Eurographics Symposium on Rendering 2018
T. Hachisuka and W. Jakob
(Guest Editors)

Volume 37 (2018), Number 4

Deep Adaptive Sampling for Low Sample Count Rendering

Alexandr Kuznetsov Nima Khademi Kalantari Ravi Ramamoorthi

University of California, San Diego

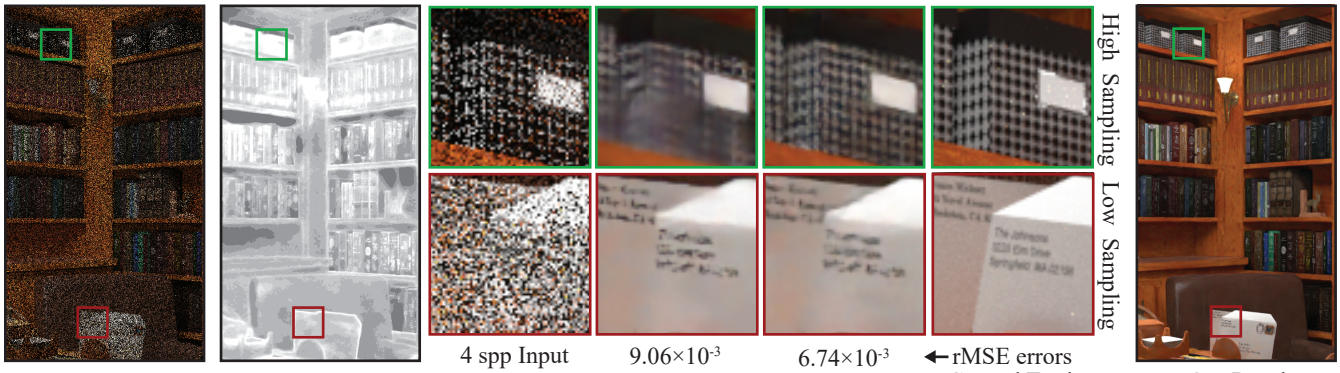

Figure 1: *We propose a deep learning approach for joint adaptive sampling and reconstruction of Monte Carlo (MC) rendered images. Using a convolutional neural network (CNN), we first estimate a sampling map from a 1 sample per pixel (spp) MC rendered input (shown on the left). In the sampling map, brighter areas indicate more adaptive samples. We then use the map to distribute three additional samples per pixel on average and denoise the resulting 4 spp render using another CNN to obtain our final image. We train both networks in an end-to-end fashion, and thus, our sampling map network distributes more samples to the areas that are challenging for the denoiser. For example, the top (green) inset shows an area which is challenging for the denoiser. Our approach throws more samples in this region, and thus, we are able to produce results with higher quality than uniform sampling. On the other hand, the denoiser can easily remove the noise from the envelope in the bottom (red) inset using the provided auxiliary buffers, such as shading normals and textures. Therefore, our approach does not waste the sampling budget for this area. The releative mean squared errors (rMSE), listed below the insets, are computed on the entire images. See Fig. 7 for comparison against several other approaches on a variety of scenes.*

Abstract

Recently, deep learning approaches have proven successful at removing noise from Monte Carlo (MC) rendered images at extremely low sampling rates, e.g., 1-4 samples per pixel (spp). While these methods provide dramatic speedups, they operate on uniformly sampled MC rendered images. However, the full promise of low sample counts requires both adaptive sampling and reconstruction/denoising. Unfortunately, the traditional adaptive sampling techniques fail to handle the cases with low sampling rates, since there is insufficient information to reliably calculate their required features, such as variance and contrast. In this paper, we address this issue by proposing a deep learning approach for joint adaptive sampling and reconstruction of MC rendered images with extremely low sample counts. Our system consists of two convolutional neural networks (CNN), responsible for estimating the sampling map and denoising, separated by a renderer. Specifically, we first render a scene with one spp and then use the first CNN to estimate a sampling map, which is used to distribute three additional samples per pixel on average adaptively. We then filter the resulting render with the second CNN to produce the final denoised image. We train both networks by minimizing the error between the denoised and ground truth images on a set of training scenes. To use backpropagation for training both networks, we propose an approach to effectively compute the gradient of the renderer. We demonstrate that our approach produces better results compared to other sampling techniques. On average, our 4 spp renders are comparable to 6 spp from uniform sampling with deep learning-based denoising. Therefore, 50% more uniformly distributed samples are required to achieve equal quality without adaptive sampling.

CCS Concepts

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

1. Introduction

Monte Carlo (MC) path tracing [Kaj86] is a powerful and general technique for physically-based rendering, but requires evaluating a large number of samples to produce noise-free images, resulting in lengthy render times. This problem has been the subject of extensive study and many approaches have been developed to quickly render a noisy image at reduced sampling rates and remove the noise in a post-process [ZJL*15]. As a result, MC rendering has now become standard for offline rendering [KFF*15].

However, with the rise of virtual and augmented reality systems and the recent interest of gaming companies in real-time ray tracing, generating noise-free results from input images with *extremely* low sampling rates (e.g., 4 samples per pixel) is becoming an important research topic. The most recent filtering approaches [CKS*17, SKW*17] have focused on this problem and are able to generate high-quality results from input images with 1-4 samples per pixel (spp). These advances have led to dramatic speedups, bringing computation times from hours or days to seconds or even real-time frame rates, often limited only by the speed of the base path tracer. Indeed, deep-learning based denoising [CKS*17] is integrated into NVIDIA's latest Optix5 GPU real-time raytracer.

The major drawback of these approaches is that they only focus on the filtering problem and use uniformly sampled images as the input. However, previous techniques [Mit87, Mit91, Guo98, HJW*08, ODR09, SSD*09, ETH*09, RKZ11, LWC12, RMZ13] have demonstrated that the quality of the output images can be significantly improved through adaptive sampling. These methods dedicate a portion of the total sampling budget to render an initial image by uniformly distributing the samples. They then compute the variance, coherence maps [Guo98], or frequency content [ETH*09], to estimate a sampling map to adaptively distribute the rest of the samples. Unfortunately, these approaches typically require the initial uniformly sampled image to be rendered with a reasonably large number of samples. Therefore, they fail to properly estimate a sampling map when the initial image is rendered with extremely low sample counts (1 spp in our case). This is the main reason that the most recent filtering techniques [CKS*17, SKW*17] only use uniformly sampled images as the input.

In this paper, we address this challenge by using deep learning for *both* the adaptive sampling and filtering/reconstruction stages. We first render a uniformly sampled image with 1 spp, which is then used along with several auxiliary buffers, such as shading normals and texture, as the input to a convolutional neural network (CNN) to estimate a sampling map (see Fig. 2). We then use the map to adaptively distribute three additional samples and generate an image with an average of 4 spp. Finally, the 4 spp image along with other auxiliary buffers are passed to another CNN to produce the final denoised image. We train both networks in an end-to-end fashion by minimizing the error of the denoised and ground truth images on a wide range of scenes with a variety of distributed effects, such as depth of field, global illumination, and motion blur. Specifically, our work makes the following contributions:

- We propose the first learning-based approach for adaptive sampling in MC rendering. We demonstrate better results than other sampling techniques on general scenes with soft shadows, depth of field, and global illumination (see Fig. 7). Overall, our 4 spp results are comparable to 6 spp results with uniform sampling. We thus take Monte Carlo rendering an important step closer towards low spp for interactive and rapid preview applications.
- Since training is done iteratively, online rendering to adaptively place the samples during training is impractical. We propose an efficient approach for generating rendered images with arbitrary number of samples to make the training practical (Sec. 4.1).
- We present a method for computing the gradient of the rendered images with respect to the sample counts, which is required for the end-to-end training (Sec. 4.2). As a result, our sampling map network learns to throw more samples in the areas that are difficult for the denoiser to filter (Figs. 1, 8 and 9).

2. Previous Work

There has been extensive research to address the Monte Carlo (MC) noise problem through adaptive sampling and reconstruction. A thorough review of all these approaches is beyond the scope of this paper. For brevity, here we only focus on the most relevant work and refer the readers to the review by Zwicker et al. [ZJL*15].

Shortly after the introduction of MC rendering [CPC84], several approaches proposed to reduce the noise by distributing the samples adaptively. These approaches [LRU85, Mit87, Mit91, BM98] use a specific metric (e.g., contrast or variance) to detect the problematic areas and assign more samples to them. Hachisuka et al. [HJW*08] propose to dedicate more samples to regions of high contrast in a multidimensional space. Overbeck et al. [ODR09] use a metric based on contrast and wavelet coefficients to estimate the sampling map. Several approaches [SSD*09, ETH*09] propose to adaptively place the samples based on the frequency content of the image. Kalantari et al. [KS13] detect the noise at each pixel using mean absolute deviation and throw more samples in the noisy regions. However, these methods suffer from two main drawbacks. First, they typically require the initial image to be sampled with a reasonably large number of samples, and thus, produce unsatisfactory results in our cases with extremely low sample counts. Second, these approaches often have a reconstruction stage to remove noise from the rendered images. However, the adaptive sampling stage of these methods is independent of the reconstruction process, and thus, is not optimal.

To address the second issue, several approaches have proposed to perform joint adaptive sampling and reconstruction. These methods use no-reference error estimation metrics to select the best parameter for a filter and assign more samples to the regions with large error. Several algorithms estimate the reconstruction error by computing the bias and variance of a Gaussian [RKZ11], non-local means [RKZ12], local linear regression [MCY14], and polynomial [MMMG16] filters and use the error to adaptively sample the image in multiple stages. A few approaches use Stein unbiased risk estimator (SURE) as their no-reference error estimation metric [LWC12, RMZ13]. Bauszat et al. [BEEM15] proposes to reconstruct a dense error map from a set of sparse, but robust estimates. Finally, Moon et al. [MGYM15] propose an iterative and recursive error estimation technique to perform joint sampling and filtering. The main drawback of these approaches is that they are not able to

reliably estimate the reconstruction error on images with low sampling rates.

Our method is inspired by the recent learning-based approaches for filtering MC noise [KBS15, BVM*17, CKS*17]. Bauszat et al. [BEM11] distributes the samples based on the summation of filter weights. Kalantari et al. [KBS15] propose to use a neural network to set the parameters of a cross-bilateral or cross non-local means filter. Bako et al. [BVM*17] extends this idea by replacing the entire pipeline with a deep convolutional neural network (CNN). Chaitanya et al. [CKS*17] propose to remove MC noise using an encoder-decoder network and demonstrate high-quality results on rendered images with 1-4 samples per pixel. Within real-time rendering, we also build on many recent frequency-based papers [MWR12, MYRD14], and particularly the approaches by Yan et al. [YMRD15] and Wu et al. [WYKR17]), which introduce denoising for very low sampling rates (e.g., 4-9 spp). However, all of these methods only work on uniformly sampled images. In contrast, we propose a deep learning approach to perform joint sampling and reconstruction on images with extremely low adaptive sampling rates.

Adaptive sampling at 4 spp, which requires estimating the sampling map from 1 spp images, has to our knowledge never been demonstrated previously. However, to make a fair comparison, we implement and compare to a few simple alternative algorithms including variance [LRU85] and SURE-based [LWC12] approaches. As discussed in Sec. 6, since computing the statistics in each pixel using only one sample is impossible, we compute them over a patch of size 4×4. We also compare against uniform sampling with equal rate (4 spp) and equal quality.

3. Method

Given a fixed budget of a small number of samples n (e.g., 4 samples per pixel), the goal of our approach is to generate an image as close as possible to the ground truth image rendered with many samples. We achieve this goal by first effectively distributing the samples to render a noisy image and then removing the noise to produce a high-quality image. The overview of our approach is given in Fig. 2. In the following sections, we describe our sampling map estimator and denoiser.

3.1. Sampling Map Estimator

We propose to model the sampling map estimation process using a convolutional neural network (CNN). In this case, the CNN takes the 1 spp noisy image as well as several auxiliary features as the input (11 channels) to produce a single channel output. Specifically, the CNN's input consists of the noisy image in RGB format (3 channels), textures in RGB format (3 channels), shading normals (3 channels), depth (one channel), and direct illumination visibility (one channel).

We use the output of the network to generate the final sampling map, which is used for distributing the additional three samples per pixel. Therefore, the average of the final sampling map for all the pixels should be close to the remaining sampling budget, i.e., 3 spp. To enforce this, we compute the final sampling map, s from the output of the network, x, using the following normalization process:

$$s(p) = \text{round}\left(\frac{M}{\sum_{j=1}^{M} e^{x(j)}} \times e^{x(p)} \times n\right), \tag{1}$$

where $s(p)$ refers to the sampling map at pixel p, M is the number of pixels in the image, and n is the remaining sampling budget (3 in our case). Note that, the exponential terms make the network's output, which is in the range $(-\infty, \infty)$, positive. Here, the first two terms basically normalize the network's output to have an average of one, and the last term enforces the sampling map to have an average of n spp. Finally, the round operator ensures that the sampling map at each pixel has an integer value.

The sampling map, s is then used by the renderer to adaptively distribute the three additional samples per pixel and produce an image with an average of 4 spp (including the initial 1 spp). This image is then used as the input to the next stage (denoiser) to produce the final denoised image.

As shown in Fig. 3, we use an encoder-decoder architecture similar to the one used by Chaitanya et al. [CKS*17] for denoising. Intuitively, the sampling map network needs to estimate the quality of the results after denoising and dedicate more samples to the problematic areas. Therefore, we use the same network architecture for the sampling map and denoiser networks.

The encoder and decoder in our network each have a total of 5 units, and we have a bottleneck unit in between them. Each encoder unit consists of two convolutional layers with kernel size of 3×3, followed by a 2×2 max pooling layer. The decoder units consist of an upsampling layer, followed by 2 convolutional layers with kernel size of 3×3. Since our system works on single images, we remove the recurrent connections from Chaitanya et al.'s network, which were used to handle sequences of images. Therefore, each unit in both the encoder and decoder contains two convolutional layers instead of three. Moreover, following Chaitanya et al. we use skip connections between the convolutional layers in the encoder and decoder to help the network produce sharper sampling maps.

3.2. Denoiser

The goal of the denoiser is to take the adaptively rendered image from the previous stage and produce a high-quality image comparable to the ground truth. We propose to use a CNN, shown in Fig. 3, to model the denoising process. Our CNN takes the noisy rendered image with an average of 4 spp (3 channels) as well as the auxiliary features (8 channels), and produces the output image (3 channels).

Note that, Chaitanya et al. [CKS*17] separate the texture from illumination and use a CNN to only denoise the illumination image. Therefore, their system is not able to handle scenes with distributed effects, like depth of field and motion blur, since they contaminate the texture with noise. In our system, we use the CNN to directly denoise the noisy rendered image, and thus, are able to handle general distributed effects (see Fig. 7).

4. Training

As discussed, we propose to train both the sampling map estimator and the denoiser networks by minimizing the error between the

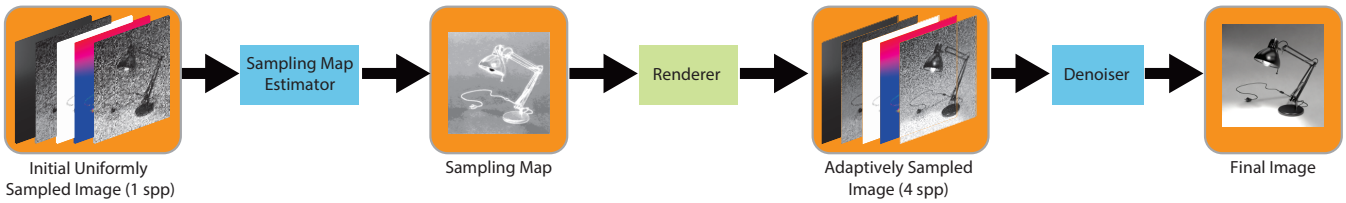

Figure 2: *In our system, we first render a scene with one spp and use it to calculate the sampling map. Then, we use the sampling map to render three additional samples. Finally, we denoise the resulting rendered image with an average of 4 spp to obtain the final denoised image.*

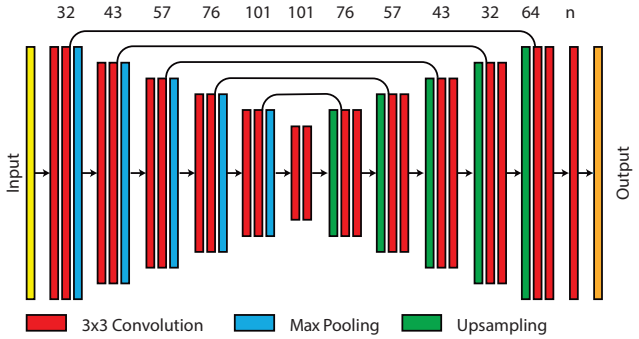

Figure 3: *We use an encoder-decoder architecture for both the sampling map and denoiser networks. Our network contains 5 encoder and decoder units, separated by a bottleneck unit. Each encoder consists of two convolutional layers followed by a max pooling layer, while the decoder units consist of an upsampling layer followed by two convolutional layers. All the convolutional layers have a kernel size of* 3×3*. We use skip connections between the convolutional layers in the encoder and decoder units. We use ReLU activation function after all the convolutional layers except the last one. Note that, the output of the sampling map network has one channel, while the denoiser's output has three channels.*

denoised and ground truth images in an end-to-end fashion. In our system, the renderer is in between the two CNNs, and thus, we need to overcome two major challenges to effectively train the networks. First, a large number of iterations are required for the system to converge during training. At every iteration, we need to render additional samples based on the estimated sampling map from the first stage, which makes the training impractical as rendering is computationally expensive. Second, to train the sampling map network using backpropagation, we need to differentiate the renderer, which is not straightforward. Next, we discuss our proposed approach for addressing these two issues as well as the optimization process.

4.1. Render Simulator

A simple way to address the computational complexity of the rendering process is to pre-render a set images at different sampling rates, $I_1, \cdots, I_N$, where N is a large number. We can then choose the appropriate image for each pixel based on the estimated sampling map during training. However, this approach requires a significant amount of computational power and storage for rendering and saving the images as well as the auxiliary buffers at all the sampling rates.

Our main observation is that we can combine two independently rendered images to generate an image with higher sampling counts. For example, we can combine two independently rendered images with 3 and 4 spp to produce an image with 7 spp as:

$$I_7 = \frac{3\, I_3 + 4\, I_4}{7},$$

where I_s is the rendered image with s samples per pixel. Therefore, we propose to pre-render a set of images with power of two sampling rates, $I_{2^0}, I_{2^1}, \cdots, I_{2^k}$, where 2^k is a large number (1024 and more). We then produce images with any number of samples by combining the appropriate images in the set as follows:

$$I_s = \frac{\sum_{i=0}^{k} 2^i\, I_{2^i}\, g(s, 2^i)}{s}, \tag{2}$$

where I_s is the image with the desired sampling rate and I_{2^i} is the pre-rendered image with 2^i samples. Here, $g(s, 2^i)$ is a binary value of 0 or 1 according to whether the binary representation of s includes the term 2^i. Note that, since the sampling map has different values at each pixel, we evaluate this expression separately for each pixel, but in this equation we omit the pixel notation for simplicity. For each render at a given sampling rate, there is an infinite number of noise patterns. Therefore, to make our network more robust to noise, we render multiple alternative images with different seeds and pick one at runtime for each power of two sampling rate.

An alternative approach is to render a large number of 1 spp images and then generate an image with arbitrary number of samples, n, by combining n 1 spp images. Although this approach is more efficient than our proposed power of two pre-rendering method, it requires a significant amount of storage to store the 1 spp rendered images as well as the auxiliary buffers. Our method provides a reasonable trade-off between storage and computational complexity.

4.2. Gradient of the Renderer

In order to train the networks in an end-to-end fashion, we need to differentiate the renderer, i.e., compute the gradient of the rendered image with respect to the sampling map at each pixel. Unfortunately, the renderer is a complex system and there is not a straightforward way to calculate the gradient analytically. Therefore, we propose to compute the gradient numerically as follows:

$$\frac{\partial I_s}{\partial s} = \frac{I_{s+h} - I_s}{h}, \quad \text{where} \quad I_{s+h} = \frac{s\, I_s + h\, I_h}{s+h}, \tag{3}$$

where h is the additional samples (e.g., 1 or 2), used for the numerical differentiation. Here, I_{s+h} is obtained using Eq. 2 from the original image with s spp, I_s, and the image rendered with additional

h spp, I_h. Unfortunately, since the rendering process is stochastic, this gradient is extremely noisy and makes the training unstable.

One way to address this problem is to compute the gradient over a large number of independently rendered h spp images as follows:

$$\frac{\partial I_s}{\partial s} = \sum_{i=1}^{N} \frac{I^i_{s+h} - I_s}{h\,N}, \quad \text{where} \quad I^i_{s+h} = \frac{s\,I_s + h\,I^i_h}{s+h}, \tag{4}$$

where N is a large number to ensure the gradients are noise-free. However, this approach requires pre-rendering a large number of images with h spp, which is computationally expensive and requires significant amount of storage. To address this issue, we simplify Eq. 4 as follows:

$$\frac{\partial I_s}{\partial s} = \sum_{i=1}^{N} \frac{s\,I_s + h\,I^i_h - (s+h)\,I_s}{h\,N\,(s+h)}. \tag{5}$$

Here, we have replaced I^i_{s+h} with its definition in Eq. 4 and performed basic simplifications. Next, we apply the summation to each term in the numerator as follows:

$$\frac{\partial I_s}{\partial s} = \frac{\sum_{i=1}^{N} I^i_h/N - \sum_{i=1}^{N} I_s/N}{s+h}. \tag{6}$$

Here, the first term in the numerator is the average of a large number of independently rendered images with h spp, and thus, in the limit ($N \to \infty$), becomes the ground truth image I_∞.[†] Since I_s is independent of i, the second term can be simplified to I_s. Therefore, we have:

$$\frac{\partial I_s}{\partial s} = \frac{I_\infty - I_s}{s+h}. \tag{7}$$

As is typical in numerical differentiation, we set h equal to zero to obtain the final gradient:

$$\boxed{\frac{\partial I_s}{\partial s} = \frac{I_\infty - I_s}{s}}. \tag{8}$$

Note that, while setting h to zero is not physically possible (corresponds to no rendered image), we do this to ensure the gradient is independent of this parameter. As can be seen, the gradient is directly proportional to the distance of the ground truth and current image. Moreover, it is inversely proportional to the current sampling rate s. This is intuitive, since as the sampling rate increases, adding more samples does not significantly affect the pixel color. The final gradient in Eq. 8 is smooth and we can use it to effectively train both networks in an end-to-end fashion.

5. Implementation Details

In this section, we discuss the implementation details required for reproducing our results. Our implementation is available at: http://cseweb.ucsd.edu/~viscomp/projects/dasr/.

Dataset – Our training data consists of a set of 700 input and ground truth images from 50 scenes, where each scene is rendered from 2 to 30 different viewpoints. As shown in Fig. 4, our training data contains a wide range of images with a variety of distributed effects, such as depth of field, glossy reflections, motion blur, and global illumination.

Figure 4: *A subset of the scenes in our training set containing a variety of distributed effects, such as global illumination, glossy reflections, motion blur, and depth of field.*

We render the ground truth images with at least 1024 spp to obtain noise-free images. For inputs, we render a set of images with power of two sampling counts to be able to efficiently produce images with arbitrary sampling rates during training (Sec. 4). Moreover, by rendering multiple alternative images for the same power of two sampling count, we get more noise patterns.

Since the range of color and depth values are large, we compress their range using a logarithmic function before using them as the input to our system, i.e., $\log(1+z)$, where z is the pixel values of the color or depth. For the other auxiliary buffers, we perform standard normalization by computing the mean and standard deviation over the entire training set.

Training Details – Since we have two networks in our system, performing the training in a single stage is challenging. To reduce the complexity of the training process, we propose to perform it in three stages. In the first stage, we train the denoiser network on a set of noisy inputs and their corresponding ground truth images. To ensure the network is able to handle adaptively sampled images, we use randomly generated sampling maps to produce adaptively sampled input images with an average of 4 spp. Moreover, we generate the sampling maps randomly at lower resolution and obtain the final map using 8x upsampling to ensure smoothness. In the second stage, we use the denoiser network from the previous stage and only train our sampling map network until convergence. Finally, in the third stage, we fine tune both networks by alternating between training the sampling map and denoiser networks at every iteration.

As is common with the deep learning approaches, we break down the images into patches of size 512×512. For training, we use Adam solver [KB14] with the default parameters and a learning rate of 0.0001. We train our system using mini-batches of size 6 as we found they provide the best trade-off between the speed and convergence. In our system, the training in the first, second, and third stages converge after 25000, 5000, and 40000 iterations, respectively.

Loss – In order to measure the error between the denoised and ground truth images during training, we use the following loss consisting of spatial and gradient terms:

[†] While the ground truth image is theoretically obtained by rendering the scene with an infinite number of samples, in practice we consider the images rendered with a large but finite number of samples (e.g., 1024 spp) as ground truth.

	Variance	SURE	Uniform	Ours
rMSE	9.95	8.86	7.50	6.26

Table 1: *Comparison of our approach against other methods in terms of relative mean squared error (rMSE). We factor out* 10^{-3} *from all the reported values for clarity. The rMSE values are averaged over 60 test images. For equal quality, we obtain an rMSE of* 5.92×10^{-3}*, which is slightly lower than ours since we are only able to render images with an integer number of samples.*

$$\mathcal{L} = 0.5\mathcal{L}_s + 0.5\mathcal{L}_g \tag{9}$$

where $\mathcal{L}_s$ and $\mathcal{L}_g$ refer to the spatial and gradient terms, respectively. For the spatial loss, we use relative ℓ_1 as it properly models the human vision sensitivity to noise in the dark regions:

$$\mathcal{L}_s(\hat{c}, c) = \frac{\|\hat{c} - c\|_1}{|c| + \varepsilon} \tag{10}$$

where $\varepsilon = 0.01$ and $\hat{c}$ and c are the denoised and ground truth images, respectively. For the gradient term, we use a slightly modified version of the high frequency error norm (HFEN) [RB11] to ensure the edges in the final image are sharp. Specifically, our gradient term is defined as:

$$\mathcal{L}_g(\hat{c}, c) = \frac{\|g(\hat{c}) - g(c)\|_1}{|c| + \varepsilon} \tag{11}$$

where $g(\hat{c})$ is the Laplacian of Gaussian operator. Comparing to the original HFEN, we introduce the division by $|c| + \varepsilon$ to better model the human vision system's sensitivity to variations in the dark areas. Note that, since the range of color values can be significantly large, we apply a logarithmic function to the ground truth image to compress its range before computing the loss, i.e., $c = \log(1 + \tilde{c})$, where $\tilde{c}$ is the ground truth image in the linear domain.

6. Results

We implemented our approach in PyTorch [PGC*17] and compared against several other sampling techniques including adaptive sampling using variance [LRU85] and SURE [LWC12] metrics, as well as uniform sampling. Note that, adaptive sampling for extremely low sampling rates has not been demonstrated before. Since computing the variance of each pixel using one sample is impossible, we calculate it in a 4×4 patch to ensure we have sufficient samples. We use the same strategy to compute the required variance for the SURE-based sampling map. Moreover, we implemented the SURE-based approach using both the cross-bilateral filter, as used in the original paper, and our denoiser network. However, we only show their results using the cross-bilateral filter as we found it produces better results. The SURE's sampling map is noisy, so we apply another cross-bilateral filter to smooth it out. Note that, the cross-bilateral filters are only used to generate the sampling map to render the additional samples and they are not used for the final denoising.

We also perform equal quality (EQ) uniform sampling comparison. In all cases, we used our denoiser network to remove noise from the rendered images with different approaches. We use each approach to generate adaptively sampled input images on our training set and retrain the denoiser network to ensure the performance of the denoiser is optimized for the particular sampling strategy. Moreover, note that there is a long tail in the EQ distribution, which indicates that our adaptive sampling strategy enables robustness with large gains where possible, and more modest but still significant benefits in the common case.

	SURE - 1 spp	SURE - 2 spp	SURE - 3 spp
rMSE	8.86	8.32	8.08

Table 2: *Comparison of SURE method with different initial sampling counts in terms of relative mean squared error (rMSE). We factor out* 10^{-3} *from all the reported values for clarity. The rMSE values are averaged over 60 test images.*

Error Comparison – We show the average relative mean squared error (rMSE) of all the approaches on a set of 60 test scenes in Table 1. Our test scenes contain a variety of effects such as global illumination, motion blur, and glossy reflections and none of the test scenes are included in our training set. Although we optimize our approach using a different error function (Eq. 9), we still produce better results than other approaches in terms of rMSE, which demonstrates the generality of our system. It is worth noting that other adaptive sampling approaches perform poorly even compared to uniform sampling. This is mainly because 1 spp input images do not provide sufficient information for these approaches to reliably estimate their required statistics, and thus, their estimated sampling maps are usually noisy and inaccurate (see Figs. 8 and 9).

Note that, as discussed, we compute the variances for the SURE method in a 4×4 patch on the 1 spp input images. We also experimented with an alternative sampling strategy, where we use 2 or 3 spp input images to be able to directly compute the variances in each pixel. As shown in Table 2, the results of the SURE approach improve as the initial sampling rate increases. This is expected, since by increasing the initial uniform sampling rate, we decrease the budget for adaptive sampling. Therefore, SURE's results converge to the result of uniform sampling. However, in our main comparison, we start from 1 spp input images for all the approaches to have a fair comparison.

Moreover, we show the rMSE error of all the approaches, relative to ours, for our 60 test scenes in Fig. 5. Our approach produces better results than all the other approaches on all but two scenes where our error is slightly worse than uniform sampling. We also show the histogram of required sampling rates for the equal quality comparison over all the 60 images in Fig. 6. To achieve similar error to our method using uniform sampling (EQ), on average, we need to increase the sampling rate to 6.12 spp. Therefore, with uniform sampling we require 50% more than our adaptive sampling approach to achieve the same equality, which is a significant improvement.

Representative Scenes – We show the results of all the approaches for six representative scenes (see Fig. 5) from our test dataset visually (Figs. 1 and 7) and numerically (Table 3). For each scene, we show two insets with high and low sampling densities to demonstrate the effectiveness of our approach. In general, our method produces significantly better results in the top green insets with high adaptive sampling rate, while performing as well as other approaches in regions with low adaptive sampling rate shown in the

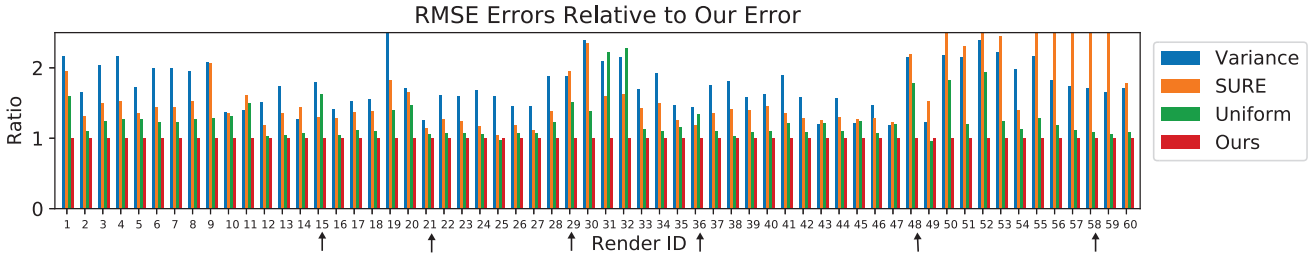

Figure 5: *We show the rMSE error of all the approaches relative to ours for the 60 test scenes. Our method produces better results than all the other algorithms on almost all the scenes. The six scenes, shown in Fig. 7, are indicated on this plot.*

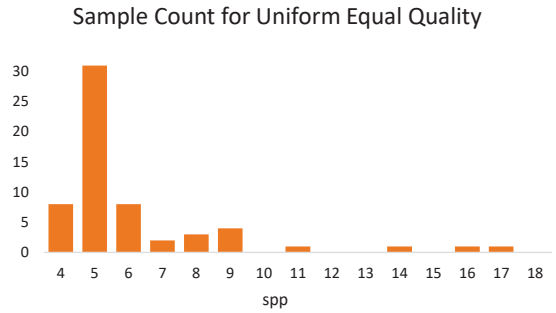

Figure 6: *We show the histogram of required sampling rate to achieve equal quality on all the 60 test scenes. The average of the required sampling rates are 6.12.*

red bottom insets. The DINING scene contains glasses with complex reflections and refractions. As shown in the green top inset, our approach correctly throws more samples in this area and is able to produce high-quality results. However, other approaches undersample this difficult region, and thus, their results contain residual noise. On the other hand, the noise on the cup in the red bottom inset can be easily removed by our denoiser. Therefore, despite having low sampling density in this area, we are able to produce comparable results to other techniques.

Next, we examine the STAIRWAY scene with global illumination. The highlights on the chandelier are difficult to capture, and thus, our approach assigns more samples to this region. As a result, our method produces an image with higher quality than the other approaches. However, the denoiser can effectively remove the noise in the red bottom inset, while preserving the sharp features by utilizing the auxiliary buffers like textures. Therefore, all the approaches produce high-quality results in this area.

The SPACESHIP scene is particularly challenging, since the auxiliary buffers do not contain any features for the visible objects behind the transparent glass. Our sampling map network assigns a significant number of samples to this challenging area, and thus, our method produces a better result than the other approaches. Note the missing sharp line in variance and uniform, as well as the blotchiness in SURE's results. The MONKEY HEAD is a scene with global illumination and motion blur. Other approaches produce results with blotchiness in the motion blur area (green top inset) because of lack of sufficient samples, while our method is able to dedicate the majority of samples to this region and produce a smooth result. On the other hand, the red bottom inset shows a region with low sampling density where all the approaches produce high-quality results.

ID	Name	Var.	SURE	Unif.	Ours
36	LIBRARY	9.73	7.99	9.06	**6.74**
29	DINING	4.16	4.32	3.34	**2.21**
21	STAIRWAY	8.70	7.94	7.39	**6.95**
15	SPACESHIP	2.13	1.55	1.94	**1.19**
48	MONKEY HEAD	0.37	0.37	0.31	**0.17**
58	PLANTS	0.75	1.44	0.47	**0.44**

Table 3: *Quantitative comparison in terms of relative mean squared error (rMSE) for the six images in Figs. 1 and 7. For clarity, we factor out* 10^{-3} *from all the values.*

The PLANTS scene demonstrates depth of field and soft shadow effects. As a result, auxiliary buffers are also noisy, but the denoiser is still able to handle this. In this case, a simple blurring of the scene suffices, and uniform sampling is adequate; our method provides only a moderate benefit.

Sampling Map – Figure 8 shows comparison of our sampling map against other techniques on different scenes. The variance-based method focuses on noisy regions, like the table cloth in the DINING scene, despite it's being very low frequency in the ground truth image. The SURE-based method misses some areas which are hard for the denoiser like the highlight on the lamp's base in the LIBRARY scene. Our method on the other hand, throws more samples in the areas which are hard for the denoiser like edges.

Figure 9 takes a closer look at the LIBRARY scene from Fig. 1. Here, the denoiser network has difficulty removing the noise from the green top inset, but can easily denoise the red bottom inset. The approach based on variance incorrectly throws more samples in the red inset, as it works independently of the denoiser. The SURE-based approach estimates the the error after filtering, but the 1 spp input does not provide sufficient information for this approach. Therefore, it is not able to properly sample the green inset, while at the same time wastes the sampling budget in the red inset. Our sampling map network produces a smooth map and is able to assign more samples to the problematic areas in the green inset, while dedicating fewer samples to the regions in the red inset. As shown, our method produces high-quality results in both cases, while other methods fail to handle the green inset.

Timing – Our method takes around 10 ms to evaluate the sampling map (5 ms) and denoiser (5 ms) networks on a machine with a GeForce GTX 1080 Ti GPU. This is a negligible overhead

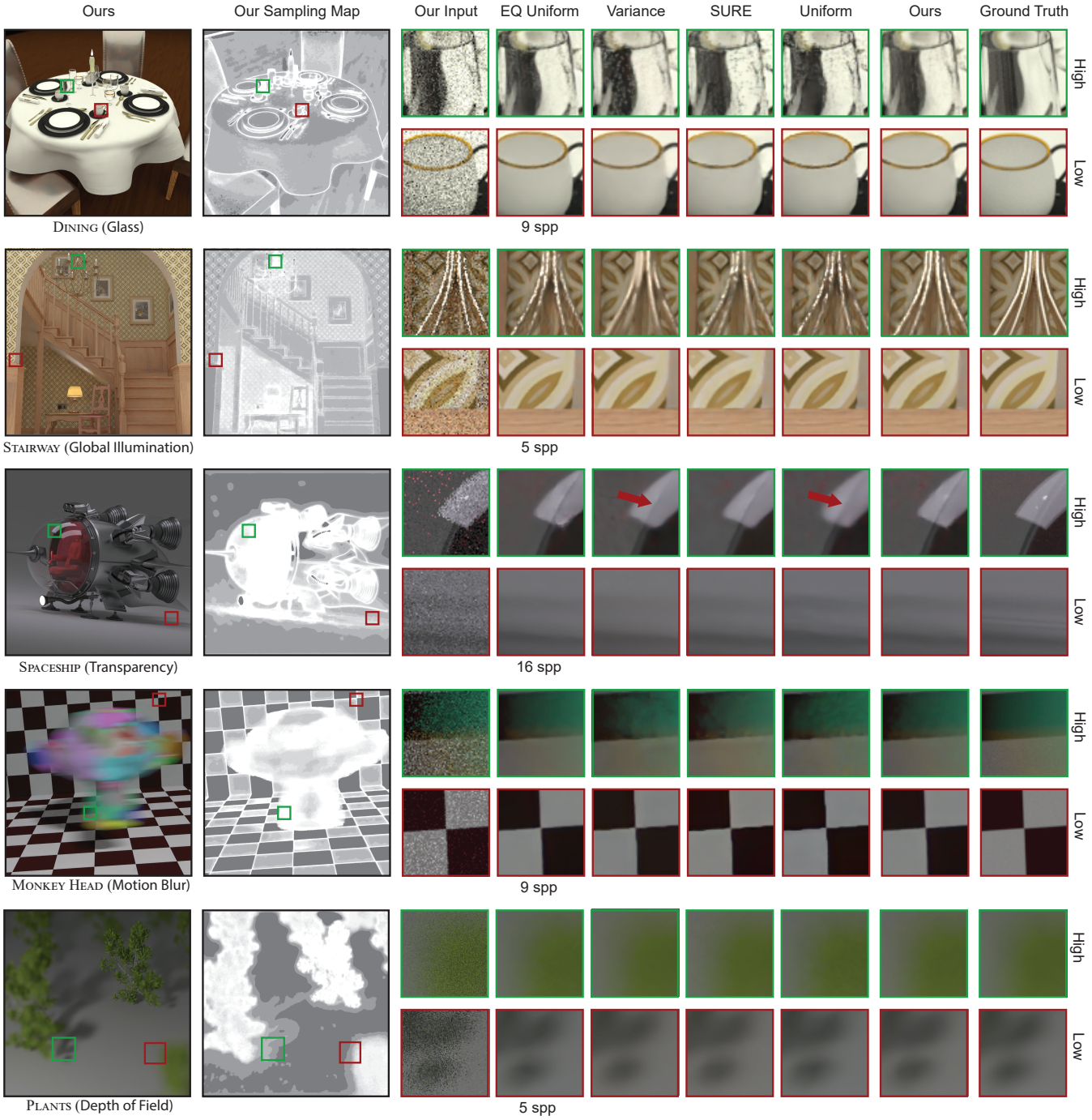

Figure 7: *Comparison of our approach against several other sampling techniques. The number of uniformly distributed samples required for obtaining the same quality as our approach with 4 spp is listed below the insets in the equal quality (EQ) column. For each scene, we show two insets from the high and low sampling density areas. In general, our method produces significantly better results in the top green insets with high adaptive sampling rate, while doing as well as other approaches in the red bottom insets with low adaptive sampling rate. See numerical comparison on these images in Table 3.*

compared to the time required for rendering the input images with Blender's Cycles raytracer (used in our implementation) or even GPU raytracers like Optix. Therefore, our approach provides a significant speedup (50%) compared to uniform sampling, as this approach requires rendering 6 spp to achieve the same quality as ours. Note that, we report sampling counts rather than overall times to avoid referring to a particular system and implementation.

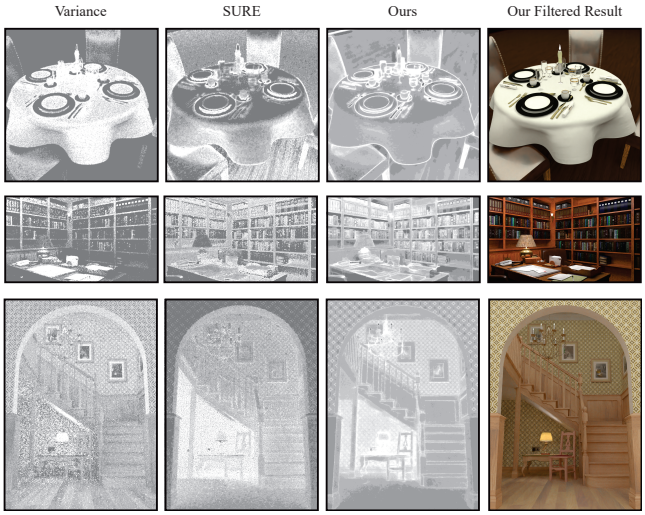

Figure 8: *Comparison of sampling map against other techniques for different scenes.*

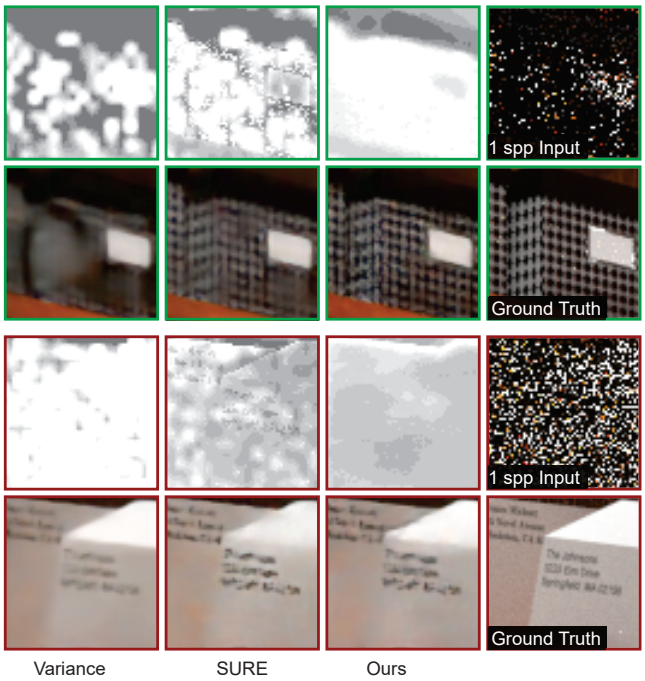

Figure 9: *Comparison of sampling map against other techniques on the insets from Fig. 1.*

Convergence Analysis – For convergence analysis, we rendered multiple batches of 1 spp images. Then we generated the sampling map and rendered 3 additional adaptive samples as usual for each batch. After combining the batches into a single image (the total number of samples is four times the number of batches), we denoise the resulting image. Figure 10 shows how the error continues to decrease as we add more batches. Although our method continues to have the lowest error, the rate of convergence is slightly slower than one of the uniform method.

Limitations – In the cases where most regions are equally difficult for the denoiser, our approach does not produce significantly better results than uniform sampling numerically, e.g., PLANTS scene in Fig. 7 (see Table 3). However, as discussed, our approach improves the visual quality of the results significantly in the vast majority of examples.

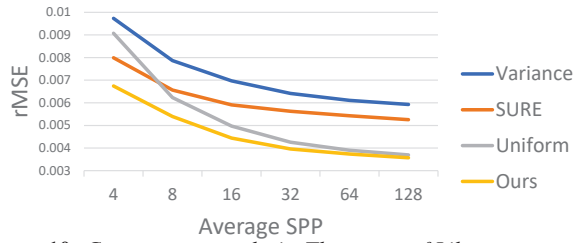

Figure 10: *Convergence analysis. The errors of Library scene with respect to more samples for different methods after denoising.*

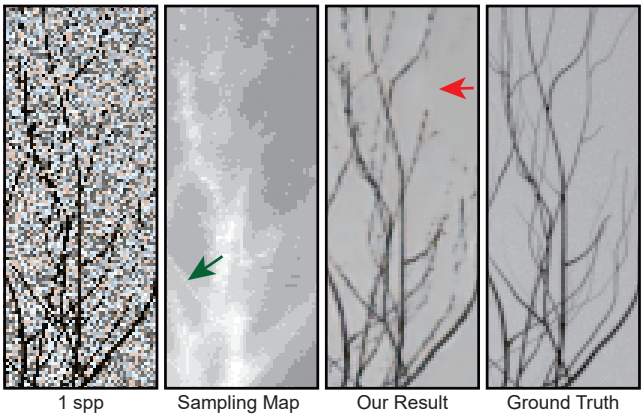

Figure 11: *Our sampling map cannot capture very thin structures such as some branches on this bush shown by the red arrow. However, it still able to capture a branch 2 pixel wide shown by the green arrow.*

Finally, our sampling map network is not able to properly sample the thin structures in noisy areas, as shown in Fig. 11. This is indeed a challenging case as the 1 spp input does not have sufficient information for the sampling map network to detect the regions with thin structures.

7. Conclusions and Future Work

We have presented a novel deep learning approach for joint adaptive sampling and reconstruction of MC rendered images with low sampling counts. We use a convolutional neural network (CNN) to estimate a sampling map from a uniformly sampled 1 spp image. We then use the map to render three additional samples adaptively. Finally, we denoise the rendered image using another CNN to produce the final image. We train both networks on a set of scenes with different distributed effects in an end-to-end fashion. To the best of our knowledge, there are no existing adaptive sampling techniques for extremely low sample counts and the simple alternatives perform significantly worse than our approach and even uniform sampling. Our method is the first learning-based approach that enables adaptive sampling in extremely low sample count MC rendering.

In the future, it would be interesting to use generative adversarial networks to improve the perceptual quality of the results. Moreover, we would like to investigate the possibility of performing joint sampling and denoising at multiple scales to be able to generate high-quality results at the rates lower than 1 sample per pixel. At the other extreme, we would like to investigate learning-based adaptive sampling for higher sample counts in production rendering. Even though standard adaptive sampling approaches can be applied for higher sample counts, there may still be advantages to a learning-based approach.

8. Acknowledgements

This work was funded in part by NSF grant 1451830, Samsung, the Ronald L. Graham endowed Chair, and the UC San Diego Center for Visual Computing. We also acknowledge computing resources from NSF Chase-CI 1730158.

We also would like to thank the following authors for the scenes used in the paper: ThePefDispenser, Mikel007, Wig42, thecali, Greg Zaal, GJ2012, Ali Gonzalez, Jay-Artist and Bhavin Solanki.

References

[BEEM15] BAUSZAT P., EISEMANN M., EISEMANN E., MAGNOR M.: General and robust error estimation and reconstruction for monte carlo rendering. *Computer Graphics Forum (Proc. Eurographics) 34*, 2 (jan 2015). 2

[BEM11] BAUSZAT P., EISEMANN M., MAGNOR M.: Adaptive sampling for geometry-aware reconstruction filters. In *Proc. Vision, Modeling and Visualization (VMV)* (Oct 2011), Eurographics, pp. 183–190. 3

[BM98] BOLIN M. R., MEYER G. W.: A perceptually based adaptive sampling algorithm. In *SIGGRAPH 98* (1998), pp. 299–309. 2

[BVM*17] BAKO S., VOGELS T., MCWILLIAMS B., MEYER M., NOVÁK J., HARVILL A., SEN P., DEROSE T., ROUSSELLE F.: Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM TOG 36*, 4 (July 2017). 3

[CKS*17] CHAITANYA C. R. A., KAPLANYAN A. S., SCHIED C., SALVI M., LEFOHN A., NOWROUZEZAHRAI D., AILA T.: Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM TOG 36*, 4 (July 2017), 98:1–98:12. 2, 3

[CPC84] COOK R., PORTER T., CARPENTER L.: Distributed Ray Tracing. In *SIGGRAPH 84* (1984), pp. 137–145. 2

[ETH*09] EGAN K., TSENG Y., HOLZSCHUCH N., DURAND F., RAMAMOORTHI R.: Frequency analysis and sheared reconstruction for rendering motion blur. *ACM TOG 28*, 3 (2009). 2

[Guo98] GUO B.: Progressive radiance evaluation using directional coherence maps. In *SIGGRAPH 98* (1998), pp. 255–266. 2

[HJW*08] HACHISUKA T., JAROSZ W., WEISTROFFER R., DALE K., HUMPHREYS G., ZWICKER M., JENSEN H.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM TOG 27*, 3 (2008). 2

[Kaj86] KAJIYA J.: The Rendering Equation. In *SIGGRAPH 86* (1986), pp. 143–150. 2

[KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014). 5

[KBS15] KALANTARI N., BAKO S., SEN P.: A machine learning approach for filtering monte carlo noise. *ACM TOG 34*, 4 (2015). 3

[KFF*15] KELLER A., FASCIONE L., FAJARDO M., GEORGIEV I., CHRISTENSEN P., HANIKA J., NICHOLS G., EISENACHER C.: The path-tracing revolution in the movie industry. SIGGRAPH 2015 Course Notes, 2015. 2

[KS13] KALANTARI N., SEN P.: Removing the noise in Monte Carlo rendering with general image denoising algorithms. *CGF 32*, 2 (2013). 2

[LRU85] LEE M., REDNER A., USELTON S.: Statistically optimized sampling for distributed ray tracing. In *SIGGRAPH 85* (1985), pp. 61–68. 2, 3, 6

[LWC12] LI T., WU Y., CHUANG Y.: SURE-based optimization for adaptive sampling and reconstruction. *ACM TOG 31*, 6 (2012). 2, 3, 6

[MCY14] MOON B., CARR N., YOON S.: Adaptive rendering based on weighted local regression. *ACM TOG 33*, 5 (2014). 2

[MGYM15] MOON B., GUITIAN J., YOON S., MITCHELL K.: Adaptive rendering with linear predictions. *ACM TOG 34*, 4 (2015). 2

[Mit87] MITCHELL D.: Generating antialiased images at low sampling densities. In *SIGGRAPH 87* (1987), pp. 65–72. 2

[Mit91] MITCHELL D.: Spectrally Optimal Sampling for Distribution Ray Tracing. In *SIGGRAPH 91* (1991), pp. 157–164. 2

[MMMG16] MOON B., MCDONAGH S., MITCHELL K., GROSS M.: Adaptive polynomial rendering. *ACM TOG 35*, 4 (2016). 2

[MWR12] MEHTA S., WANG B., RAMAMOORTHI R.: Axis-aligned filtering for interactive sampled soft shadows. *ACM TOG 31*, 6 (2012). 3

[MYRD14] MEHTA S., YAO J., RAMAMOORTHI R., DURAND F.: Factored axis-aligned filtering for rendering multiple distribution effects. *ACM TOG 33*, 4 (2014). 3

[ODR09] OVERBECK R., DONNER C., RAMAMOORTHI R.: Adaptive Wavelet Rendering. *ACM TOG 28*, 5 (2009). 2

[PGC*17] PASZKE A., GROSS S., CHINTALA S., CHANAN G., YANG E., DEVITO Z., LIN Z., DESMAISON A., ANTIGA L., LERER A.: Automatic differentiation in pytorch. 6

[RB11] RAVISHANKAR S., BRESLER Y.: Mr image reconstruction from highly undersampled k-space data by dictionary learning. *IEEE Transactions on Medical Imaging 30*, 5 (May 2011), 1028–1041. 6

[RKZ11] ROUSSELLE F., KNAUS C., ZWICKER M.: Adaptive sampling and reconstruction using greedy error minimization. *ACM TOG 30*, 6 (Dec. 2011), 159:1–159:12. 2

[RKZ12] ROUSSELLE F., KNAUS C., ZWICKER M.: Adaptive rendering with non-local means filtering. *ACM TOG 31*, 6 (Nov. 2012), 195:1–195:11. 2

[RMZ13] ROUSSELLE F., MANZI M., ZWICKER M.: Robust denoising using feature and color information. *CGF 32*, 7 (2013), 121–130. 2

[SKW*17] SCHIED C., KAPLANYAN A., WYMAN C., PATNEY A., CHAITANYA C. R. A., BURGESS J., LIU S., DACHSBACHER C., LEFOHN A., SALVI M.: Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In *HPG* (2017), ACM, pp. 2:1–2:12. 2

[SSD*09] SOLER C., SUBR K., DURAND F., HOLZSCHUCH N., SILLION F.: Fourier depth of field. *ACM TOG 28*, 2 (2009). 2

[WYKR17] WU L., YAN L., KUZNETSOV A., RAMAMOORTHI R.: Multiple axis-aligned filters for rendering of combined distribution effects. *CGF 36*, 4 (2017), 155–166. 3

[YMRD15] YAN L., MEHTA S., RAMAMOORTHI R., DURAND F.: Fast 4D sheared filtering for interactive rendering of distribution effects. *ACM TOG 35*, 1 (2015). 3

[ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHI R., ROUSSELLE F., SEN P., SOLER C., YOON S.: Recent advances in adaptive sampling and reconstruction for monte carlo rendering. *Computer Graphics Forum (EUROGRAPHICS STAR 2015) 34*, 2 (2015), 667–681. 2