during path tracing, since these effects can be efficiently implemented as post-processes, and moreover, they introduce noise in the G-Buffer.

We use low-discrepancy Halton sequences [Halton 1964] when sampling the light source and scattering directions, and apply path space regularization to glossy and specular materials after scattering [Kaplanyan and Dachsbacher 2013]. This significantly reduces the number of sparse high-intensity outliers in glossy reflections, at the cost of a small bias.

We limit the number of indirect bounces to one for practical interactivity. As such, our path tracer generates up to one direct lighting path (camera-surface-light) and one indirect path (camera-surface-surface-light) at each pixel. The total input generation cost per pixel comprises rasterization, three rays, two material evaluations, and one material sampling. Throughout the paper, we call it a *one-sample image* to emphasize that we trace one path, even though it has two next event estimations along its way.

3.2 Auxiliary Inputs for Reconstruction

Our G-Buffer contains rich information about geometry, materials, and light sources of the scene. We make a subset of this available to the reconstruction by exporting a *deep image*, which consists of multiple buffers. In addition to the noisy, high-dynamic range RGB image, we export the following set of G-Buffer features from the rasterization pass to the reconstruction algorithm: view-space shading normals (a 2D vector), depth, and the material's roughness. In total the input to the reconstruction algorithm thus consists of $3 + 4 = 7$ scalar values per pixel.

The color values are stored in linear space as 16-bit half precision floating point (FP16) values to retain high dynamic range (HDR). We linearize the depth values for higher accuracy and store them as FP16. The remaining 3 channels are stored with 8 bits per channel. We calculate the view space shading normal using the camera's projection matrix and store its x and y components.

We further simplify the input by demodulating the noisy RGB image by the albedo of the directly visible material. By storing this *untextured illumination* we remove most of the texture complexity from the noisy image, significantly facilitating training and reducing the required network capacity. After the untextured illumination has been reconstructed, we re-modulate by the albedo in order to include the texture detail in the final rendering.

As we sample directly visible surfaces only once at each pixel, all of the aforementioned inputs are prone to image-space aliasing. Antialiasing these inputs would necessitate a higher sampling rate, precluding interactive rendering. We found, however, that applying a readily available screen-space antialiasing technique to the reconstructed output image instead resolved remaining aliasing at a negligible added cost (see Section 5 for details).

4 IMAGE SEQUENCE RECONSTRUCTION WITH RECURRENT DENOISING AUTOENCODER

Our image reconstruction algorithm is a data-driven method that learns a mapping from noisy input image sequences to noise-free output image sequences based on a large number of training pairs, each consisting of an example input sequence and the desired output sequence (i.e. training target).

We base our reconstruction method on the recent, very general work on image restoration using a convolutional network with hierarchical skip connections [Mao et al. 2016]. Their network is a simple CNN where each layer has 128 kernels with a 3×3-pixel spatial support. The best performing variant has 30 layers, each operating on the same spatial resolution. A skip connection is added from every second layer to the corresponding layer at the end of the network, i.e., first layer is connected to the last layer, third layer to last minus two, and so on. While their network removes additive Gaussian noise very well from individual images, it has various weaknesses considering our application. It is too slow because each layer operates on a full-resolution image and it has problems dealing with spatially very sparse samples that are common in Monte Carlo renderings. Perhaps most significantly, the results are not temporally stable because each frame is denoised in isolation.

We address these weaknesses by modifying the architecture to include subsampling and upsampling stages as well as recurrent connections (Sections 4.1 and 4.2). We then continue by discussing the preparation of training data (Section 4.3) and the exact loss function we optimize during training (Section 4.4).

4.1 Denoising Autoencoder with Skip Connections

As depicted in Figure 2, our network architecture includes distinct *encoder* and *decoder* stages that operate on decreasing and increasing spatial resolutions, respectively. This makes it similar to the Flownet [Fischer et al. 2015] and U-Net [Ronneberger et al. 2015] architectures that give good results in optical flow estimation and image segmentation, respectively, and also emphasizes the connection to *denoising autoencoders* [Vincent et al. 2008]. Autoencoders are networks that learn to reconstruct their inputs from some internal representation, and denoising autoencoders also learn to remove noise from the inputs. We use the term *denoising autoencoder* because we reconstruct from noisy inputs.

Since the layers that operate on the highest spatial resolutions are generally the most time consuming, this design leads to approximately an order of magnitude faster execution compared to Mao et al. [2016], with negligible decrease in quality (for Gaussian noise). The receptive field of all the deeper layers is also several times larger in the input image, allowing us to consider larger pixel neighborhoods and therefore better deal with very sparse inputs.

Because the network learns a mapping from inputs to outputs, it has the desirable property that any number of auxiliary inputs can be provided in addition to the color data. The optimization during training considers all these inputs and automatically finds the best way to use them to disambiguate the color data.

4.2 Recurrent Denoising Autoencoder for Temporal Denoising

Recurrent neural networks (RNN) are used for processing arbitrarily long input sequences. An RNN has feedback loops that connect the output of the previous hidden states to the current ones, thus retaining important information between inputs (Figure 2 right). This makes it a good fit to our application for two reasons. First, in order to denoise a continuous stream of images, we need to achieve temporally stable results. Second, because our inputs are very sparse, the recurrent connections also gather more information about the illumination over time.

In order to retain temporal features at multiple scales, we introduce fully convolutional recurrent blocks after every encoding stage. Related designs have been recently used for video super-resolution [Huang et al. 2015], but we are not aware of earlier applications in the context of an autoencoder with skip connections. It is important that the entire architecture, including the recurrent connections remains fully convolutional. It allows us to train the network with small fixed-size crops (e.g., 128×128 pixels) and later apply it to sequences of arbitrary resolution and length. If there was even a single resolution-specific layer, the property would be lost.

Similarly to Shi et al. [2016], we have found it more efficient to place the recurrent blocks in the encoder part as opposed to the decoder. The reasoning is that the signal is sparser in the encoder. We started with a single recurrent block in the bottleneck of the autoencoder (last layer of the encoder), similar to the design of

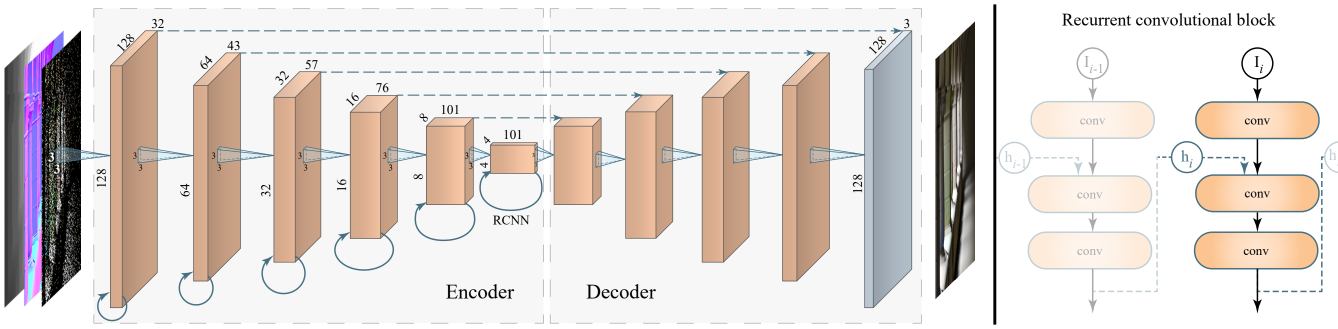


Fig. 2. Architecture of our recurrent autoencoder. The input is 7 scalar values per pixel (noisy RGB, normal vector, depth, roughness). Each encoder stage has a convolution and *2 × 2* max pooling. A decoder stage applies a *2 × 2* nearest neighbor upsampling, concatenates the per-pixel feature maps from a skip connection (the spatial resolutions agree), and applies two sets of convolution and pooling. All convolutions have a *3 × 3*-pixel spatial support. On the right we visualize the internal structure of the recurrent RCNN connections. I is the new input and h refers to the hidden, recurrent state that persists between animation frames.

Pătrăucean et al. [2015], but found it insufficient. A lot of information flows through the skip connections, which causes skipping the recurrent block altogether. Therefore, we place a recurrent block at every encoding stage (Figure 2), right before max pooling.

Each recurrent block consists of three convolution layers with a 3×3-pixel spatial support. One layer processes the input features from the previous layer of the encoder. It then concatenates the results with the features from the previous hidden state, and passes it through two remaining convolution layers. The result becomes both the new hidden state and the output of the recurrent block. This provides a sufficient temporal receptive field and, together with the multi-scale cascade of such recurrent blocks, allows to efficiently track and retain image features temporally at multiple scales. The convolution layers in a recurrent block operate on the same input resolution and the same number of features as the encoding stage it is attached to.

Formally, the output and the hidden state can be represented using a recurrent equation:

$$h_i = O_i = C_{3\times3}\left(C_{3\times3}\left(C_{3\times3}(I_i)^\frown h_{i-1}\right)\right),$$

where $C_{3\times3}$ is a convolution kernel with a 3×3-pixel spatial support as defined in Eq. 1, O_i is the output, I_i is the current input, h_i is the hidden state for the input i, and $^\frown$ is a concatenation operator.

As a precursor to the RNN design, we experimented with training the network with three-frame window sequences as input and target. While this reduced temporal flickering, it still was clearly visible in sequences longer than the training window.

4.3 Training

We will now describe the preparation of training data. We start with a smooth camera fly-through animation with e.g. 1000 frames for each scene available for training. For every frame, we generate 10 different noisy images at 1 sample per pixel, the auxiliary features, and the target image for training. By having multiple noisy images, we can ask that each of these instances of Monte Carlo noise lead to the same reconstructed image. This increases the number of training pairs tenfold at a negligible cost compared to creating additional target images. Note that the noisy images share the auxiliary features because primary rays are rasterized (Section 3.2).

We generate 1024×1024 images during rendering, while the training is performed using smaller 128×128 crops that are randomly selected for each *training sequence*. We use sequences of 7 consecutive frames to provide enough temporal context during training. We also randomly select the scene fly-through sequence, as well as the beginning of a training subsequence within the selected fly-through sequence. We randomly alternate between forward and backward playback in order to train the network on various camera movements. We have also found useful to randomly stop the camera when forming the training sequence. For that, instead of advancing the camera to the next frame, we keep the camera and the target image at the current frame of animation, while advancing the noisy image to a different random seed.

In addition, we use randomly picked rotation of the training sequence by 90/180/270 degrees to train on more movement directions. We also pick a random modulation in the range $[0, 2]$ separately to each color channel, and apply them to the entire sequence. It forces the network to better learn the linear input–target color relation, as well as the independence of the channels.

4.4 Loss Function

A loss function defines how the error between network outputs and training targets is computed during training. The most commonly used loss function in image restoration is L_2, which is the mean squared error between the predicted image P and the target image T. However, it is also known that using L_1 loss instead of L_2 can reduce the splotchy artifacts from reconstructed images [Zhao et al. 2016]. Our first loss term is a spatial L_1 loss, denoted as $\mathcal{L}_s$ for a single image in the temporal training sequence:

$$\mathcal{L}_s = \frac{1}{N}\sum_i^N |P_i - T_i|,$$

where P_i and T_i are the ith pixel of the predicted and target image correspondingly. We have also found it useful to flatten the image by raising all color channels to the power γ before computing the loss. A value of $1/2.2$ would be close to the perceptual gamma correction, however, we found that a more aggressive value of 0.2 allows to penalize the error in the dark regions of the image even more efficiently. Our implementation raises both the input and target images into power 0.2 as a preprocess.

The L_1 spatial loss provides a good overall image metric that is tolerant to outliers. In order to further penalize the differences in fine details, such as edges, we also use a gradient-domain L_1 loss

$$\mathcal{L}_\mathrm{g} = \frac{1}{N}\sum_{i}^{N} |\nabla P_i - \nabla T_i|,$$

where each gradient $\nabla(\cdot)$ is computed using a *High Frequency Error Norm* (HFEN), an image comparison metric from medical imaging [Ravishankar and Bresler 2011]. The metric uses a Laplacian of Gaussian kernel for edge-detection. The Laplacian works to detect edges, but is sensitive to noise, so the image is pre-smoothed with a Gaussian filter first to make edge-detection work better. We used the recommended parameter of $\sigma = 1.5$ for Gaussian kernel size.

These losses minimize the error for each image in isolation. However, they do not penalize temporal incoherence (e.g., flickering between frames), and neither do they encourage the optimizer to train the recurrent connections to pass more data across frames. Therefore, we introduce a temporal L_1 loss $\mathcal{L}_\mathrm{t}$

$$\mathcal{L}_\mathrm{t} = \frac{1}{N}\sum_{i}^{N} \left(\left| \frac{\partial P_i}{\partial t} - \frac{\partial T_i}{\partial t} \right| \right),$$

where a temporal derivative $\partial P_i / \partial t$ for an ith image pixel is computed using finite differencing in time between the ith pixels of the current and the previous image in the temporal training sequence.

We use a weighted combination of these three losses as the final training loss

$$\mathcal{L} = w_\mathrm{s}\, \mathcal{L}_\mathrm{s} + w_\mathrm{g}\, \mathcal{L}_\mathrm{g} + w_\mathrm{t}\, \mathcal{L}_\mathrm{t},$$

where $w_{\mathrm{s/g/t}}$ are the adjustable weights that control the contribution of each loss. We picked $w_{\mathrm{s/g/t}} = 0.8/0.1/0.1$ to approximately equalize the scales, which improved convergence.

We have also found it important to assign higher weight to the loss functions of frames later in the sequence to amplify temporal gradients, and thus incentivize the temporal training of RNN blocks. We use a Gaussian curve to modulate $w_{\mathrm{s/g/t}}$: for a sequence of 7 images we use $(0.011, 0.044, 0.135, 0.325, 0.607, 0.882, 1)$.

To verify that the combined loss leads to an improvement over the spatial-only loss $\mathcal{L}_\mathrm{s}$, we measured the structural similarity metric (SSIM) [Wang et al. 2004] on a validation sequence in SponzaDiffuse after 100 epochs of training. SSIM showed an improvement from 0.9335 for $\mathcal{L}_\mathrm{s}$ to 0.9417 for the combined loss.

5 IMPLEMENTATION AND RESULTS

We will now describe the design parameters and study the convergence properties of our network. Then we will detail the practical aspects of training and using the network for interactive reconstruction. Finally we will focus on the performance measurements and compare the resulting quality against the state-of-the-art.

5.1 Network Design Analysis

We study the convergence behavior of our network by varying its layer count and the set of auxiliary input features. Also, when a trained network is applied to an image, it acts as a reconstruction filter.

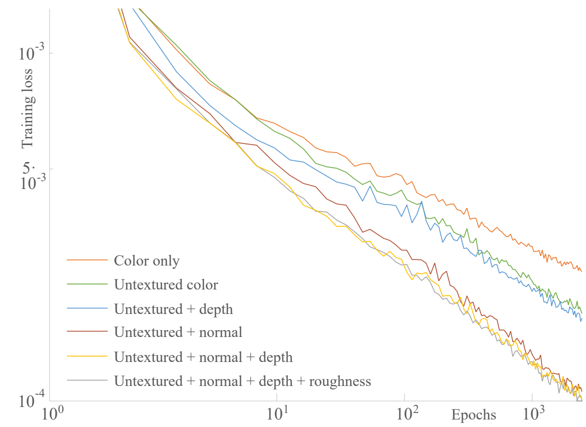


Fig. 3. The convergence of average training loss as a function of epochs for our network trained with and without auxiliary features.

Therefore, we demonstrate its qualitative properties by artificially applying the network repeatedly to the same image.

Network Size. Figure 4 shows the convergence behavior for a varying number of encoder and decoder stages, as well as for a different number of feature maps. We observe that after a certain design point the training loss decreases only marginally while the computational cost keeps increasing. We select the smallest configuration that gives good results, highlighted in yellow on the left. In total there are 36 convolution layers in our network: 18 in the feedforward path (7 in encoder, 10 in decoder and one output layer, see Figure 4a) and 3 in each of the 6 RCNN blocks (Figure 2, right).

We set the output feature count to 32 per pixel in the first stage of the encoder, and then multiply the number by $4/3$ after every subsampling operation. This leads to a fixed-rate compression by a factor of $4/(4/3) = 3$ after every subsampling. Therefore, information is generally lost at every stage, but it gets reintroduced through the skip connections. The decoder part then amplifies the amount of data by 3× after every upsampling.

Auxiliary Features. Figure 3 shows a convergence plot for training loss averaged over 10 experiments using the SponzaDiffuse scene (Figure 6). We start by switching to the untextured color, and then gradually grow the set of auxiliary features to observe the network's training characteristics.

We observe that untextured lighting—demodulating the image with the albedo at a primary hit—significantly improves the convergence speed. Another big improvement is obtained by introducing normals as an auxiliary feature. Normals help the network to detect silhouettes of objects and to better detect discontinuities in shading. Additional, smaller improvements are obtained from depth and roughness. In general, the auxiliary features help to disambiguate the colors by providing information about the contours and silhouettes of the scene objects, as well as about different materials.

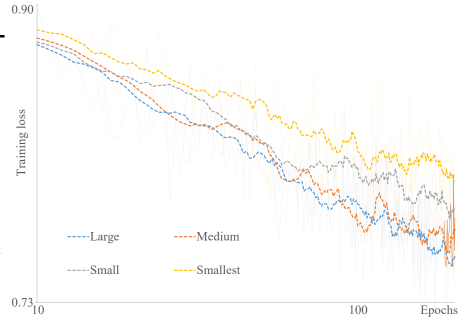
Reconstruction Filter's Properties. In order to demonstrate the qualitative behavior of the proposed reconstruction filter, we artificially applied it to an input image multiple times recursively. That is, we first take a noisy color image and the auxiliary features, perform

AE smallest	AE small	AE	AE large
conv-32,32	conv-32,32	conv-32,32	conv-32,32
conv-43	conv-43	conv-43	conv-43
conv-57	conv-57	conv-57	conv-57
conv-57	**conv-76**	conv-76	conv-76
	conv-76	**conv-101**	conv-101
		conv-101	**conv-135**
			conv-135
			conv-101*2
		conv-76*2	conv-76*2
	conv-57*2	conv-57*2	conv-57*2
conv-43*2	conv-43*2	conv-43*2	conv-43*2
conv-32*2	conv-32*2	conv-32*2	conv-32*2
conv-64	conv-**128,64**	conv-128,64	conv-128,64

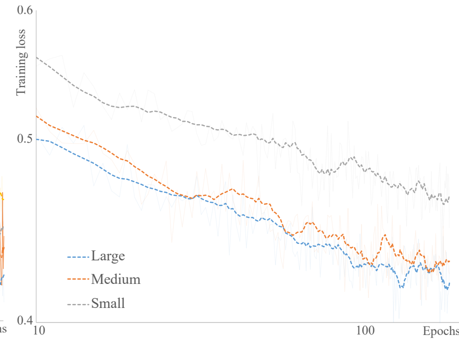
a)

AE feat small	AE feat	AE feat large
conv-24,24	conv-**32,32**	conv-**48,48**
conv-36	conv-**43**	conv-**58**
conv-54	conv-**57**	conv-**69**
conv-81	conv-**76**	conv-**83**
conv-122	conv-**101**	conv-**100**
conv-122	conv-**101**	conv-**100**
conv-81*2	conv-**76***2	conv-**83***2
conv-54*2	conv-**57***2	conv-**69***2
conv-36*2	conv-**43***2	conv-**58***2
conv-24*2	conv-**32***2	conv-**48***2
conv-128,64	conv-128,64	conv-128,64

b)

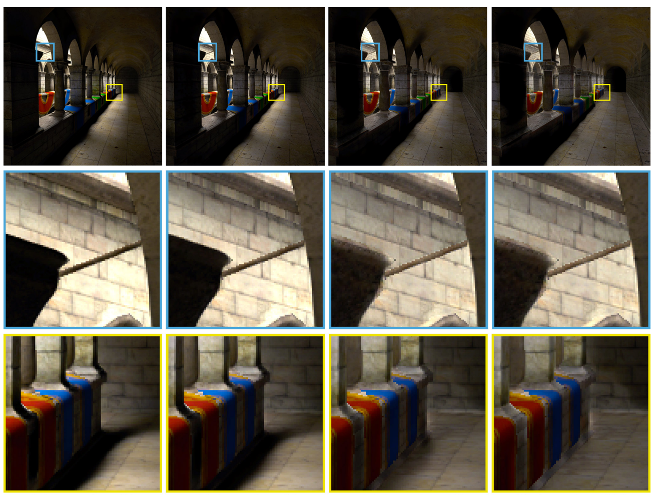


c)



d)

Fig. 4. Ablation study of our network design. Parameter sweeps on **a)** the number of layers and **b)** the number of features. Here conv-N means a convolution layer with N output features, and $*2$ indicates that there are two such layers back-to-back. Additionally, there is an output layer with 3 output features (R,G,B). The highlighted columns show the best trade-off between size and training loss. **c)** Training loss convergence plots for networks trained with different number of layers from a). **d)** Training loss convergence plots for networks trained with different number of output features from b).



(a) iteration #1 (b) iteration #2 (c) iteration #5 (d) iteration #10

Fig. 5. We perform a qualitative analysis of the reconstruction filter's properties by applying the filter to the input image recursively for 1/2/5/10 iterations. The filter acts as an edge-preserving smoothing filter in some regions, while acting as a contrast or unsharp masking filter in others.

the inference once, then take the result and use it as a "noisy" color image for another inference (features stay unchanged), and so on.

Figure 5 shows the qualitative analysis of the filter by applying the filter recursively 1/2/5/10 times to a frame from SponzaDiffuse. The middle row shows an inset with a subtle trim shadow on top, which is gradually spread around the whole region after multiple iterations, with a flag pole being a stop for the filter. This means that the filter behaves as an edge-preserving kernel, i.e., it tries to spread around the noisy samples, while preserving the edges. On the other hand, the filter also tries to preserve the local contrast. For example, in the bottom row in Figure 5, even after 10 iterations, the illumination is mostly blurred, however, the filter still tries to preserve the initial contrast along the edge of the far corner.

SponzaDiffuse SponzaGlossy Classroom

Fig. 6. We train the network using fly-throughs of these 3 scenes.

5.2 Training Data vs. Generalization

Our network has a total of 3.2 million trainable parameters, and even though convolutional networks are naturally tolerant to overfitting, we need to make sure that our training set is sufficiently large. We know from earlier image restoration results (e.g. [Mao et al. 2016]) that our training set should contain at least hundreds of image sequences that offer considerable variety if we want the trained network to work well in all scenes.

While fly-throughs of 3D scenes offer a convenient way of gathering arbitrary amounts of training data, variation can be a concern. The network learns to reconstruct the kind of features it sees during training, and if the training set does not, for example, have any high-frequency features such as vegetation, its ability to reconstruct such (unseen) features can remain limited. Ideally, we could thus train the network with dozens of very different scene geometries, lighting setups, and camera motions.

We experiment with this setup by using the three scenes shown in Figure 6 (SponzaDiffuse, SponzaGlossy and Classroom) for training the network, and then use it for reconstructing fly-throughs of other scenes as well. *All results in this paper and in the accompanying video were reconstructed using this training setup, unless explicitly stated otherwise.* SponzaDiffuse is the Crytek Sponza scene with the default set of materials and a large area light source on top of it. For SponzaGlossy we further modify the materials by replacing the diffuse BRDF with a glossy GGX BRDF with roughness $\alpha = 0.06$; the same textures are reused for the specular color. The light source moves slowly along the long axis of the scene during the

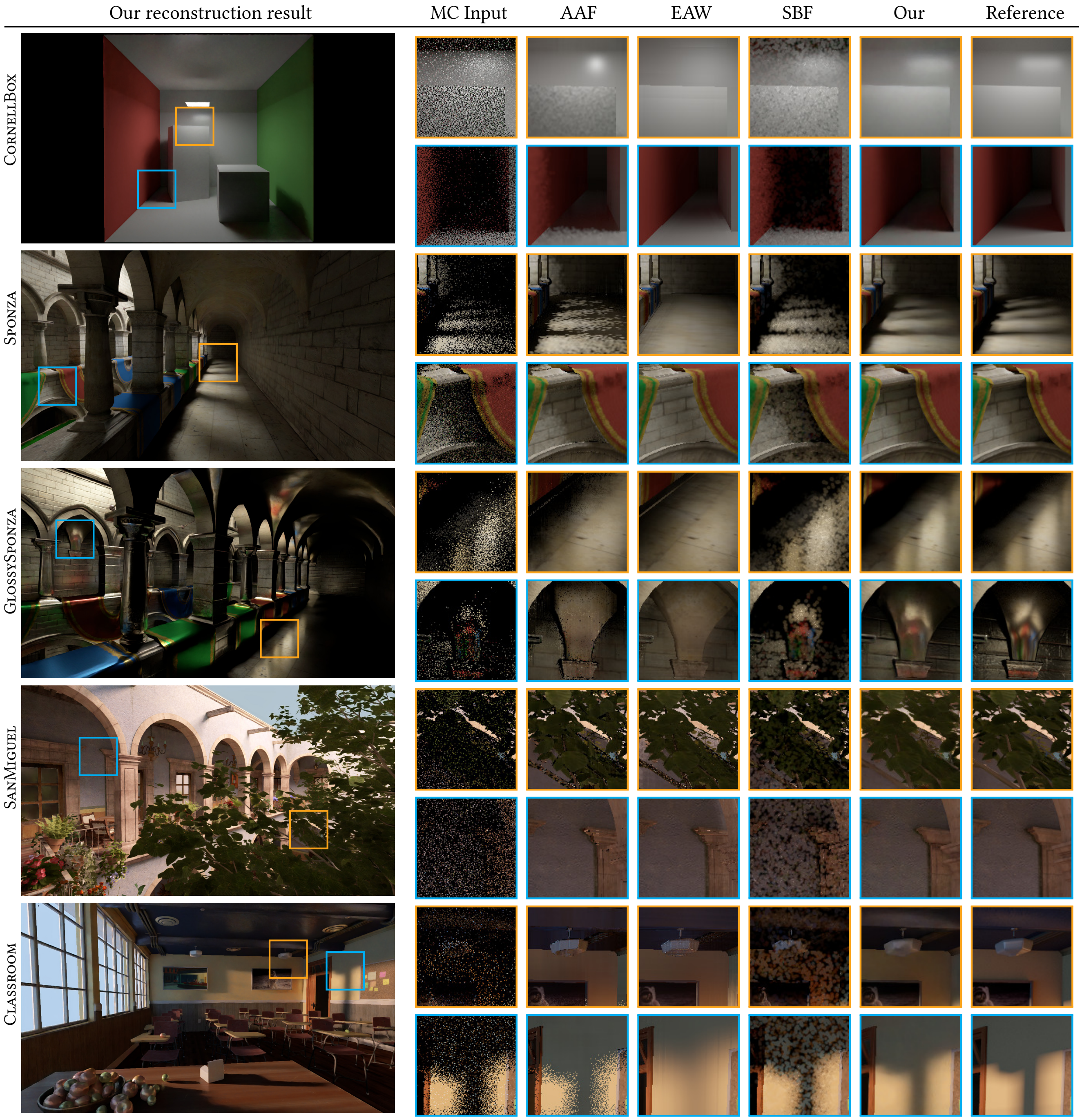


Fig. 7. Closeups of 1-bounce global illumination results for 1 spp input (MC), axis-aligned filter (AAF), À-Trous wavelet filter (EAW), SURE-based filter (SBF), and our result. Statistics are in Figure 8. Please consult the supplemental material for full resolution images and video sequences.

fly-through to provide changes in illumination. Classroom has one directional light source, sky illumination, and a rich set of textures, thin geometric shapes, and layered materials. The target images are rendered with 2000 spp for SponzaDiffuse, and with 4000 for the other scenes.

An alternative solution that would sidestep the concern of sufficient scene variety in the training set is to train a network specifically for a single scene (or a small set of scenes). This could be an attractive solution for example for a game level; the training would become a part of the "baking" step and the network would work well

Filter	CornellBox		Sponza		GlossySponza		SanMiguel		Classroom	
	RMSE	SSIM	RMSE	SSIM	RMSE	SSIM	RMSE	SSIM	RMSE	SSIM
AAF	0.018	0.978	0.029	0.869	0.069	0.664	0.079	0.774	0.064	0.786
EAW	0.019	0.978	0.043	0.902	0.089	0.586	0.088	0.768	0.059	0.841
SBF	0.044	0.818	0.060	0.502	0.057	0.649	0.087	0.575	0.072	0.610
Our	**0.014**	**0.984**	**0.016**	**0.953**	**0.034**	**0.843**	**0.055**	**0.844**	**0.040**	**0.909**

Fig. 8. Error statistics for still images from Figure 7.

for arbitrary image sequences of that one scene—but possibly poorly for any other content. We demonstrate the ability of the network to specialize on a particular data set by performing a separate training on SanMiguel using 600 input frames and 2000 spp target images that we generated specifically for this experiment. The supplementary material shows the inference results on the same scene with different camera motion. All other images and videos, including the SanMiguel results, do not include this training set.

5.3 Implementation of Training

We implemented the training of our network using Lasagne [Dieleman et al. 2015] and Theano [2016], and used NVIDIA DGX-1 for training. The recurrent blocks are trained by back propagating through time [Werbos 1988], where the feed-forward subparts of the RNN are replicated to unroll the recurrence loops. The training time for 500 epochs is approximately 16 hours on a single GPU, of which 1 hour goes into preprocessing the dataset so that random crops can be efficiently fetched using `memmap`.

We train for 500 epochs using Adam [Kingma and Ba 2014] with learning rate 0.001 and decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.99$. The learning rate is ramped up tenfold using a geometric progression during the first 10 training epochs, and then decreased according to $1/\sqrt{t}$ schedule. We use a minibatch size of 4 sequences, and each epoch randomizes the order of training data. All parameters are initialized following He et al. [2015b], and leaky ReLU activation [Maas et al. 2013] with $\alpha = 0.1$ is used in all layers except the last one, which uses linear activation. Max pooling is used for subsampling and nearest neighbor filtering for upsampling. The exact choice of these various parameters tends to affect the results only slightly; we refer an interested reader to a recent survey [Mishkin et al. 2016].

5.4 Reconstruction Quality with Low Sample Counts

In order to compare with the prior work, we have implemented multiple state-of-the-art algorithms including axis-aligned filter (AAF) for both soft shadows [Mehta et al. 2012] and indirect illumination [Mehta et al. 2013], edge-avoiding À-Trous wavelet filter (EAW) [Dammertz et al. 2010], SURE-based filter (SBF) [Li et al. 2012], and learning based filter (LBF) [Kalantari et al. 2015].

Since our ray budget allows for only one shadow ray per pixel, for axis aligned shadow filter [Mehta et al. 2012] we gather the minimum and maximum occlusion distance within a 7×7-pixel window to estimate the minimum and maximum slopes in the light field frequency spectrum. In our implementation of Li et al. [2012], we do not perform any adaptive sampling. We have only one sample/pixel, and thus we estimate the color variance using a 2×2-pixel spatial window. Thanks to a noise-free G-buffer, we also skip the normalization by variance (Eq.6 in the paper) for the auxiliary features.

Fig. 9. Comparison at 4 samples/pixel. Learning-based filter trained on 4/8/16/32 spp (left, SSIM: 0.8280) and recurrent autoencoder trained on 1 spp (right, SSIM: 0.9074). Our network was not trained on this scene.

Temporal antialiasing (TAA) provides pixel-scale antialiasing at a negligible cost and also reduces the flickering of small features at a cost of subtle blur [Karis 2014]. Most of the prior methods are not temporally stable at a single sample/pixel, and in order to do fair comparisons, we apply TAA as a supplemental post-process pass to all methods. Our method also benefits from TAA because the recurrent loops at finer resolutions have insufficient temporal receptive field to reproject the high-frequency features. The video offers an evaluation of temporal stability between the methods.

Figure 7 shows the main results and closeups from the five scenes. CornellBox demonstrates the ability of our network to preserve the hard features, such as contact shadows and edges while handling the scant and flat appearance. SponzaDiffuse, SponzaGlossy and Classroom scenes were used for training, albeit with different camera fly-throughs and Classroom also had a different lighting setup during training. This demonstrates the ability of the network to adapt to arbitrary viewpoints and different lighting setups. SanMiguel scene was never presented to the network prior to the inference. This demonstrates the network's ability to generalize to completely new data, where the illumination, materials, and scene geometry differ from what the network was trained on. For example, none of our training sets contain foliage, which is usually challenging to filter. The video also shows that our network generalizes convincingly to SponzaSpecular that has specular materials, even though pure specularities were not present in the training set. It is therefore somewhat surprising that it works as well as it does, and a training-theoretical argument can be made that further quality improvements should be possible through more varied training. Figure 8 provides root-mean-square error (RMSE) and structural similarity metric (SSIM) measurements for the results in Figure 7.

Figure 10 demonstrates that our method is agnostic to the input and can also filter other challenging situations, such as complex soft and hard contact shadows, at a reasonable quality.

Higher Sample Count and Number of Bounces. The learning based filter (LBF) was trained on 20 scenes that simulated a wide variety of Monte Carlo effects at 4, 8, 16, 32, and 64 samples/pixel. In order to perform a fair comparison, we run LBF and our filter using a 4 spp input (Figure 9). Even though our network was trained only with 1 spp inputs, we can see that it generalizes well to inputs with 4 samples per pixel, and the result quality surpasses LBF.

In Figure 11 we have also demonstrated that the autoencoder trained on 1spp scenario can generalize to significantly higher sample count as well as to a higher number of bounces. We generated images using path tracing with up to three bounces and 256 samples

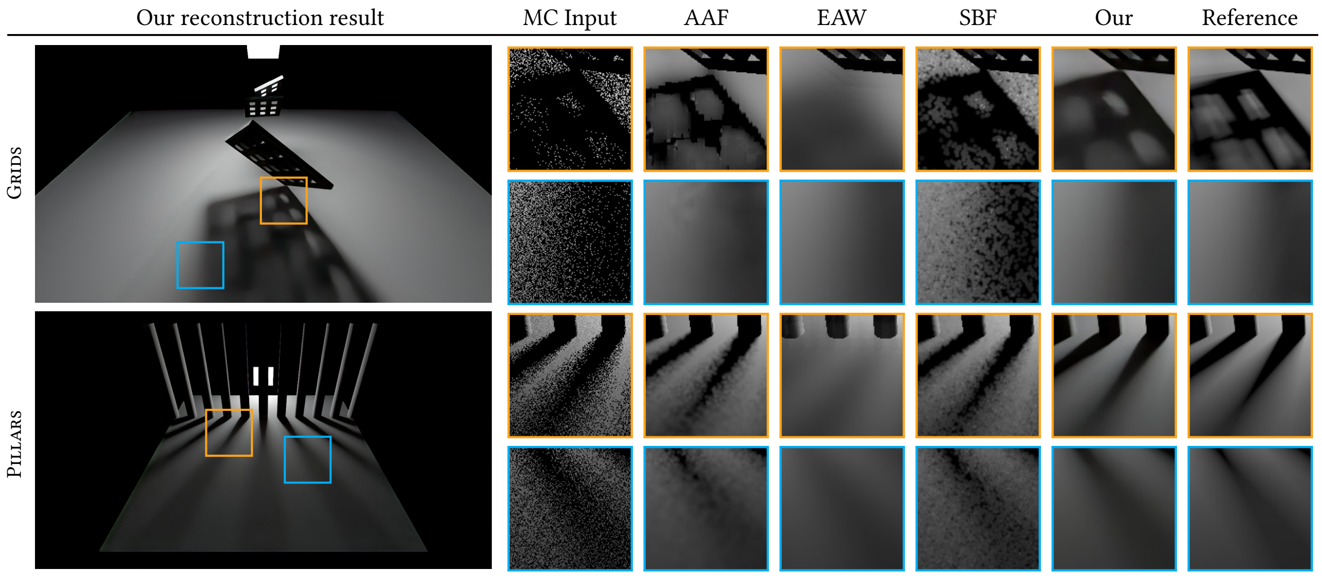


Fig. 10. Closeups for shadow filtering for 1 spp input (MC), axis-aligned filter (AAF), À-Trous wavelet filter (EAW), SURE-based filter (SBF), and our result.

per pixel. We also provide a comparison with the state-of-the-art offline denoiser using Nonlinearly-weighted First Order Regression (NFOR) [Bitterli et al. 2016]. We have found that even when trained with 1spp, our method can generalize to the input with higher sample count and higher number of bounces, providing reasonable denoising quality within its performance ballpark (Figure 12).

Failure cases. If the training data is insufficient, or the samples are too sparse to disambiguate important output features, the result will lack detail and can produce splotchy-looking results. Visually, this results in over-smoothed images with approximately correct colors (often also with edge preservation) and sometimes painted appearance, as can be seen in the crop with thin shadows on the floor in Sponza row in Figure 7. If there are not enough samples, some features (e.g., a small specular highlight) can be also missing, like the glossy reflections on the lamp fixture in the crop of the Classroom row. On the other hand, the tendency of the network to produce an average answer often provides a suboptimal solution. Figure 11 shows that the network trained with 1spp can provide good overall results even on a 256spp input, however, offline methods specifically crafted for this noise level perform systematically better than ours when using quantitative error metrics (Figure 12).

5.5 Reconstruction Performance

We implemented the inference (i.e. runtime reconstruction) using fused CUDA kernels and cuDNN 5.1[1] convolution routines with Winograd optimization.

We were able to achieve highly interactive performance on the latest GPUs. For a 720p image (1280 × 720 pixels), the reconstruction time was 54.9ms on NVIDIA (Pascal) Titan X. The execution time scales linearly with the number of pixels.

The performance of the comparison methods varies considerably. EAW (10.3ms) is fast, while SBF (74.2ms), AAF (211ms), and LBF (1550ms) are slower than our method (54.9ms). The NFOR method has a runtime of 107–121s on Intel i7-7700HQ CPU. Our comparisons are based on the image quality obtainable from a fixed number of input samples, disregarding the performance differences. That said, the performance of our OptiX-based path tracer varies from 70ms in SponzaGlossy to 260ms in SanMiguel for 1 sample/pixel. Thus in this context, until the path tracer becomes substantially faster, it would be more expensive to take another sample/pixel than it is to reconstruct the image using our method.

Furthermore, our method is a convolutional network, and there is a strong evidence that the inference of such networks can be accelerated considerably by building custom reduced-precision hardware units for it, e.g., over 100× [Han et al. 2016]. In such a scenario, our method would move from highly interactive speeds to the realtime domain.

6 CONCLUSIONS AND FUTURE WORK

We presented the first application of recurrent denoising autoencoders, and deep convolutional networks in general, to the problem of light transport reconstruction, producing noise-free and temporally coherent animation sequences with global illumination. We can see several interesting avenues of future work for this approach.

In this work, we demonstrated state-of-the-art quality in interactive reconstruction. Looking ahead, we would like to study how much the results can be improved by introducing more varied training material. In addition, it could be beneficial to specialize (and possibly simplify) the design to target lower-dimensional distribution effects, such as motion blur and depth of field. It is likely possible to extend our method to handle these effects by providing lens and time coordinates as inputs to the network. It will also be interesting to apply these ideas to the high-sample rate regime of

[1] https://developer.nvidia.com/cudnn

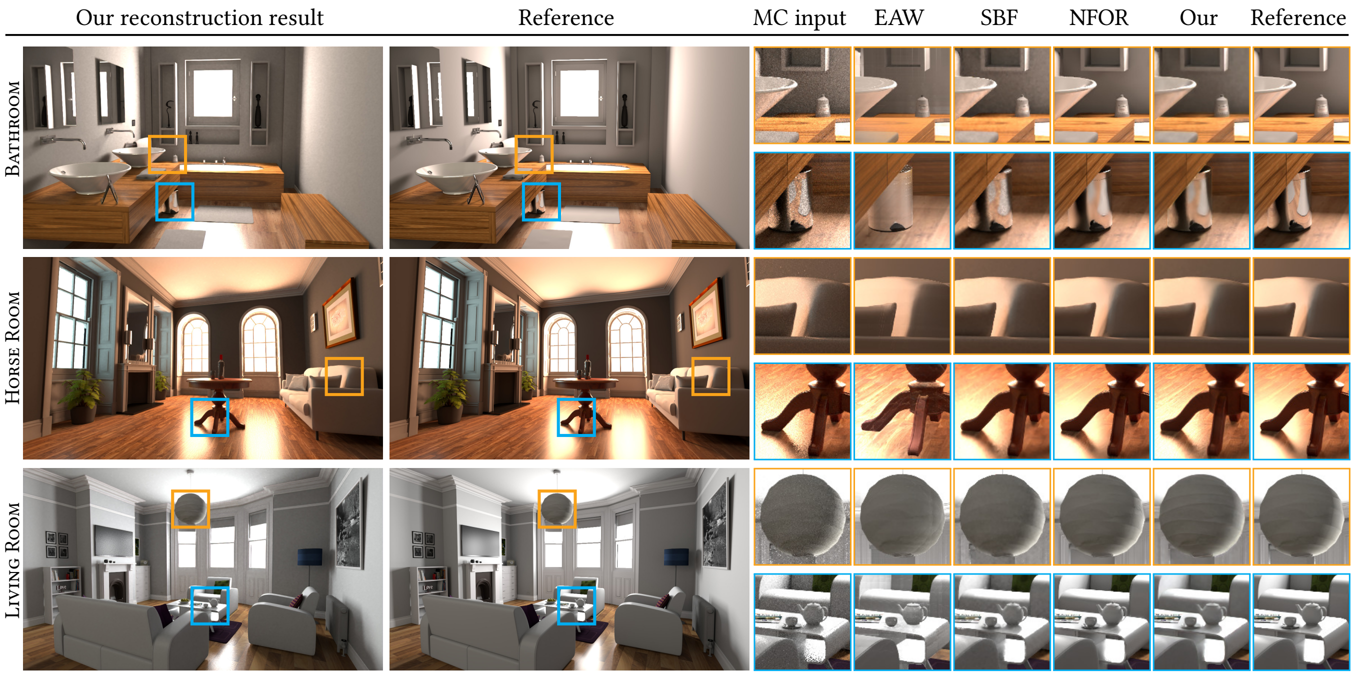


Fig. 11. Generalization example: Autoencoder trained on 1spp data was applied to 256spp MC input images. Closeups of 2-bounce global illumination input, À-Trous wavelet filter (EAW), SURE-based filter (SBF), Nonlinearly-weighted First Order Regression (NFOR), and our result.

Filter	Bathroom		Horse Room		Living Room		Frame time
	RMSE	SSIM	RMSE	SSIM	RMSE	SSIM	
EAW	0.127	0.840	0.094	0.831	0.071	0.870	10.3ms
SBF	0.044	0.902	0.040	0.939	0.042	0.931	74.2ms
NFOR	**0.009**	**0.990**	**0.018**	**0.984**	**0.012**	**0.984**	110s (CPU)
Our	0.041	0.944	0.034	0.945	0.029	0.942	54.9ms

Fig. 12. Errors for images with high sample count (256 spp) from Figure 11.

film-quality rendering, where a much smaller amount of noise remains in input images but the geometry (e.g. hair) is also orders of magnitude more detailed.

7 ACKNOWLEDGMENTS

We thank Jon Hasselgren and Jacob Munkberg for optimizing the inference performance with Cuda and cuDNN, Shiqiu Liu and John Burgess for help with the comparisons. Alla Chaitanya was supported by a Doctoral Research Scholarship from the Fonds de recherche du Québec – Nature et technologies, a J. Armand Bombardier Scholarship at the University of Montreal, and a Engineering Doctoral Award at McGill University. We also thank Jan Kautz and his group for valuable suggestions, as well as to Dmitry Korobchenko for discussions about the loss function. We thank Nir Benty and Kai-Hwa Yao for assisting with the Falcor framework, to John Spitzer and Alexander Evstigneev for providing game testing infrastructure for training data generation, to Nicholas Hull and Anjul Patney for adjusting the scenes, as well as to Fabrice Rousselle and the Disney Research group for helping with the NFOR comparison.

REFERENCES

Bako, S., Vogels, T., McWilliams, B., Meyer, M., Novák, J., Harvill, A., Sen, P., DeRose, T., and Rousselle, F. 2017. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Trans. Graph.* (2017).

Bauszat, P., Eisemann, M., John, S., and Magnor, M. 2015. Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field. *Comp. Graph. Forum* 34, 1 (2015).

Bauszat, P., Eisemann, M., and Magnor, M. 2011. Guided Image Filtering for Interactive High-quality Global Illumination *(Comp. Graph. Forum)*. 8.

Bikker, J. and van Schijndel, J. 2013. The brigade renderer: A path tracer for real-time games. *J. Comp. Games Tech.* 8 (2013).

Bitterli, B., Rousselle, F., Moon, B., Iglesias-Guitián, J. A., Adler, D., Mitchell, K., Jarosz, W., and Novák, J. 2016. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. *Comp. Graph. Forum* 35, 4 (2016).

Dammertz, H., Sewtz, D., Hanika, J., and Lensch, H. 2010. Edge-avoiding À-Trous wavelet transform for fast global illumination filtering. In *Proc. High Perf. Graph.*

Deering, M., Winner, S., Schediwy, B., Duffy, C., and Hunt, N. 1988. The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics. *Proc. SIGGRAPH* 22, 4 (1988).

Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S. K., and others. 2015. Lasagne: First release. (2015).

Dosovitskiy, A., Springenberg, J. T., and Brox, T. 2015. Learning to generate chairs with convolutional neural networks. In *Proc. CVPR*. 1538–1546.

Egan, K., Durand, F., and Ramamoorthi, R. 2011a. Practical Filtering for Efficient Ray-traced Directional Occlusion. *ACM Trans. Graph.* 30, 6 (2011).

Egan, K., Hecht, F., Durand, F., and Ramamoorthi, R. 2011b. Frequency Analysis and Sheared Filtering for Shadow Light Fields of Complex Occluders. *ACM Trans. Graph.* 30, 2 (2011).

Egan, K., Tseng, Y.-T., Holzschuch, N., Durand, F., and Ramamoorthi, R. 2009. Frequency Analysis and Sheared Reconstruction for Rendering Motion Blur. *ACM Trans. Graph.* 28, 3 (2009).

Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D., and Brox, T. 2015. FlowNet: Learning Optical Flow with Convolutional Networks. *ArXiv e-prints* (2015). arXiv:1504.06852.

Flynn, J., Neulander, I., Philbin, J., and Snavely, N. 2016. DeepStereo: Learning to Predict New Views From the World's Imagery. In *Proc. CVPR*.

Gastal, E. S. and Oliveira, M. M. 2012. Adaptive manifolds for real-time high-dimensional filtering. *ACM Trans. Graph.* 31, 4 (2012).

Girshick, R., Donahue, J., Darrell, T., and Malik, J. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proc. CVPR*. 580–587.

Halton, J. H. 1964. Algorithm 247: Radical-inverse Quasi-random Point Sequence. *Comm. ACM* 7, 12 (1964).

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proc. Comp. Arch.*

Hanika, J., Dammertz, H., and Lensch, H. 2011. Edge-Optimized À-Trous Wavelets for Local Contrast Enhancement with Robust Denoising. *Comp. Graph. Forum* 30, 7 (2011).

He, K., Zhang, X., Ren, S., and Sun, J. 2015a. Deep Residual Learning for Image Recognition. *ArXiv e-prints* (2015). arXiv:1512.03385.

He, K., Zhang, X., Ren, S., and Sun, J. 2015b. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proc. IEEE Conf. Comp. Vis.*

Hill, S., McAuley, S., Burley, B., Chan, D., Fascione, L., Iwanicki, M., Hoffman, N., Jakob, W., Neubelt, D., Pesce, A., and Pettineo, M. 2015. Physically Based Shading in Theory and Practice. In *SIGGRAPH Courses*. Article 22.

Hill, S., McAuley, S., Dupuy, J., Gotanda, Y., Heitz, E., Hoffman, N., Lagarde, S., Langlands, A., Megibben, I., Rayani, F., and de Rousiers, C. 2014. Physically Based Shading in Theory and Practice. In *SIGGRAPH Courses*. Article 23.

Huang, Y., Wang, W., and Wang, L. 2015. Bidirectional recurrent convolutional networks for multi-frame super-resolution. In *Proc. NIPS*.

Iizuka, S., Simo-Serra, E., and Ishikawa, H. 2016. Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. *ACM Trans. Graph.* 35, 4 (2016).

Jensen, H. W. and Christensen, N. J. 1995. Optimizing Path Tracing using Noise Reduction Filters. In *Proc. WSCG*.

Kajiya, J. T. 1986. The rendering equation. In *Proc. SIGGRAPH*.

Kalantari, N. K., Bako, S., and Sen, P. 2015. A machine learning approach for filtering Monte Carlo noise. *ACM Trans. Graph.* 34, 4 (2015).

Kaplanyan, A. S. and Dachsbacher, C. 2013. Path Space Regularization for Holistic and Robust Light Transport. *Comp. Graph. Forum* 32, 2 (2013).

Karis, B. 2014. High-quality temporal supersampling. In *Advances in Real-Time Rendering in Games, SIGGRAPH Courses*.

Keller, A., Fascione, L., Fajardo, M., Georgiev, I., Christensen, P., Hanika, J., Eisenacher, C., and Nichols, G. 2015. The path tracing revolution in the movie industry. *SIGGRAPH Courses* (2015).

Kingma, D. and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *ArXiv e-prints* (2014). arXiv:1412.6980.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*.

Larsson, G., Maire, M., and Shakhnarovich, G. 2016. Learning Representations for Automatic Colorization. In *Proc. ECCV*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. 1998. Gradient-based learning applied to document recognition. In *Proc. IEEE*.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., and Shi, W. 2016. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *ArXiv e-prints* (2016). arXiv:1609.04802.

Lee, M. E. and Redner, R. A. 1990. A note on the use of nonlinear filtering in computer graphics. *IEEE Comp. Graph. App.* 10, 3 (1990).

Lehtinen, J., Aila, T., Laine, S., and Durand, F. 2012. Reconstructing the indirect light field for global illumination. *ACM Trans. Graph.* 31, 4 (2012).

Li, T.-M., Wu, Y.-T., and Chuang, Y.-Y. 2012. SURE-based Optimization for Adaptive Sampling and Reconstruction. *ACM Trans. Graph.* 31, 6 (2012).

Maas, A. L., Hannun, A. Y., and Ng, A. Y. 2013. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, Vol. 30.

Mao, X., Shen, C., and Yang, Y. 2016. Image Restoration Using Convolutional Autoencoders with Symmetric Skip Connections. In *Proc. NIPS*. arXiv:1606.08921.

McCool, M. D. 1999. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Trans. Graph.* 18, 2 (1999).

Mehta, S. U., Wang, B., and Ramamoorthi, R. 2012. Axis-aligned Filtering for Interactive Sampled Soft Shadows. *ACM Trans. Graph.* 31, 6 (2012).

Mehta, S. U., Wang, B., Ramamoorthi, R., and Durand, F. 2013. Axis-aligned Filtering for Interactive Physically-based Diffuse Indirect Lighting. *ACM Trans. Graph.* 32, 4 (2013).

Mehta, S. U., Yao, J., Ramamoorthi, R., and Durand, F. 2014. Factored Axis-aligned Filtering for Rendering Multiple Distribution Effects. *ACM Trans. Graph.* 33, 4 (2014).

Mishkin, D., Sergievskiy, N., and Matas, J. 2016. Systematic evaluation of CNN advances on the ImageNet. *ArXiv e-prints* (2016). arXiv:1606.02228.

Moon, B., Iglesias-Guitian, J. A., Yoon, S.-E., and Mitchell, K. 2015. Adaptive rendering with linear predictions. *ACM Trans. Graph.* 34, 4 (2015).

Moon, B., McDonagh, S., Mitchell, K., and Gross, M. 2016. Adaptive polynomial rendering. *ACM Trans. Graph.* 35, 4 (2016).

Munkberg, J., Hasselgren, J., Clarberg, P., Andersson, M., and Akenine-Möller, T. 2016. Texture space caching and reconstruction for ray tracing. *ACM Trans. Graph.* 35, 6 (2016).

Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Trans. Graph.* 29, 4 (2010).

Pathak, D., Krähenbühl, P., Donahue, J., Darrell, T., and Efros, A. 2016. Context Encoders: Feature Learning by Inpainting. In *Proc. CVPR*.

Patraucean, V., Handa, A., and Cipolla, R. 2015. Spatio-temporal video autoencoder with differentiable memory. *ArXiv e-prints* (2015). arXiv:1511.06309.

Raiko, T., Valpola, H., and Lecun, Y. 2012. Deep Learning Made Easier by Linear Transformations in Perceptrons, In Proc. Artificial Intelligence and Statistics. *J. of Machine Learning Research* 22.

Ravishankar, S. and Bresler, Y. 2011. MR Image Reconstruction From Highly Undersampled k-Space Data by Dictionary Learning. *IEEE Trans. Med. Imaging* 30, 5 (2011), 1028–1041.

Robison, A. and Shirley, P. 2009. Image space gathering. In *Proc. High Perf. Graph.*

Ronneberger, O., Fischer, P., and Brox, T. 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* 234–241.

Rousselle, F., Manzi, M., and Zwicker, M. 2013. Robust Denoising using Feature and Color Information. *Comp. Graph. Forum* 32, 7 (2013). DOI: https://doi.org/10.1111/cgf.12219

Schmidhuber, J. 2015. Deep Learning in Neural Networks: An Overview. *Neural Networks* 61 (2015).

Schwenk, K. 2013. *Filtering techniques for low-noise previews of interactive stochastic ray tracing.* Ph.D. Dissertation. Darmstadt University.

Sen, P. and Darabi, S. 2012. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Trans. Graph.* 31, 3 (2012).

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proc. CVPR*.

Stein, C. M. 1981. Estimation of the mean of a multivariate normal distribution. In *Annals of Stat.*, Vol. 9.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *ArXiv e-prints* (2016). arXiv:1605.02688.

Veach, E. 1998. *Robust Monte Carlo Methods for Light Transport Simulation.* Ph.D. Dissertation. Advisor(s) Guibas, Leonidas J.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. 2008. Extracting and Composing Robust Features with Denoising Autoencoders. *Proc. Machine Learning* (2008).

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *Trans. Img. Proc.* 13, 4 (2004), 600–612.

Werbos, P. J. 1988. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks* 1, 4 (1988).

Yan, L.-Q., Mehta, S. U., Ramamoorthi, R., and Durand, F. 2015. Fast 4D Sheared Filtering for Interactive Rendering of Distribution Effects. *ACM Trans. Graph.* 35, 1 (2015).

Zhao, H., Gallo, O., Frosio, I., and Kautz, J. 2016. Loss Functions for Image Restoration with Neural Networks. *IEEE Transactions on Computational Imaging* (2016).

Zimmer, H., Rousselle, F., Jakob, W., Wang, O., Adler, D., Jarosz, W., Sorkine-Hornung, O., and Sorkine-Hornung, A. 2015. Path-space Motion Estimation and Decomposition for Robust Animation Filtering. *Comp. Graph. Forum* 34, 4 (2015).

Zwicker, M., Jarosz, W., Lehtinen, J., Moon, B., Ramamoorthi, R., Rousselle, F., Sen, P., Soler, C., and Yoon, S.-E. 2015. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Comp. Graph. Forum* 34, 2 (2015).