A Machine Learning Approach for Filtering Monte Carlo Noise

Nima Khademi Kalantari   Steve Bako   Pradeep Sen
University of California, Santa Barbara

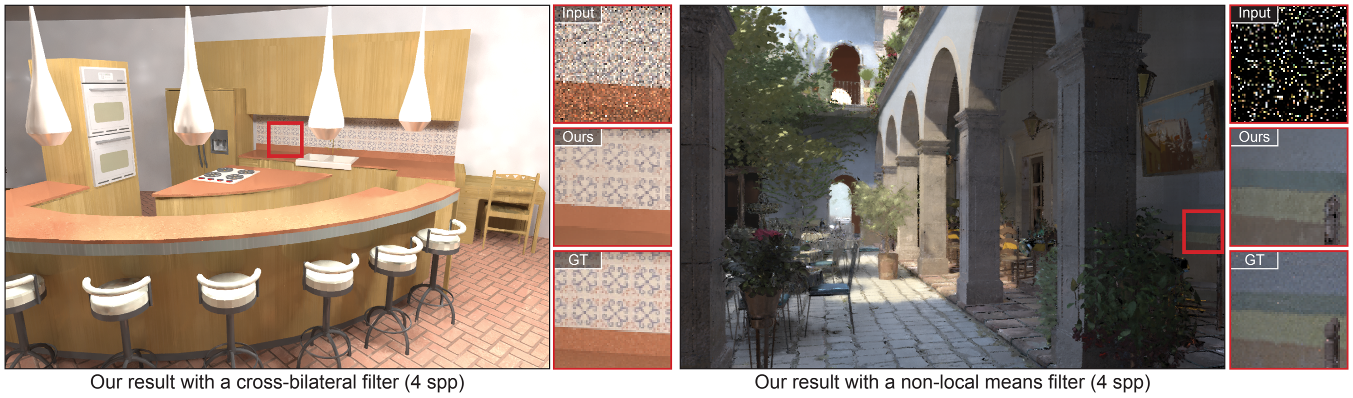

Input
Ours
GT
Input
Ours
GT
Our result with a cross-bilateral filter (4 spp)
Our result with a non-local means filter (4 spp)

Figure 1: *We propose a machine learning approach to filter Monte Carlo rendering noise as a post-process. In our method, we use a set of scenes with a variety of distributed effects to train a neural network to output correct filter parameters. We then use the trained network to drive a filter to denoise a new MC rendered image. We show the result of our approach with a cross-bilateral filter for the* KITCHEN *scene (1200×800) on the left and with a non-local means filter for the* SAN MIGUEL HALLWAY *scene (800×1200) on the right. Both of these scenes are path-traced and contain severe noise at 4 samples per pixel (spp). However, our trained network is able to estimate the appropriate filter parameters and effectively reduce the noise in only a few seconds. Note, the tonemapping of the insets has been adjusted for best visibility. Scene credits:* KITCHEN *– Jo Ann Elliott;* SAN MIGUEL HALLWAY *– Guillermo M. Leal Llaguno.*

Abstract

The most successful approaches for filtering Monte Carlo noise use feature-based filters (e.g., cross-bilateral and cross non-local means filters) that exploit additional scene features such as world positions and shading normals. However, their main challenge is finding the optimal weights for each feature in the filter to reduce noise but preserve scene detail. In this paper, we observe there is a complex relationship between the noisy scene data and the ideal filter parameters, and propose to *learn* this relationship using a nonlinear regression model. To do this, we use a multilayer perceptron neural network and combine it with a matching filter during both training and testing. To use our framework, we first train it in an offline process on a set of noisy images of scenes with a variety of distributed effects. Then at run-time, the trained network can be used to drive the filter parameters for new scenes to produce filtered images that approximate the ground truth. We demonstrate that our trained network can generate filtered images in only a few seconds that are superior to previous approaches on a wide range of distributed effects such as depth of field, motion blur, area lighting, glossy reflections, and global illumination.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

Keywords: Monte Carlo rendering, neural networks

ACM Reference Format
Kalantari, N., Bako, S., Sen, P. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. ACM Trans. Graph. 34, 4, Article 122 (August 2015), 12 pages. DOI = 10.1145/2766977 http://doi.acm.org/10.1145/2766977.

1 Introduction

Producing photorealistic images from a scene model requires computing a complex multidimensional integral of the scene function at every pixel of the image. For example, generating effects like depth of field and motion blur requires integrating over domains such as lens position and time. Monte Carlo (MC) rendering systems approximate this integral by tracing light rays (samples) in the multidimensional space to evaluate the scene function. Although an approximation to this integral can be quickly evaluated with just a few samples, the inaccuracy of this estimate relative to the true value appears as unacceptable noise in the resulting image. Since the variance of the MC estimator decreases linearly with the number of samples, many samples are required to get a reliable estimate of the integral. The high cost of computing additional rays results in lengthy render times that negatively affect the applicability of MC renderers in modern film production.

One way to mitigate this problem is to quickly render a noisy image with a few samples and then filter it as a post-process to generate an acceptable, noise-free result. This approach has been the subject of extensive research in recent years [Dammertz et al. 2010; Bauszat et al. 2011; Rousselle et al. 2012; Sen and Darabi 2012; Li et al. 2012; Rousselle et al. 2013; Moon et al. 2014]. The more successful methods typically use feature-based filters (e.g., cross-bilateral or cross non-local means filters) to leverage additional scene features such as world positions that help guide the filtering process. Since these features are highly correlated with scene detail, using them in the filtering process greatly improves the quality of the results.

Some approaches have used this information to handle specific distributed effects such as global illumination [Dammertz et al. 2010; Bauszat et al. 2011] and depth of field [Chen et al. 2011]. However, the major challenge is how to exploit this additional information to denoise *general* distributed effects, which requires setting the filter weights for all features (called *filter parameters* hereafter) so that noise is removed while scene detail is preserved. To do this, Sen and Darabi [2011; 2012] proposed to use the functional dependencies between scene features and random parameters calculated us-

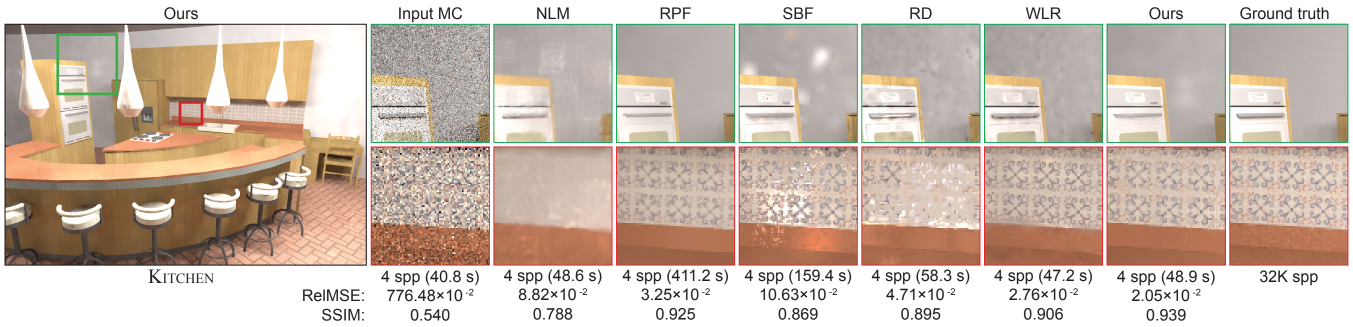


Figure 2: *Comparison between our approach and several state-of-the-art algorithms on the* KITCHEN *scene rendered at 4 spp. Note that the ground truth image is still noisy even at 32K spp. Non-local means filtering (NLM) [Rousselle et al. 2012] is a color-based method which cannot keep geometry or texture detail. Random parameter filtering (RPF) [Sen and Darabi 2012], SURE-based filtering (SBF) [Li et al. 2012], robust denoising (RD) [Rousselle et al. 2013], and weighted local regression (WLR) [Moon et al. 2014] use additional scene features (e.g., world positions) to keep the details. However, they often do not weight the importance of each feature optimally, resulting in under/over blurred regions or splotches in the final result. Our approach preserves scene detail and generates a higher-quality, noise-free result faster than most other methods. The relative mean squared error (RelMSE) and structural similarity (SSIM) index are listed below each image. Larger SSIM values indicate better perceptual quality. Full images are available in the supplementary materials.* *Scene credit: Jo Ann Elliott.*

ing mutual information, a process that removed noise but was slow. Several algorithms [Li et al. 2012; Rousselle et al. 2013; Moon et al. 2014] build upon this by using error estimation metrics to select the best filter parameters from a discrete set. The main drawback of these methods is that their error metrics are usually noisy at low sampling rates, reducing the accuracy of filter selection. Furthermore, they choose the filter parameters from a preselected, discrete set that may not contain the optimum. As a result, these methods produce images with over/under blurred regions, as shown in Fig. 2.

In this paper, we observe that there is a complex relationship between the input noisy data and the optimal filter parameters. Moreover, these filter parameters can be effectively estimated using different factors (e.g., feature variances and noise in local regions), but each individual factor by itself cannot accurately predict them. Based on these observations, we propose a *supervised learning* algorithm that learns the complex relationship between these factors and the optimal filter parameters to avoid the problems of previous approaches. To do this, we train a nonlinear regression model on a set of noisy MC rendered images and their corresponding ground truth images, using a multilayer perceptron (MLP) tightly coupled with a matching filter during both training and testing.

During the training stage, we render both the noisy images at low sampling rates as well as their corresponding ground truth images for a set of scenes with a variety of distributed effects. The noisy images are then processed to extract a set of useful features in square regions around every pixel. Finally, the proposed network is trained on these features to drive the filter to produce images that resemble the ground truth according to a specific error metric.

Once the network has been trained, we use it in the test stage to filter new noisy renderings with general distributed effects. Our method takes only a few seconds, yet produces better results than existing methods for a wide range of distributed effects including depth of field, motion blur, area lighting, glossy reflections, and global illumination. Furthermore, unlike many of the state-of-the-art approaches [Li et al. 2012; Rousselle et al. 2013], we achieve this without adaptive sampling, so our method is a simple post-process step that effectively removes MC noise. In summary, our work makes the following contributions:

- We present a machine learning approach to reduce general MC noise. Although machine learning has been used in graphics before, our approach is the first to perform supervised learning for MC noise reduction.
- Specifically, we show how a neural network can be effectively trained in combination with a filter to generate results that are close to the ground truth images.

2 Related work

Since the introduction of distributed ray tracing by Cook et al. [1984], researchers have proposed a variety of algorithms to address the noise in Monte Carlo (MC) rendering. Some of these include variance reduction techniques [Veach and Guibas 1995; Lawrence et al. 2004], low-discrepancy sampling patterns [Mitchell 1987; Shirley 1991], new Monte Carlo formulations with faster convergence [Veach and Guibas 1997; Jensen 2001], and methods that position or reuse samples based on the shape of the multidimensional integrand [Hachisuka et al. 2008; Lehtinen et al. 2011]. In this discussion, we shall only focus on primarily filtering approaches like our own, since many of these previous methods can be considered orthogonal to filtering methods. We refer readers seeking more background on MC rendering to texts on the subject [Dutré et al. 2006; Pharr and Humphreys 2010].

Filtering approaches render a noisy image with a few samples and then denoise it through a filtering process. Some methods adaptively sample as well, further improving the results. We divide the previous work on MC filtering into two general categories: **(1)** methods that only use sample color during filtering, and **(2)** methods, like our own, that use additional scene information.

Color-based filters – These methods are often inspired by traditional image denoising techniques and only use pixel color information from the rendering system to remove MC noise. Early work by Lee and Redner [1990] used nonlinear filters (median and alpha-trimmed mean filters) to remove spikes while preserving edges. Rushmeier and Ward [1994] proposed to spread the energy of input samples through variable width filter kernels. To reduce the noise in path-traced images, Jensen and Christensen [1995] separated illumination into its direct and indirect components, filtered the indirect portion, and then added the components back together. Bala et al. [2003] exploited an edge image to facilitate the filtering process, while Xu and Pattanaik [2005] used a bilateral filter [Tomasi and Manduchi 1998] to remove MC noise. Egan et al. used frequency analysis to shear a filter for specific distributed effects such as motion blur [2009] and occlusion/shadowing [2011a; 2011b], while Mehta et al. [2012; 2013; 2014] used related analysis to derive simple formulas that set the variance of a screen-space Gaussian filter to target noise from specific effects. Most of these last approaches use the analysis to adaptively position samples as well.

For denoising general distributed effects, Overbeck et al. [2009] adapted wavelet shrinkage to MC noise reduction, while Rousselle et al. [2011] selected an appropriate scale for a Gaussian filter at every pixel to minimize the reconstruction error. Rousselle et al. [2012] improved this using a non-local means filter [Buades et al. 2005]. Using the median absolute deviation to estimate the noise at every pixel, Kalantari and Sen [2013] were able to apply arbitrary image denoising techniques to MC rendering. Finally, Delbracio et al. [2014] proposed a method based on non-local means filtering which computes the distance between two patches using their color histograms. Although these color-based methods are general and work on a variety of distributed effects, they usually need many samples to produce reasonable results. At low sampling rates, they generate unsatisfactory results on challenging scenes.

Filters that use additional information – The approaches in this category leverage additional scene features (e.g., world positions, shading normals, texture values, etc.) which are computed by the MC renderer. Thus, they tend to generate higher-quality results compared to the color-based approaches mentioned above.

For example, McCool [1999] removed MC noise by using depths and normals to create a coherence map for an anisotropic diffusion filter. To efficiently render scenes with global illumination, Segovia et al. [2006] and Laine et al. [2007] used a geometry buffer. Meanwhile, to reduce global illumination noise, Dammertz et al. [2010] incorporated wavelet information into the bilateral filter and Bauszat et al. [2011] used guided image filtering. Shirley et al. [2011] used a depth buffer to handle depth of field and motion blur effects, while Chen et al. [2011] combined a depth map with sample variance to filter the noise from depth of field. Note that all of these specific methods focus on a fixed set of distributed effects and are not general.

To handle general MC noise using additional scene features, Sen and Darabi [2011; 2012] observed the need to vary the filter's feature weights across the image. Specifically, they proposed to compute these weights using mutual information to approximate the functional dependencies between scene features and the random parameters. Li et al. [2012] used Stein's unbiased risk estimator (SURE) [Stein 1981] to estimate the appropriate spatial filter parameter in a cross-bilateral filter, while hard coding the weights of the remaining cross terms. Rousselle et al. [2013] significantly improved upon this by using the SURE metric to select between three candidate cross non-local means filters that each weight color and features differently. Finally, Moon et al. [2014] compute a weighted local regression on a reduced feature space and evaluate the error for a discrete set of filter parameters to select the best one.

The main problem with such approaches, which constitute the state of the art, is that they weight each filter term through either heuristic rules and/or an error metric which is quite noisy at low sampling rates. Thus, they are not able to robustly estimate the appropriate filter weights in challenging cases, as shown in Fig. 2. In contrast, our method learns the complex relationship between the noisy input images and the ideal filter parameters in order to better leverage these additional scene features. Because we use a neural network to learn this relationship, we conclude this section with a brief summary of previous work using neural networks in computer graphics and image denoising.

Neural networks in graphics/denoising – Grzeszczuk et al. [1998] used neural networks to create physically realistic animation. Nowrouzezahrai et al. [2009] used them to predict per vertex visibility. Dachsbacher [2011] classified different visibility configurations using neural networks. Ren et al. [2013] used a neural network to model the radiance regression function to render indirect illumination of a fixed scene in real time. On the other hand, neural networks have also been used in image denoising where they have been directly trained on a set of noisy and clean patches [Burger et al. 2012]. Finally, it is worth noting that Jakob et al.'s method [2011], while not utilizing neural networks, performs learning through expectation maximization to find the appropriate parameters of a Gaussian mixture model to denoise photon maps, a different but related problem. To the best of our knowledge, our method is the first to use neural networks (or even machine learning, for that matter) for MC noise reduction.

3 A new learning framework for MC filtering

The goal of our approach is to take a noisy input image rendered with only a few samples and generate a noise-free image that is similar to the ground truth rendered with many samples. As is common with filtering approaches, the filtered image $\hat{\mathbf{c}} = \{\hat{c}_r, \hat{c}_g, \hat{c}_b\}$ at pixel i is computed as a weighted average of all of the pixels in a square neighborhood $\mathcal{N}(i)$ (e.g., 55×55) centered around pixel i:

$$\hat{\mathbf{c}}_i = \frac{\sum_{j \in \mathcal{N}(i)} d_{i,j} \bar{\mathbf{c}}_j}{\sum_{j \in \mathcal{N}(i)} d_{i,j}}, \tag{1}$$

where $d_{i,j}$ is the weight between pixel i and its neighbor j as defined by the filter, and $\bar{\mathbf{c}}_j$ is the noisy pixel color computed by averaging all the sample colors in pixel j. For example, for a standard Gaussian filter, $d_{i,j}$ would be the Gaussian-weighted distance between pixels i and j in the spatial domain. Previous work [Sen and Darabi 2012; Li et al. 2012] has used more sophisticated filters such as the cross-bilateral filter [Tomasi and Manduchi 1998], because they can leverage additional scene features (e.g., world positions, shading normals, texture values, etc.) to improve the quality of filtering. In that case, $d_{i,j}$ is given by:

$$\begin{aligned} d_{i,j} = & \exp\left[-\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{2\alpha_i^2}\right] \times \exp\left[-\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{2\beta_i^2}\right] \\ & \times \prod_{k=1}^{K} \exp\left[-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}\right], \end{aligned} \tag{2}$$

where $\bar{\mathbf{p}}_i$ and $\bar{\mathbf{f}}_{i,k}$ refer to pixel i's screen space position and scene feature k, respectively, and α_i^2, β_i^2, and $\gamma_{k,i}^2$ are the variances at pixel i for the spatial, color, and k^{th} feature terms. In this case, D and D_k are specific distance functions for colors and scene features (see Sec. 3.2). Note that here we use the cross-bilateral filter to explain our algorithm, but our framework can be extended to use other differentiable filters in a similar way (see Appendix).

Since the filter parameters α_i, β_i, and $\gamma_{k,i}$ control the widths for this filter, the challenge is how to find the values for each to produce the highest-quality results possible. In fact, a key difference between approaches such as random parameter filtering [Sen and Darabi 2012], SURE-based filtering [Li et al. 2012], and robust denoising [Rousselle et al. 2013] is how they set the filter parameters. To understand the difference between our approach and these methods, we begin by writing the filtering process more generally as:

$$\hat{\mathbf{c}}_i = h(\bar{\mathbf{s}}_{\mathcal{N}(i)}, \boldsymbol{\theta}_i), \quad \text{where} \quad \bar{\mathbf{s}}_{\mathcal{N}(i)} = \bigcup_{j \in \mathcal{N}(i)} \bar{\mathbf{s}}_j. \tag{3}$$

Here, $\bar{\mathbf{s}}_{\mathcal{N}(i)}$ is the collection of mean *primary features*[1] in the neighborhood of the i^{th} pixel, h is the filter function which implements

[1]In this paper, the term "primary features" refers to scene features that are computed directly by the rendering system when shading samples. This includes sample positions, colors, and K scene features such as world positions, shading normals, and texture values. The word "mean" refers to the fact that the features of every sample in a pixel are averaged together.

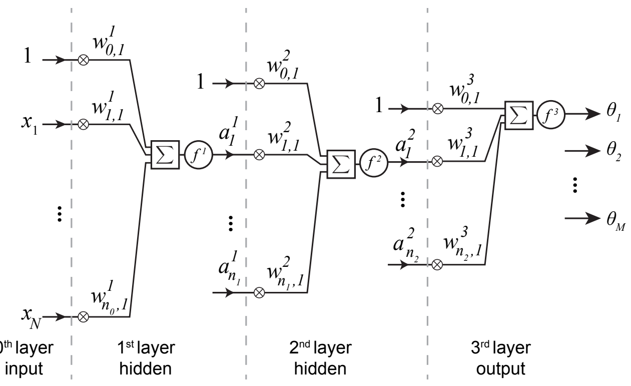


Figure 3: *A multilayer perceptron (MLP) consists of multiple layers, each with several nodes. The nodes at each layer are fully connected to the nodes in the next layer through a set of weights, where weight $w^l_{t,s}$ connects node t in layer $l-1$ to node s at layer l, and $w^l_{0,s}$ is the bias for the s^{th} node at layer l. The output of each node is computed using Eq. 7. For clarity, we only show connections for one node at each layer. In our implementation, we used a neural network with one hidden layer of size 10 to model $\mathcal{G}$ in Eq. 5.*

Eq. 1, and $\boldsymbol{\theta}_i$ is an array of M filter parameters at pixel i. For example, for the cross-bilateral filter shown in Eq. 2, $\boldsymbol{\theta}_i$ has $M = K+2$ parameters ($\boldsymbol{\theta}_i = \{\alpha, \beta, \gamma_1, \cdots, \gamma_K\}_i$) corresponding to the standard deviations for each component of the filter. In theory, the optimal filter parameters at every pixel would minimize the error between the filtered pixel value (Eq. 3) and ground truth color $\mathbf{c}_i$:

$$\boldsymbol{\theta}^*_i = \arg\min_{\boldsymbol{\theta}_i} E(h(\bar{\mathbf{s}}_{\mathcal{N}(i)}, \boldsymbol{\theta}_i), \mathbf{c}_i), \tag{4}$$

where E is an error metric (e.g., mean squared error). To find the approximate filter parameters, $\hat{\boldsymbol{\theta}}_i$, that estimate the optimum ones, $\boldsymbol{\theta}^*_i$, the noisy mean primary features in a pixel's neighborhood can be processed to generate a set of more meaningful data, which we call *secondary features* $\mathbf{x}_i = \{x_1, x_2, \cdots, x_N\}_i$. These secondary features include feature variances, noise approximation in local regions, and so on. We then can approximate the filter parameters through a function $\mathcal{G}$ of the secondary features:

$$\hat{\boldsymbol{\theta}}_i = \mathcal{G}(\mathbf{x}_i). \tag{5}$$

We observe that function $\mathcal{G}$, which attempts to approximate the relationship between the secondary features and the optimal filter parameters, is complicated and difficult to model explicitly. For this reason, we propose to find this function with a learning system that performs the following energy minimization on training data:

$$\mathcal{G}^* = \arg\min_{\mathcal{G}} E(h(\bar{\mathbf{s}}_{\mathcal{N}(i)}, \mathcal{G}(\mathbf{x}_i)), \mathbf{c}_i). \tag{6}$$

Once we minimize this energy function to find approximate $\hat{\mathcal{G}}$, given a new test scene we can compute the filter parameters $\hat{\boldsymbol{\theta}}_i = \hat{\mathcal{G}}(\mathbf{x}_i)$ that will be used in the filter to generate a filtered image that is close to the ground truth.

This framework allows us to observe that many of the previous MC filtering approaches can be classified into two categories: 1) methods that perform the minimization in Eq. 4 using an error estimation metric, and 2) algorithms that attempt to model $\mathcal{G}$ directly. Error minimization approaches [Li et al. 2012; Rousselle et al. 2013; Moon et al. 2014] rely on an error estimate (e.g., SURE) to select filter parameters, but this estimate is often noisy at low sampling rates which negatively affects the accuracy of the filter selection. Furthermore, they still have to do a search through the parameter space at run-time to minimize Eq. 4, which is often done by looping through a discrete set of parameters.

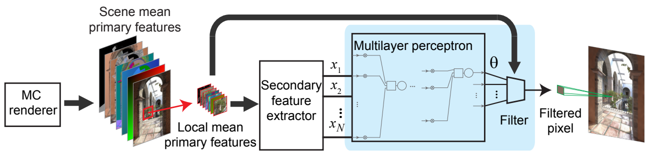

Scene mean primary features
MC renderer
Local mean primary features
Secondary feature extractor
x_1
x_2
x_N
Multilayer perceptron
θ
Filter
Filtered pixel

Figure 4: *Our approach combines a standard MLP (Fig. 3) with a matching filter. The local mean primary features (illustrated by a stack of images) contain color, position, and additional features such as world positions, shading normals, etc. A set of secondary features $\{x_1, \cdots, x_N\}_i$ (see Sec. 3.3) are extracted from the mean primary features in a local neighborhood of each pixel. The MLP takes the secondary features and outputs the parameters of the filter. The filter then takes the block of mean primary features and outputs a filtered pixel. During training, we minimize the error between the filtered pixel and the ground truth. Once trained, the network can generate appropriate filter parameters for an arbitrary test image.*

On the other hand, approaches in the second category (e.g., Sen and Darabi [2012]) avoid error estimation by trying to model $\mathcal{G}$ directly, but the underlying relationship is complex and difficult to express explicitly. We avoid all of these problems by approximating $\mathcal{G}$ with a learning system that directly minimizes the error in Eq. 6. Specifically, we represent $\mathcal{G}$ using a neural network (see Fig. 3) and directly combine it with a matching filter during training and testing (see Fig. 4). Note that since the ground truth images are available during training, we can directly compute the error between the filtered and ground truth image without need for error estimation. At test time, we then use the trained $\hat{\mathcal{G}}$ on the secondary features from *new* scenes to compute filter parameters that will allow the filter to produce results close to the ground truth.

We now describe how to train a neural network in combination with the filter by minimizing the energy function from Eq. 6.

3.1 Our proposed neural network

As with any machine learning system, there are three crucial elements we must decide: **(1)** the choice of model for representing $\mathcal{G}$, **(2)** an appropriate error metric E to measure the distance between the filtered and ground truth images, and **(3)** an optimization strategy to minimize the energy function in Eq. 6.

For our representation, we use a neural network, formally a multilayer perceptron (MLP), as a regression model since it is a simple, yet powerful system for discovering complex nonlinear relationships between its inputs and outputs. Moreover, MLPs are inherently parallel and can be efficiently implemented on a GPU. However, our proposed approach differs from standard MLPs in that we have incorporated the filter into the training process, which requires us to "backpropagate" through it to update the weights of the network during training. This implies that the filter must be differentiable with respect to its filter parameters. Fortunately, common filters such as the Gaussian, cross-bilateral, and cross non-local means filters are all differentiable and can be applied in our system.

As shown in Fig. 3, MLPs consist of multiple layers known as the input, hidden, and output layers. Each layer has several nodes which are fully connected to all nodes in the next layer through weights. The output of a certain node is a function of the weighted sum of the outputs of the nodes from the previous layer plus an additional bias term used as an offset. Specifically, the output of the s^{th} node at the l^{th} layer is given by:

$$a^l_s = f^l\left(\sum_{t=1}^{n_{l-1}} w^l_{t,s} a_t^{\,l-1} + w^l_{0,s}\right), \tag{7}$$

where n_{l-1} is the number of nodes in layer $l-1$, $w^l_{t,s}$ is the weight associated with the connection between node t in layer $l-1$ and node s in layer l, $w^l_{0,s}$ is the bias for this node, and f^l is the activation function for layer l. If we use linear activation functions (e.g., $f^l(x) = x$) in all layers, the neural network becomes a simple linear regression model. Therefore, we typically use nonlinear activation functions, such as the sigmoid function $f^l(x) = 1/(1+e^{-x})$. See Sec. 4.1 for a complete description of the actual MLP architecture we used in this paper.

Next, we describe our metric to measure the error between the filtered and ground truth pixel values. Rather than using standard MSE, we use the relative mean squared error (RelMSE) metric proposed by Rousselle et al. [2011] which is better suited for our application, since the human visual system is more sensitive to color variations in darker regions of the image. However, we modify relative MSE slightly to get:

$$E_i = \frac{n}{2} \sum_{q \in \{r,g,b\}} \frac{(\hat{c}_{i,q} - c_{i,q})^2}{c^2_{i,q} + \varepsilon}, \tag{8}$$

where n is the number of samples per pixel, $\hat{c}_{i,q}$ and $c_{i,q}$ are the q^{th} color channel of the filtered and ground truth pixels, respectively, and ε is a small number (0.01 in our implementation) to avoid division by zero. Here, the division by $c^2_{i,q}$ accounts for the aforementioned perceptual effect by giving higher weight to the regions where the ground truth image is darker.

Our modification to Rousselle et al.'s original relative MSE metric is the multiplication by $n/2$. Since the squared error decreases linearly with the number of samples in MC rendering, images with fewer samples have greater error than images with more samples. This would make our learning system biased in favor of training images with lower sampling rates. By multiplying the squared error by n, we cancel this inverse relationship and force all of the images to have an equal contribution to the error regardless of their sampling rate. Furthermore, the error is divided by 2 to produce a simpler derivative (see Eq. 10). Note that our proposed metric is different than the commonly used error metrics in the neural network community, such as mean squared error (MSE) and cross-entropy error.

Finally, we describe our optimization strategy to train the network. To do this, we first need a large set of input data and the corresponding ground truth images, which can be generated prior to training. For each noisy image, we extract a set of secondary features at each pixel (see Section 3.3). With this data in hand, we can then train the network through an iterative, three-step process called *backpropagation* [Rumelhart et al. 1986]. The goal of this process is to find the optimal weights $w^l_{t,s}$ for all nodes in the network which minimize the error between the computed and desired outputs (i.e., ground truth values) for all pixels in the training images, $E = \sum_{i \in \text{all pixels}} E_i$.

Before starting the backpropagation process, the weights are randomly initialized to small values around zero (between -0.5 to 0.5 in our implementation). Then in the first step, known as the *feed-forward* pass, the output of the network is computed using all inputs. This can be implemented efficiently using a series of matrix multiplications and activation functions applied to the input data to evaluate Eq. 7. In the second step, the error between the computed and desired outputs is used to determine the effect of each weight on the output error. This requires taking the derivative of the error with respect to each weight $\partial E / \partial w^l_{t,s}$. Thus, the activation functions (and in our case, the filter as well) need to be differentiable. These two steps are performed for all of the data in the training set and the error gradient of each weight is accumulated. Finally, in the third step, all the weights are updated according to their error gradient. This completes a single iteration of training, known as an *epoch*, and typically many epochs are needed to properly train the system and obtain a converged set of weights.

We now examine the differentiability of our system as required by backpropagation. Using the chain rule, we express the derivative of the energy function from Eq. 8 with respect to the weights $w^l_{t,s}$ as:

$$\frac{\partial E_i}{\partial w^l_{t,s}} = \sum_{m=1}^{M} \left[\sum_{q \in \{r,g,b\}} \left[\frac{\partial E_{i,q}}{\partial \hat{c}_{i,q}} \frac{\partial \hat{c}_{i,q}}{\partial \theta_{m,i}} \right] \frac{\partial \theta_{m,i}}{\partial w^l_{t,s}} \right], \tag{9}$$

where M is the number of filter parameters. The first term is the derivative of the error with respect to the filtered pixels $\hat{c}_{i,q}$, which can be easily calculated as follows:

$$\frac{\partial E_i}{\partial \hat{c}_{i,q}} = n \frac{\hat{c}_{i,q} - c_{i,q}}{c^2_{i,q} + \epsilon}. \tag{10}$$

In addition, the $\theta_{m,i}$'s are the outputs of a standard MLP network (see Fig. 3), so the last term in Eq. 9 can be analytically calculated as usual [Rumelhart et al. 1986]. Finally, the middle term requires that our filter be differentiable so we can compute the derivative of the filtered color with respect to the filter parameters $\partial h_q(\bar{\mathbf{s}}_{\mathcal{N}(i)}, \boldsymbol{\theta}_i)/\partial \theta_{m,i}$. Fortunately, as shown in the Appendix, the common filters used for MC denoising such as the cross-bilateral [Li et al. 2012] and cross non-local means [Rousselle et al. 2013] filters are differentiable.

To summarize the training process, we compute the derivative in Eq. 9 for each weight in the network, and update the weights after every epoch. This iterative process continues until convergence.

3.2 Primary features

Primary features are those directly output by the rendering system. We use 7 primary features ($M = 7$) in our cross-bilateral filter (Eq. 2), consisting of screen position, color, and five additional features ($K = 5$): **(1)** world position, **(2)** shading normal, **(3,4)** texture values for the first and second intersections, and **(5)** direct illumination visibility. This last feature was also used by Rousselle et al. [2013] and we found it to be useful in our system.

During rendering, for each sample we output screen position in x and y, color in RGB format, world position in Cartesian coordinates (x, y, and z), shading normal (i, j, and k), texture values for the first and second intersections in RGB format, and a single binary value for the direct illumination visibility, for a total of 18 floating point values. We then average these values for all samples in a pixel to produce the mean primary features for every pixel in the image. Note that at this point, the average direct illumination visibility represents the fraction of shadow rays that see the light and is not a binary value. Moreover, as was done in Rousselle et al. [2013], we prefilter the additional features using a non-local means filter in an 11×11 window with patch size 7×7.

Similar to recent approaches [Li et al. 2012; Rousselle et al. 2013], we normalize the distance of the color and additional features by their variances. We use the following function for the color term:

$$D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j) = \frac{\|\bar{\mathbf{c}}_i - \bar{\mathbf{c}}_j\|^2}{\psi_i^2 + \psi_j^2 + \zeta}, \tag{11}$$

where ψ_i and ψ_j are the standard deviation of color samples at pixel i and j, respectively, and ζ is a small number (10^{-10}) to avoid division by zero. For the additional features, we use:

$$D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k}) = \frac{\|\bar{\mathbf{f}}_{i,k} - \bar{\mathbf{f}}_{j,k}\|^2}{\max(\psi^2_{k,i}, \delta)}, \tag{12}$$

where $\psi_{k,i}$ is the standard deviation of the k^{th} feature at pixel i and δ is a small number (10^{-4}) to avoid division by zero. These parameters are all similar to those of Rousselle et al. [2013]. On a final note, we smooth the noisy standard deviations for the additional features, $\psi_{k,i}$, by filtering them using the same weights computed by the non-local means filter when filtering the primary features.

3.3 Secondary features

At every pixel, we compute a set of secondary features from the neighboring noisy samples to serve as inputs to our neural network. Most features are inspired by (or are direct implementations of) those used in previous algorithms for MC denoising.

Feature statistics: We compute the mean and standard deviation for the $K = 5$ additional features of all samples in the pixel. To capture more global statistics, we also calculate the mean and standard deviation of the pixel-averaged features in a 7×7 block around each pixel. We compute the statistics for each component (e.g., i, j, k for shading normal) separately and average them together to get a single value per feature. Thus, we have 20 total values for each pixel and the block around it.

Gradients: The gradients of features have been previously used to decrease the weight of a feature in regions with sharp edges [Rousselle et al. 2013]. We calculate the gradient magnitude (scalar) of the K additional features using a Sobel operator (5 values total).

Mean deviation: This term is the average of the absolute difference between each individual pixel in a block and the block mean. A normalized version of this metric was proposed by Hachisuka et al. [2008] as an error metric for adaptive sampling. This feature can help identify regions with large error so the network can adjust the filter parameters accordingly. For each of the K additional features, we compute the mean deviation of all the pixel-averaged features in a 3×3 block around each pixel. As with feature statistics, this is computed on each component separately and then averaged to obtain a single value for each additional feature (5 values total).

MAD: We use the median absolute deviation (MAD) used by Kalantari and Sen [2013] to estimate the amount of noise in each pixel, which is directly related to the size of the filter. We compute MAD for each K additional features (5 values total).

Sampling rate: Finally, we use the inverse of the sampling rate. The variance of MC noise decreases linearly with the number of samples and, therefore, the filter parameters should reflect this. Since we only train a single network, it should be able to handle different sampling rates and adjust the filter size accordingly.

In summary, we compute a total of $N = 36$ secondary features for each pixel. The network takes these features as input and outputs the parameters to be used by the filter to generate the final filtered pixel. This is done for all the pixels to produce the final result.

4 Implementation Details

4.1 Neural network structure

The complexity of a network is determined by the number of layers and the node count of each. Although a more complex network containing several layers with many nodes can model more complicated relationships between its inputs and outputs, training such networks is difficult and there is an increased likelihood of overfitting to the training data. In these cases, the network could model the noise rather than the underlying relationships and, therefore, would perform poorly on actual test data. We empirically found that one hidden layer of size 10 provides the best performance in general (see Sec. 6). The input layer consists of 36 nodes corresponding to our secondary features, while our output layer has one node for each filter parameter (6 total). Note that we use a fixed color weight, β_i, and, thus, our output layer only requires 6 nodes (see Sec. 4.2).

For the hidden layer, we chose the sigmoid activation function $f(x) = 1/[1 + \exp(-x)]$ and for the output layer, we used the softplus function $f(x) = \log(1 + e^x)$, a smooth differentiable approximation to the rectifier function. The output of the softplus function is always positive $(0, \infty)$, making it the ideal candidate to output the filter parameters in the final layer of the MLP network.

4.2 Training

As explained in Section 3, training is done using the iterative backpropagation algorithm [Rumelhart et al. 1986]. At every iteration, all of the weights in the network are updated based on the error gradient with respect to the weights. We used the resilient backpropagation (RPROP) method [Riedmiller and Braun 1993] to update the weights, rather than the commonly used gradient descent. In RPROP, the weights are updated based on the signs of the gradients, rather than their magnitudes. At every iteration of the backpropagation process, the gradients of a particular weight are rewarded if the current and previous iterations are consistent (have the same sign) and penalized otherwise. We found this method results in faster convergence for our system compared to gradient descent. To further speed up convergence, we normalize the network features by subtracting the mean and dividing by the standard deviation of the training set, as is frequently done in neural networks [Le Cun et al. 1998]. Finally, we observed that, in practice, optimizing the filter parameter for the color term of the cross-bilateral filter results in overfitting. We experimentally found that setting this parameter to a fixed value, $\beta_i = 7$, is sufficient to keep the necessary scene details while removing noise.

In our training set, we used 20 scenes with different distributed effects such as depth of field, motion blur, and global illumination (see Fig. 7). For each scene, we rendered the ground truth image with a large number of samples (e.g., 32K). To handle scenes with different sampling rates, we rendered each training scene with 4, 8, 16, 32, and 64 samples per pixel and trained all of the images on a single network. Moreover, to avoid overfitting to one particular noise pattern, we rendered each scene five times at each sampling rate (a total of 25 images for each scene). To train the network on all of these images efficiently, we performed mini-batch training [Hastie et al. 2009], where each iteration of the backpropagation process is performed on a subset of the training set. Specifically, we compute the derivative from Eq. 9 for each pixel across all of the images in the subset. We then average all of the gradients and use the result to update the weights. In our implementation, our subset consisted of one image at each sampling rate from all of the training scenes. Furthermore, we alternated between the five noisy examples at each iteration.

4.3 Handling high dynamic range spikes

MC rendered images have high dynamic range and could contain spikes, pixels with radiance values orders of magnitude greater than their neighbors. Usually these high energy pixels cannot be spread efficiently, even with a big filter, and, thus, need to be handled differently. After applying the filter, we identify these spikes in the filtered result by first calculating the mean and standard deviation of the RGB colors for all the neighboring pixels in a block of size 3×3. If any color channel of the center pixel has values more than 2 standard deviations away from the block average, then it is labeled as a spike and replaced with the median block color. Fig. 5 shows how spike removal affects KITCHEN, a test scene. As can

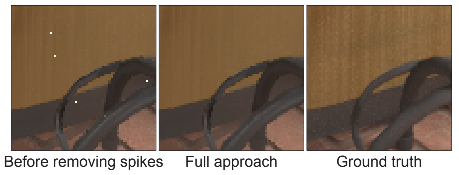


Figure 5: *The image on the left shows the result of our approach before spike removal on an inset of the* KITCHEN *scene. In our method, we remove high magnitude spikes in the filtered image as a post-process to produce the result shown in the middle. The ground truth image is shown on the right for comparison.*

be seen, our system does not heavily rely on spike removal and can spread the energy of most of the high magnitude pixels itself. Although this spike removal process is not differentiable, we found that it does not aversely affect the training. A better way of handling spikes within our framework is left for future investigation.

5 Results

We implemented our algorithm, called learning-based filtering (LBF), in C++ and integrated it into PBRT2 [Pharr and Humphreys 2010]. The neural network and filter were written in CUDA for GPU acceleration. All timings were obtained on an Intel quad-core 3.7 GHz machine with 24GB of memory and a GeForce GTX TITAN GPU. We used uniform low-discrepancy samples to generate the noisy images for both the training and test sets. Note that we trained a single network on scenes with 4, 8, 16, 32, and 64 samples per pixel to produce all the results in this paper. Moreover, we used a cross-bilateral filter (of size 55×55) unless otherwise noted.

We begin by evaluating the robustness of our learning system to the training set. To do this, we randomly selected 8 test scenes from a pool of 42 scenes spanning a variety of distributed effects. We then trained four different networks on random training sets composed of 16 randomly selected scenes from the remaining 34. To compare the convergence of these four networks, we evaluated the progress of each network during training by computing the average error (see Eq. 8) on the 8 test scenes at every epoch. As seen in Fig. 6, all four networks converge to approximately the same error, showing that our learning system is fairly robust to the choice of training set.

For the main comparisons, we trained our network on 20 scenes with a variety of Monte Carlo effects (see Fig. 7). As the robustness test in Fig. 6 shows, the network typically converges after approximately 25 epochs. However, to ensure that the network was fully converged, we performed 50 epochs which took approximately 9 hours, and then used these converged weights to generate the results for comparisons. Note that all results shown in this paper (with the exception of Fig. 7) are produced on test scenes that were not part of our training set.

We compare our method against state-of-the-art approaches including non-local means filtering (NLM) [Rousselle et al. 2012], random parameter filtering (RPF) [Sen and Darabi 2012], SURE-based filtering (SBF) [Li et al. 2012], robust denoising (RD) [Rousselle et al. 2013], and weighted local regression (WLR) [Moon et al. 2014]. Note that with the exception of RPF, all the other algorithms are significantly benefiting from adaptive sampling in these comparisons, while our approach is not adaptive. We also compare against adaptive sampling and reconstruction (ASR) [Rousselle et al. 2011] and ray histogram fusion (RHF) [Delbracio et al. 2014] in the supplementary materials.

For all algorithms, we used the implementations provided by the authors with their default parameters. For RD, we used a window size of 20 for higher quality, as suggested by the authors [Rousselle et al.

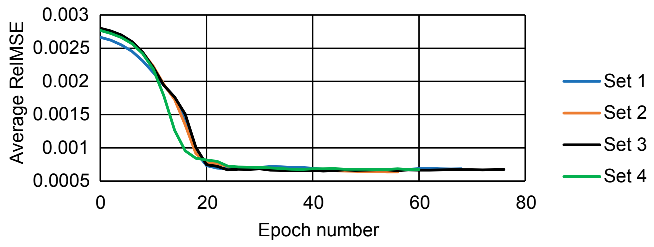


Figure 6: *We evaluate the robustness of our learning system to the choice of training set by training four different networks on 16 randomly selected scenes drawn from a pool of 34. We plot the average relative MSE of each network for a set of 8 test scenes at each epoch of the training. As can be seen, all four networks converge to roughly the same test error which shows the robustness of our approach to the training set. The images used in each set as well as the test images are available in the supplementary materials.*

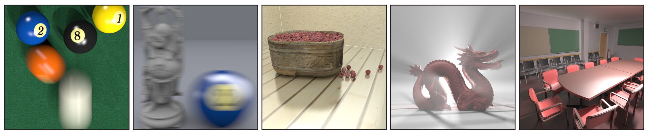


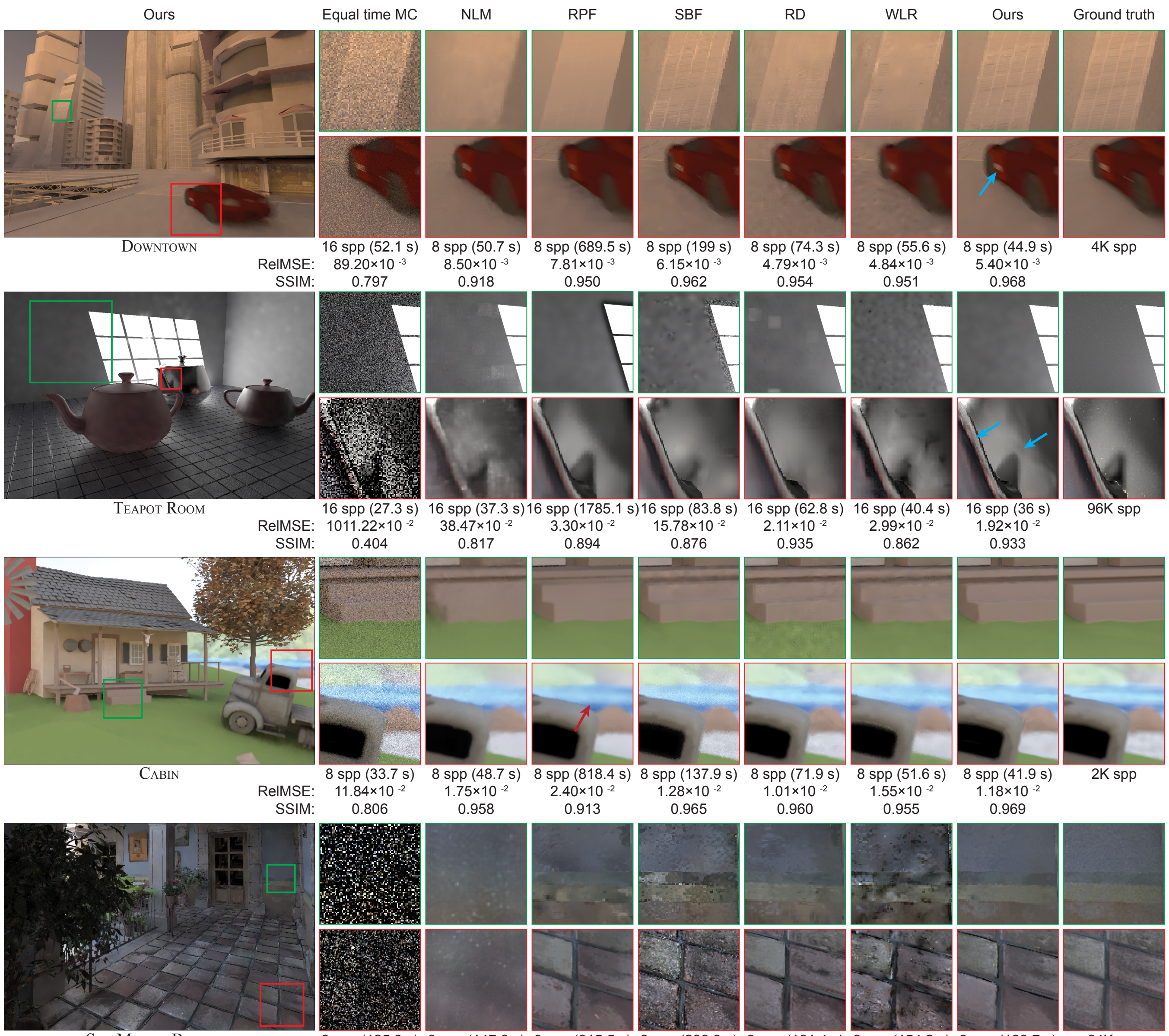
Figure 7: *A subset of our training scenes (cropped for best fit). These images have different distributed effects including depth of field, motion blur, glossy reflections, and global illumination. All 20 training images are available in the supplementary materials.* *Scene credits:* POOLBALL *– Toshiya Hachisuka;* BUDDHA, DRAGONS *– Stanford 3D Scanning Repository;* RASPBERRIES *– turbosquid.com user sirin_3d (raspberries), openfootage.net (wall/floor textures);* CONFERENCE *– Anat Grynberg and Greg Ward.*

2013]. For quantitative comparisons, we calculated relative MSE as proposed by Rousselle et al. [2011]: $\text{RelMSE} = (\hat{c} - c)^2/(c^2 + \varepsilon)$ averaged over all pixels, where $\hat{c}$ and c are the filtered and ground truth pixel colors, respectively, and ε is a small number (0.01) to avoid division by zero. We also measure the perceptual quality of the results using the structural similarity (SSIM) index [Wang et al. 2004] which is a value from 0 to 1, where 1 indicates perfect quality with respect to the ground truth image.

Fig. 8 compares our approach to state-of-the-art methods on four scenes with different distributed effects. Although our algorithm is faster than all other methods shown here, we perform equal-sample comparisons to ensure fairness (please refer to the supplementary materials for additional equal-time comparisons for some scenes). For completeness, we also show the approximate results of equal-time Monte Carlo, since the low-discrepancy sampling in PBRT2 requires a power-of-two sampling rate.

First, we examine the path-traced DOWNTOWN scene with global illumination and motion blur. NLM uses only color information when filtering, so it cannot preserve the geometry and texture details in the scene. Moreover, although the other approaches use additional features, they often do not have appropriate filter weights, resulting in either overblurred textures or residual motion blur noise. Our approach is not only the fastest but it also preserves the textures while removing the motion blur noise, resulting in an image with fewer artifacts than the other techniques. Note, for example, that only our approach is able to preserve the thin lines between the stripes on the car (indicated by the blue arrow).

The TEAPOT ROOM scene is a challenging, path-traced scene containing one glossy and two diffuse teapots with mostly indirect illumination. None of the other methods can effectively remove the strong indirect illumination noise on the back wall without introducing artifacts or overblurring the glossy reflections on the body and spout of the teapot. Note the ground truth image still has visible spikes at 96K spp, while we produce a relatively noise-free result.



	Equal time MC	NLM	RPF	SBF	RD	WLR	Ours	Ground truth
DOWNTOWN	16 spp (52.1 s)	8 spp (50.7 s)	8 spp (689.5 s)	8 spp (199 s)	8 spp (74.3 s)	8 spp (55.6 s)	8 spp (44.9 s)	4K spp
RelMSE:	89.20×10^{-3}	8.50×10^{-3}	7.81×10^{-3}	6.15×10^{-3}	4.79×10^{-3}	4.84×10^{-3}	5.40×10^{-3}	
SSIM:	0.797	0.918	0.950	0.962	0.954	0.951	0.968	
TEAPOT ROOM	16 spp (27.3 s)	16 spp (37.3 s)	16 spp (1785.1 s)	16 spp (83.8 s)	16 spp (62.8 s)	16 spp (40.4 s)	16 spp (36 s)	96K spp
RelMSE:	1011.22×10^{-2}	38.47×10^{-2}	3.30×10^{-2}	15.78×10^{-2}	2.11×10^{-2}	2.99×10^{-2}	1.92×10^{-2}	
SSIM:	0.404	0.817	0.894	0.876	0.935	0.862	0.933	
CABIN	8 spp (33.7 s)	8 spp (48.7 s)	8 spp (818.4 s)	8 spp (137.9 s)	8 spp (71.9 s)	8 spp (51.6 s)	8 spp (41.9 s)	2K spp
RelMSE:	11.84×10^{-2}	1.75×10^{-2}	2.40×10^{-2}	1.28×10^{-2}	1.01×10^{-2}	1.55×10^{-2}	1.18×10^{-2}	
SSIM:	0.806	0.958	0.913	0.965	0.960	0.955	0.969	
SAN MIGUEL BALCONY	8 spp (125.3 s)	8 spp (147.6 s)	8 spp (815.5 s)	8 spp (299.3 s)	8 spp (161.4 s)	8 spp (154.2 s)	8 spp (133.7 s)	64K spp
RelMSE:	121.2×10^{-2}	3.07×10^{-2}	1.72×10^{-2}	8.16×10^{-2}	1.61×10^{-2}	4.05×10^{-2}	1.92×10^{-2}	
SSIM:	0.458	0.754	0.892	0.841	0.883	0.781	0.899	

Figure 8: *We compare our algorithm against several state-of-the-art techniques as well as equal-time Monte Carlo. Our algorithm is faster than all these filtering approaches, but we perform equal-sample comparisons to guarantee fairness. Note that since the low-discrepancy sampling pattern in PBRT2 requires a power-of-two sampling rate, we could not match timings exactly for equal-time MC. The relative MSE and SSIM index values are listed below each image (for SSIM higher is better). Tonemapping of the insets have been adjusted equally for all algorithms for best visibility. Full images are available in the supplementary materials.* *Scene credits:* DOWNTOWN *– Herminio Nieves (buildings), tf3dm.com user ysup12 (car);* TEAPOT ROOM *– Martin Newell;* CABIN *– Andrew Kin Fun Chan and Dan Konieczka (cabin, mountain, tree), tf3dm.com user 3dregenerator (truck);* SAN MIGUEL BALCONY *– Guillermo M. Leal Llaguno.*

The CABIN scene is a path-traced scene that includes global illumination and depth of field. NLM and RPF remove the depth of field noise, but overblur the geometry of the steps and texture in the water. SBF and WLR do not weight the features appropriately and thus their results contain residual noise in the depth of field regions. Although both our approach and RD handle the depth of field regions well, RD produces visible artifacts in the smooth regions due to the low sampling rate. Meanwhile, we are able to generate a smooth result that is closer to ground truth.

Finally, SAN MIGUEL BALCONY is a path-traced scene with severe noise at 8 spp. Again, NLM overblurs the textures on the floor and the wall. Moreover, SBF and WLR produce results that are overblurred or contain residual noise. Although RPF, RD, and our method preserve the floor texture, RD and RPF cannot properly remove the noise in the smooth regions, including the back wall. Despite having a slightly higher relative MSE than RPF and RD, we produce a relatively noise-free result that is better than the other algorithms both visually and in terms of SSIM.

To test the convergence of our method as the sampling rate increases, we provide convergence plots in Fig. 9 using relative MSE and SSIM metrics for the DOWNTOWN scene and provide comparisons against low-discrepancy samples, NLM, SBF, RD, and WLR.

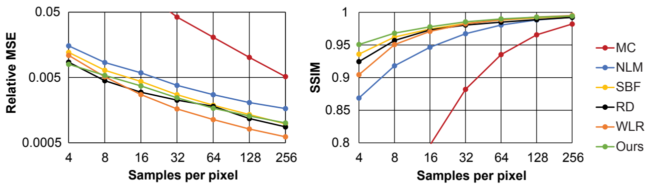


Figure 9: *On the left, we show the convergence plot in terms of relative MSE for the* DOWNTOWN *scene with our approach and several state-of-the-art approaches. Our relative MSE consistently decreases despite the fact that the network has not been trained on sampling rates higher than 64 samples per pixel. We show the convergence plot in terms of SSIM index on the right. Our method produces results with higher perceptual quality, particularly at low sampling rates. Note that although WLR has strong relative MSE, the perceptual quality of its results is lower than SBF, RD, and our approach because of visible artifacts. Also, our algorithm is faster than the others, so equal number of samples is not equal time.*

Since RPF is computationally expensive, we omit its comparison. As can be seen, our relative MSE consistently decreases even at the sampling rates that the network has not been trained on, demonstrating the network's ability to extrapolate. Although higher sampling rates could be included in the training set, we found our set to be sufficient for the purposes of this paper. Moreover, the perceptual quality of our method is higher than others based on SSIM index, particularly at lower sampling rates. Our method is a consistent estimator because the distances in the color term are normalized by the variances. Thus, as the sampling rate increases, these variances are reduced and the filter approaches identity.

We also demonstrate how other filters can be used in our framework by training with the cross non-local means filter [Li et al. 2012; Rousselle et al. 2013]. Since every filter has a different relationship between the secondary features and its filter parameters, we cannot simply use the network weights from the previous cross bilateral filter. In Fig. 10, we compare our method against NLM, SBF, and RD (all with non-local means filters) and show that our method produces a smoother result and preserves most of the scene detail.

Finally, to handle animated sequences, we simply use the same network from before *without retraining* and extend our cross-bilateral filter to operate on 3-D spatio-temporal volumes. This modification to the filter is necessary to reduce the flickering that might appear if we filter each frame independently. We tested our extension on the SCI-FI CITY, KITCHEN, SIBENIK, and SAN MIGUEL video sequences, which can be found in the accompanying video. In all cases, we used only three neighboring frames on each side of the current frame (7 frames total) for spatio-temporal filtering. Our approach is able to generate high-quality, temporally-coherent videos from noisy input sequences with low sampling rates.

In terms of timing, the overhead of our algorithm is roughly independent of the sampling rate. In general, our algorithm with a cross-bilateral filter removes noise from a 1200×800 image in roughly 8.4 seconds, where calculating the features takes approximately 5 seconds and evaluating the network and filtering takes 3.4 seconds. For video sequences, the spatio-temporal filter increases our timings to roughly one minute for each 1200×800 frame, which is reasonable given the quality of our videos.

6 Discussion, limitations, and future work

In our method, we estimate $\mathcal{G}$ by training a network through direct minimization of the error between the filtered and ground truth images (Eq. 6). An alternative approach might be to approximate $\mathcal{G}$ by minimizing the error between the estimated and ground truth

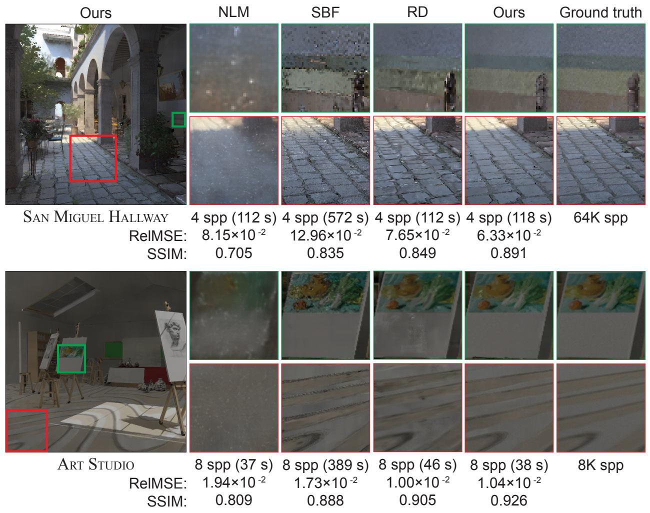


Figure 10: *Comparison between our approach and three other methods all using non-local means filters. In both scenes, NLM overblurs structure and textures while SBF and RD have noticeable artifacts in both smooth and textured regions.* *Scene credits:* SAN MIGUEL HALLWAY – *Guillermo M. Leal Llaguno;* ART STUDIO – *Giorgio Luciano.*

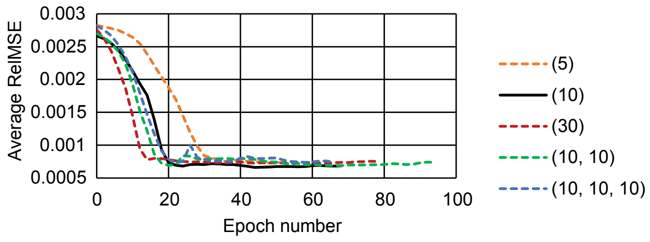


Figure 11: *We evaluate the effect of network size on learning performance. Here, we train 5 networks of different sizes and compute the average relative MSE on 8 test images at every epoch. The sizes differ in the number of hidden layers and the number of nodes in each layer. For example, (10, 10) is a network with 2 hidden layers and 10 nodes in each layer. We found that the network with one hidden layer of size 10 provides slightly better performance than the other sizes. The sizes of the input and output layers, which correspond to the number of secondary features and the number of output filter parameters, respectively, are both fixed.*

filter parameters, i.e., $E(\mathcal{G}(\mathbf{x}_i), \boldsymbol{\theta}_i^*)$. However, this approach has two major problems. First, it provides a suboptimal solution to $\mathcal{G}$ since it does not minimize the error between the filtered and ground truth images, which is our final goal. In some cases, a small error in filter parameters might result in a large error in the final image. Second, it requires calculating the optimum filter parameters $\boldsymbol{\theta}_i^*$ for each pixel in every image of the training set. This requires an expensive, brute-force search in the parameter space to perform the error minimization in Eq. 4.

As discussed in Sec. 4.1, network size affects performance and thus it is important to size it appropriately. We evaluate this effect by training five networks of different sizes on the first training set of the robustness experiment from Fig. 6. Fig. 11 shows convergence plots for the different network sizes based on the average relative MSE of 8 test scenes (also from the robustness experiment) evaluated after each epoch. Since the network with one hidden layer of size 10 produces slightly better results than the other configurations, we use it for all the results in this paper.

The foundation of our approach lies in having an effective training stage that can generate high-quality results on the test scenes.

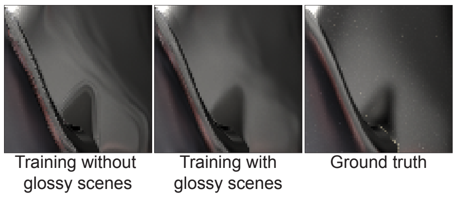


Figure 12: *We show the effect of training on scenes without glossy reflections on the* TEAPOT ROOM *test scene. Even without glossy scenes in the training set, our method is able to capture the glossy reflections on the teapot. However, the result has fewer artifacts when the glossy effect is present in the training set.*

Therefore, the training set should include scenes with a variety of distributed effects so that the network can learn how to robustly handle diverse test scenes. To evaluate the effect of a particular distributed effect, we remove all scenes with glossy reflections from the training set used in Fig. 8 and replace them with non-glossy scenes containing other distributed effects. We show the result of the new trained network on the TEAPOT ROOM test scene in Fig. 12. Although the glossy reflections are preserved better than most other approaches (see Fig. 8), the result has lower quality when compared to the network trained with scenes containing glossy effects.

Our network uses various secondary features to estimate filter parameters and perform filtering. We evaluated the impact of the secondary features in each of the 5 categories defined in Section 3.3 by excluding all the features from each category one at a time and training the network. Based on this experiment, we observed that feature statistics have the most impact on the quality of the result while removing the other features only slightly reduced the quality. However, since these other features are fast to compute, we used them all in our framework to have improved quality.

Like many other MC filtering approaches, the main limitation of our algorithm is that it produces biased but consistent results. As such, it may not be an adequate solution for applications requiring complete physical accuracy. However, we see strong potential in fields that focus more on perceptual quality such as digital film production, particularly for previsualization when quick rendering times are important and very low sampling rates are used. However, as shown in Fig. 13, our approach produces reasonable results at high sampling rates as well, which are comparable in quality to other methods.

In this paper, we focused on reconstruction using uniform low-discrepancy samples as input, and showed we could get good results without adaptive sampling. However, methods like SBF, NLM, RDFC, and WLR all significantly benefit from adaptive sampling by dedicating more samples to problematic regions, and it is possible our method could benefit as well. A straightforward way of including adaptive sampling in our framework is to use an existing approach to adaptively generate samples that we would then train on. However, a good avenue for future research is to use a similar learning approach to generate sampling densities as well.

Finally, our approach may be the first attempt to apply machine learning to Monte Carlo noise reduction, but it is probably not the only way to do this. We used a neural network as our nonlinear regression model since it was simple, effective, and fast. However, it might be possible to replace the neural network with another nonlinear regression model, such as a support vector machine [Drucker et al. 1997; Suykens and Vandewalle 1999]. We leave the investigation of such possibilities for future work.

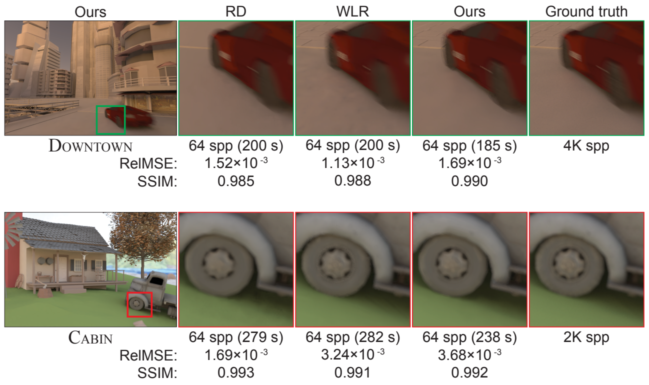


Figure 13: *Comparison between our approach and two other methods using a cross-bilateral filter at 64 samples per pixel. Our method produces results with comparable quality to other algorithms at this high sampling rate. Note that these results are produced using the network trained with 4, 8, 16, 32, and 64 samples per pixel, as in Fig. 8.* Scene credits: DOWNTOWN – *Herminio Nieves (buildings), tf3dm.com user ysup12 (car);* CABIN – *Andrew Kin Fun Chan and Dan Konieczka (cabin, mountain, tree), tf3dm.com user 3dregenerator (truck).*

7 Conclusion

We have presented a machine learning approach to reduce noise in Monte Carlo (MC) rendered images. In order to model the complex relationship between the ideal filter parameters and a set of features extracted from the input noisy samples, we use a multilayer perceptron (MLP) neural network as a nonlinear regression model. To effectively train the network, we combine the MLP network with a filter such that the standard MLP takes in a set of secondary features extracted from a local neighborhood at each pixel and outputs a set of filter parameters. These parameters and the noisy samples are given as inputs to the filter to generate a filtered pixel that is compared to the ground truth pixel during training. We train our proposed system on a set of scenes with a variety of distributed effects and then test it on different scenes containing motion blur, depth of field, area lighting, glossy reflections, and global illumination. Our results show that this simple approach demonstrates visible improvement over existing state-of-the-art methods.

Appendix: Filter derivatives

Since we use the backpropagation algorithm [Rumelhart et al. 1986] to train the network, we need to analytically compute the derivative of the output error with respect to the weights. According to Eq. 9, this requires the filter be differentiable with respect to the filter parameters. Observing that the filter weights $d_{i,j}$ are a function of the filter parameters $\boldsymbol{\theta}_i$ and using the chain rule and Eq. 1, we have:

$$\frac{\partial \hat{c}_{i,q}}{\partial \theta_{m,i}} = \frac{\partial h_q(\bar{\mathbf{s}}_{\mathcal{N}(i)}, \boldsymbol{\theta}_i)}{\partial \theta_{m,i}} = \sum_{j \in \mathcal{N}(i)} \frac{\partial h_q(\bar{\mathbf{s}}_{\mathcal{N}(i)}, \boldsymbol{\theta}_i)}{\partial d_{i,j}} \frac{\partial d_{i,j}}{\partial \theta_{m,i}}. \quad (13)$$

Using the quotient rule, we can write the first term as:

$$\frac{\partial h_q(\bar{\mathbf{s}}_{\mathcal{N}(i)}, \boldsymbol{\theta}_i)}{\partial d_{i,j}} = \frac{\bar{\mathbf{c}}_{j,q} \sum_{n \in \mathcal{N}(i)} d_{i,n} - \sum_{n \in \mathcal{N}(i)} d_{i,n} \bar{\mathbf{c}}_{n,q}}{\left(\sum_{n \in \mathcal{N}(i)} d_{i,n}\right)^2}. \quad (14)$$

The second term in Eq. 13 depends on the filter choice. For the cross-bilateral filter defined in Eq. 2, we need to take the derivative with respect to the position (α_i), color (β_i), and additional features ($\gamma_{k,i}$) filter parameters:

$$\frac{\partial d_{i,j}}{\partial \alpha_i} = d_{i,j}\frac{\|\bar{\mathbf{p}}_i - \bar{\mathbf{p}}_j\|^2}{\alpha_i^3},$$
$$\frac{\partial d_{i,j}}{\partial \beta_i} = d_{i,j}\frac{D(\bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j)}{\beta_i^3},$$
$$\frac{\partial d_{i,j}}{\partial \gamma_{k,i}} = d_{i,j}\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{\gamma_{k,i}^3}. \quad (15)$$

Other differentiable filters can be applied similarly in our system. For example, we can use the cross non-local means filter [Li et al. 2012; Rousselle et al. 2013]:

$$d_{i,j} = \exp(-\frac{D(\bar{\mathbf{Q}}_i, \bar{\mathbf{Q}}_j)}{2\rho_i^2}) \times \prod_{k=1}^{K} \exp(-\frac{D_k(\bar{\mathbf{f}}_{i,k}, \bar{\mathbf{f}}_{j,k})}{2\gamma_{k,i}^2}), \quad (16)$$

where $\bar{\mathbf{Q}}_i$ is a small patch (5×5 in our implementation) of mean sample colors around pixel i and ρ_i is the standard deviation at pixel i for the non-local means term. Furthermore, D is defined as:

$$D(\bar{\mathbf{Q}}_i, \bar{\mathbf{Q}}_j) = \frac{1}{|\bar{\mathbf{Q}}|}\sum_{k \in \mathcal{K}} \frac{\|\bar{\mathbf{c}}_{i+k} - \bar{\mathbf{c}}_{j+k}\|^2}{\psi_{i+k}^2 + \psi_{j+k}^2 + \zeta} \quad (17)$$

where $|\bar{\mathbf{Q}}|$ is the number of pixels in the patch (25 for a 5×5 patch) and $\mathcal{K}$ is a set of offsets from the center pixel within a patch. Moreover, ψ_{i+k} and ψ_{j+k} are the standard deviations of color samples at pixel $i + k$ and $j + k$, respectively, and ζ is a small number (10^{-10}) to avoid division by zero. The derivative of this filter with respect to the filter parameters can be calculated similarly to the cross-bilateral filter. The only difference is that, instead of the position and color terms in Eq. 15, we have:

$$\frac{\partial d_{i,j}}{\partial \rho_i} = d_{i,j}\frac{D(\bar{\mathbf{Q}}_i, \bar{\mathbf{Q}}_j)}{\rho_i^3} \quad (18)$$

Note that the cubic terms in the denominator of Eqs. 15 and 18 increase computational complexity. Therefore, in our implementation, we define auxiliary filter parameters $\lambda_{m,i} = 1/(2\theta_{m,i}^2)$ to use instead of the original filter parameters. The filter derivatives with respect to these auxiliary parameters do not have the cubic term in the denominator and are, therefore, more computationally efficient. To reflect this change in our system, the network outputs these auxiliary parameters and the filters defined in Eqs. 2 and 16 are modified appropriately.

Acknowledgments

We thank Nvidia for the hardware donation of a GeForce GTX TITAN. This work was funded by National Science Foundation CAREER grants IIS-1342931 and IIS-1321168.

References

BALA, K., WALTER, B., AND GREENBERG, D. P. 2003. Combining edges and points for interactive high-quality rendering. *ACM Trans. Graph. 22*, 3 (July), 631–640.

BAUSZAT, P., EISEMANN, M., AND MAGNOR, M. 2011. Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum 30*, 4, 1361–1368.

BUADES, A., COLL, B., AND MOREL, J. M. 2005. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation 4*, 2, 490–530.

BURGER, H., SCHULER, C., AND HARMELING, S. 2012. Image denoising: Can plain neural networks compete with BM3D? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2392–2399.

CHEN, J., WANG, B., WANG, Y., OVERBECK, R. S., YONG, J.-H., AND WANG, W. 2011. Efficient depth-of-field rendering with adaptive sampling and multiscale reconstruction. *Computer Graphics Forum 30*, 6, 1667–1680.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. *SIGGRAPH Comput. Graph. 18*, 3 (Jan.), 137–145.

DACHSBACHER, C. 2011. Analyzing visibility configurations. *IEEE Transactions on Visualization and Computer Graphics 17*, 4 (April), 475–486.

DAMMERTZ, H., SEWTZ, D., HANIKA, J., AND LENSCH, H. P. 2010. Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In *Proceedings of High Performance Graphics (HPG)*, 67–75.

DELBRACIO, M., MUSÉ, P., BUADES, A., CHAUVIER, J., PHELPS, N., AND MOREL, J.-M. 2014. Boosting Monte Carlo rendering by ray histogram fusion. *ACM Trans. Graph. 33*, 1 (Feb.), 8:1–8:15.

DRUCKER, H., BURGES, C. J. C., KAUFMAN, L., SMOLA, A., AND VAPNIK, V. 1997. Support vector regression machines. In *Advances in Neural Information Processing Systems 9*, MIT Press, 155–161.

DUTRÉ, P., BALA, K., BEKAERT, P., AND SHIRLEY, P. 2006. *Advanced Global Illumination*. AK Peters Ltd.

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 (July), 93:1–93:13.

EGAN, K., DURAND, F., AND RAMAMOORTHI, R. 2011. Practical filtering for efficient ray-traced directional occlusion. *ACM Trans. Graph. 30*, 6 (Dec.), 180:1–180:10.

EGAN, K., HECHT, F., DURAND, F., AND RAMAMOORTHI, R. 2011. Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Trans. Graph. 30*, 2 (Apr.), 9:1–9:13.

GRZESZCZUK, R., TERZOPOULOS, D., AND HINTON, G. 1998. Neuroanimator: Fast neural network emulation and control of physics-based models. In *ACM SIGGRAPH '98*, ACM, New York, NY, USA, 9–20.

HACHISUKA, T., JAROSZ, W., WEISTROFFER, R. P., DALE, K., HUMPHREYS, G., ZWICKER, M., AND JENSEN, H. W. 2008. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27* (Aug.), 33:1–33:10.

HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J., HASTIE, T., FRIEDMAN, J., AND TIBSHIRANI, R. 2009. *The elements of statistical learning*, vol. 2. Springer.

JAKOB, W., REGG, C., AND JAROSZ, W. 2011. Progressive expectation-maximization for hierarchical volumetric photon mapping. *Computer Graphics Forum 30*, 4, 1287–1297.

JENSEN, H. W., AND CHRISTENSEN, N. J. 1995. Optimizing path tracing using noise reduction filters. In *Winter School of Computer Graphics (WSCG) 1995*, 134–142.

JENSEN, H. W. 2001. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA.

KALANTARI, N. K., AND SEN, P. 2013. Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum 32*, 2pt1, 93–102.

LAINE, S., SARANSAARI, H., KONTKANEN, J., LEHTINEN, J., AND AILA, T. 2007. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, 277–286.

LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHI, R. 2004. Efficient BRDF importance sampling using a factored representation. *ACM Trans. Graph. 23*, 3 (Aug.), 496–505.

LE CUN, Y., BOTTOU, L., ORR, G. B., AND MÜLLER, K.-R. 1998. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag.

LEE, M., AND REDNER, R. 1990. A note on the use of nonlinear filtering in computer graphics. *IEEE Computer Graphics and Applications 10*, 3 (May), 23–29.

LEHTINEN, J., AILA, T., CHEN, J., LAINE, S., AND DURAND, F. 2011. Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4 (Aug.), 55:1–55:12.

LI, T.-M., WU, Y.-T., AND CHUANG, Y.-Y. 2012. Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph. 31*, 6 (Nov.), 194:1–194:9.

MCCOOL, M. D. 1999. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Trans. Graph. 18*, 2 (Apr.), 171–194.

MEHTA, S., WANG, B., AND RAMAMOORTHI, R. 2012. Axis-aligned filtering for interactive sampled soft shadows. *ACM Trans. Graph. 31*, 6 (Nov.), 163:1–163:10.

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 (July), 96:1–96:12.

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 (July), 57:1–57:12.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. *SIGGRAPH Comput. Graph. 21*, 4 (Aug.), 65–72.

MOON, B., CARR, N., AND YOON, S.-E. 2014. Adaptive rendering based on weighted local regression. *ACM Trans. Graph. 33*, 5 (Sept.), 170:1–170:14.

NOWROUZEZAHRAI, D., KALOGERAKIS, E., AND FIUME, E. 2009. Shadowing dynamic scenes with arbitrary BRDFs. *Computer Graphics Forum 28*, 2, 249–258.

OVERBECK, R. S., DONNER, C., AND RAMAMOORTHI, R. 2009. Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5 (Dec.), 140:1–140:12.

PHARR, M., AND HUMPHREYS, G. 2010. *Physically Based Rendering: From Theory to Implementation*, second ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

REN, P., WANG, J., GONG, M., LIN, S., TONG, X., AND GUO, B. 2013. Global illumination with radiance regression functions. *ACM Trans. Graph. 32*, 4 (July), 130:1–130:12.

RIEDMILLER, M., AND BRAUN, H. 1993. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *IEEE International Conference on Neural Networks*, 586–591 vol.1.

ROUSSELLE, F., KNAUS, C., AND ZWICKER, M. 2011. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph. 30*, 6 (Dec.), 159:1–159:12.

ROUSSELLE, F., KNAUS, C., AND ZWICKER, M. 2012. Adaptive rendering with non-local means filtering. *ACM Trans. Graph. 31*, 6 (Nov.), 195:1–195:11.

ROUSSELLE, F., MANZI, M., AND ZWICKER, M. 2013. Robust denoising using feature and color information. *Computer Graphics Forum 32*, 7, 121–130.

RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. 1986. Learning representations by back-propagating errors. *Nature 323* (Oct.), 533–536.

RUSHMEIER, H. E., AND WARD, G. J. 1994. Energy preserving non-linear filters. In *ACM SIGGRAPH '94*, 131–138.

SEGOVIA, B., IEHL, J. C., MITANCHEY, R., AND PÉROCHE, B. 2006. Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, ACM, New York, NY, USA, GH '06, 53–60.

SEN, P., AND DARABI, S. 2011. Implementation of random parameter filtering. Tech. Rep. EECE-TR-11-0004, University of New Mexico.

SEN, P., AND DARABI, S. 2012. On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Trans. Graph. 31*, 3 (June), 18:1–18:15.

SHIRLEY, P., AILA, T., COHEN, J., ENDERTON, E., LAINE, S., LUEBKE, D., AND MCGUIRE, M. 2011. A local image reconstruction algorithm for stochastic rendering. In *Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '11, 9–14.

SHIRLEY, P. 1991. Discrepancy as a quality measure for sample distributions. In *Proc. Eurographics*, vol. 91, 183–194.

STEIN, C. M. 1981. Estimation of the mean of a multivariate normal distribution. *The Annals of Statistics 9*, 6, 1135–1151.

SUYKENS, J., AND VANDEWALLE, J. 1999. Least squares support vector machine classifiers. *Neural Processing Letters 9*, 3, 293–300.

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision*, 839–846.

VEACH, E., AND GUIBAS, L. J. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In *ACM SIGGRAPH '95*, ACM, New York, NY, USA, 419–428.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *ACM SIGGRAPH '97*, ACM, New York, NY, USA, 65–76.

WANG, Z., BOVIK, A., SHEIKH, H., AND SIMONCELLI, E. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (April), 600–612.

XU, R., AND PATTANAIK, S. N. 2005. A novel Monte Carlo noise reduction operator. *IEEE Computer Graphics and Applications 25*, 31–35.