DOI: 10.1111/cgf.14454
EUROGRAPHICS 2022 / R. Chaine and M. H. Kim
(Guest Editors)
Volume 41 (2022), Number 2

Progressive Denoising of Monte Carlo Rendered Images

Arthur Firmino[1,2], Jeppe Revall Frisvad[2], and Henrik Wann Jensen[1]

[1]Luxion
[2]Technical University of Denmark

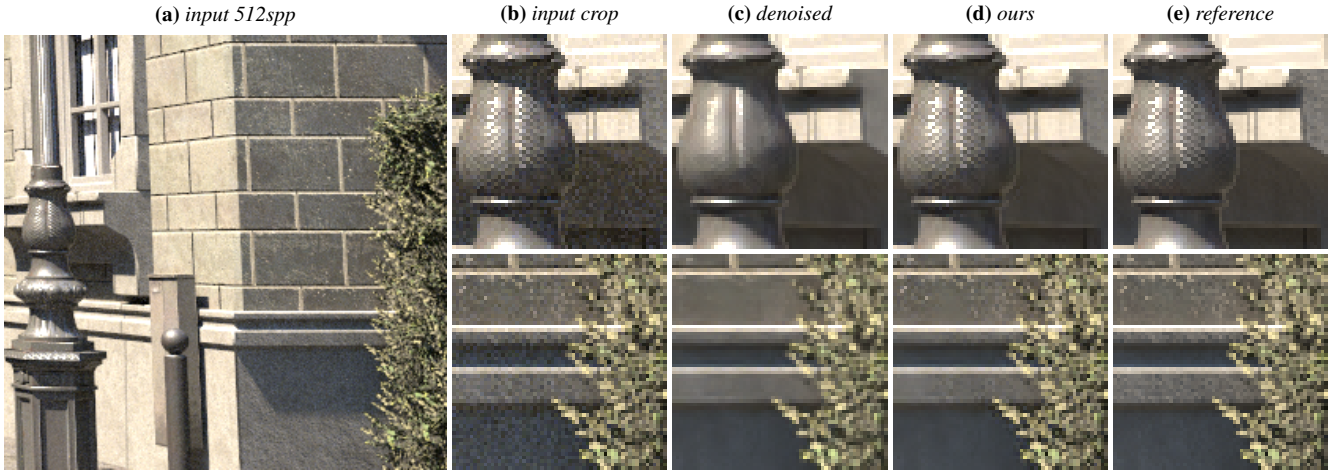

Figure 1: *Loss of detail when denoising, (b) to (c), and subsequent recovery with our approach (d). By incorporating error estimates of (b) and (c), our method is able to infer a per-pixel mixing parameter used to interpolate between the two images and reduce the overall error. Here we used Intel® Open Image Denoise (OIDN) to denoise the image, albedo and normal auxillary features included.*

Abstract

Image denoising based on deep learning has become a powerful tool to accelerate Monte Carlo rendering. Deep learning techniques can produce smooth images using a low sample count. Unfortunately, existing deep learning methods are biased and do not converge to the correct solution as the number of samples increase. In this paper, we propose a progressive denoising technique that aims to use denoising only when it is beneficial and to reduce its impact at high sample counts. We use Stein's unbiased risk estimate (SURE) to estimate the error in the denoised image, and we combine this with a neural network to infer a per-pixel mixing parameter. We further augment this network with confidence intervals based on classical statistics to ensure consistency and convergence of the final denoised image. Our results demonstrate that our method is consistent and that it improves existing denoising techniques. Furthermore, it can be used in combination with existing high quality denoisers to ensure consistency. In addition to being asymptotically unbiased, progressive denoising is particularly good at preserving fine details that would otherwise be lost with existing denoisers.

CCS Concepts

• ***Computing methodologies*** → ***Image processing; Rendering; Ray tracing;***

1. Introduction

Monte Carlo (MC) light transport algorithms, such as path tracing, have been ubiquitous since the introduction of the rendering equation in 1986 [Kaj86]. However, the nature of these algorithms is such that many samples may be required to reach an acceptable level of variance. While research on variance reduction techniques has continuously progressed, achieving noise-free images directly from MC rendering algorithms is often still prohibitively expensive and time consuming. This is true despite the advent of more powerful hardware.

To ameliorate this problem, denoising techniques are employed as a post-processing step to remove any apparent residual noise.

Improvement in quality of denoisers was one of the decisive factors leading to the path tracing revolution in the movie industry [CJ16]. At the time, use of features such as per-pixel normals, motion vectors, and surface albedo as well as error estimated using Stein's unbiased risk estimate (SURE) was a key to obtain better denoisers [LWC12, RMZ13, ZJL*15]. In recent years, denoisers have further improved in quality due to the use of neural networks and machine learning [HY21]. Learned denoisers are not necessarily perfect, however, and may suffer from bias that is sufficient to be perceived as an objectionable amount of blurred details.

To preserve details during denoising, Vogels et al. [VRM*18] proposed use of an asymmetric loss function when training. This allows for artistic control between aggressive and conservative denoising. Back et al. [BHHM20] proposed another deep learning method where they combine independent and correlated estimates (e.g. MC rendered and denoised images). Even more recently, an optimization-based technique to combine an ensemble of denoised images has been put forward [ZZXY21]. Combining an ensemble of denoised images requires significant processing time and is certainly intended to serve as a post-processing step. As mentioned by Christensen et al. [CFS*18] in their description of Pixar's RenderMan, it would be better to denoise images during progressive rendering to enable faster decision making.

We suggest a method for *progressive denoising* where a neural network learns to infer an optimal per-pixel mixing parameter given two input images as well as estimates of their per-pixel error. In practice, one of these images is a MC rendered image and the other its denoised counterpart. The error estimates are of their squared error, using the estimated variance of the sample mean for the former and SURE for the latter. We demonstrate that our proposed solution substantially improves quality when applied onto existing denoisers, and that it is asymptotically unbiased in the limit of many samples. This is demonstrated in comparisons and in application with existing pre-trained high quality denoisers.

Inspired by an earlier use of confidence intervals for adaptive sampling in MC rendering [Pur87, TJ97], we use confidence intervals based on the Student's t distribution to bound the mixing parameter received from our network. This ensures that the rendered image will converge to the correct solution as the number of samples increases. With the bounded mixing parameter, our method enables automatic selective denoising on a per-pixel basis during a progressive rendering. In this way, details in the image that the denoiser initially blurs out will gradually appear and become crisp in the image as more samples are progressively added.

Overview. The overall flow of our method is to get the variance of the rendered image, denoise it and estimate the error in the denoised image using SURE. The two images (rendered and denoised) and their respective error estimates serve as input for our network, which finds three parameters per pixel for a theoretically justified activation function (f_{act}). The activation function calculates a mixing parameter α used for per pixel linear interpolation of the rendered image and the denoised image. To ensure convergence as the number of samples increases, we use a t-statistic based on the variance in a neighborhood of pixels to bound α (see Figure 6 in Section 4 for a visual overview). The assumptions made are that the radiance values are normally distributed. This is important for the accuracy of SURE, and for the meaningfulness of the t-statistic.

2. Related Work

With Cook's introduction of stochastic sampling into the rendering process [CPC84, Coo86], the scene was set for research on noise reduction in Monte Carlo rendered images. Some of the first papers on denoising of rendered images [LR90, RW94] describe well how noise in MC rendered images tends to be different from the noise observed in signal processing and computer vision. We refer to denoising for MC rendered images as Monte Carlo denoising.

2.1. Monte Carlo Denoising

Monte Carlo denoising has been a vital component of the rendering process since the widespread use in industry of Monte Carlo rendering algorithms [KCK*18, BAC*18, CFS*18, KCSG18]. The first approaches relied on local non-linear filtering [LR90, RW94, JC95], and it is generally recognized that for denoising of rendered images the filter needs to locally adapt to the noise level of every pixel [KS13]. To obtain more information for per pixel adaptation, denoisers started taking advantage of auxiliary features (e.g. surface normals and albedo) by employing cross-bilateral [SD12] and higher-order filters [BRM*16]. Our method continues the trend of per-pixel adaptation, but we use our error estimates for selective application of any preferred denoising technique.

Deep learning has been used for predicting both the parameters of classical filters [KBS15] and the filtering kernel itself [BVM*17]. A less restrictive approach has been to directly predict a final radiance value, and here a variety of methods have been proposed including the use generative adversarial networks [XZW*19], residual networks [WW19], and autoencoders [CKS*17]. Most learning based techniques also support inclusion of auxiliary features as input.

One drawback of deep learning based MC denoisers is the lack of control over the amount of perceived blurring, detail preservation, and variance-bias trade off. To this end, Vogels et al. [VRM*18] introduce in their denoiser asymmetric loss functions controlled by a parameter that can be specified at runtime. This allows artists to modulate between conservative denoising with some details contextually preserved and more aggressive denoising with little noise but more blurring. Such an approach is beneficial in their setting where fine artistic control is desired for each rendered frame. Instead, our method lets an artificial neural network determine the extent to which the denoising should be applied in every pixel.

2.2. Combining Denoised Estimates

Recent work has emerged with the idea of using denoising not as the final product, but as the penultimate step before combining denoising results and other estimates to achieve better results [BHHM20, ZZXY21]. This is promising in particular for learning-based denoisers, because while they often excel at low sample counts they suffer from bias that is detrimental at higher sample counts leading to a loss of consistency.

Back et al. [BHHM20] propose a method of combining the original and the denoised images in order to improve the quality of the final image. Similar to our method, they leverage neural networks for the task, but unlike them we base our approach on computed error estimates of the input images which are included as inputs to our network. Furthermore, our method computes a per-pixel mixing parameter, while they apply a predicted kernel to the input color images. Lastly, we guarantee our method converges to the ground truth image in the limit of many samples, while they do not.

More recently, Zheng et al. [ZZXY21] have put forward an optimization-based technique for combining the outputs of multiple MC denoisers. Like ours, their approach is based on computing mean squared error (MSE) estimates of the denoised images and while they use a dual-buffer strategy for these estimates, we opt to use SURE instead. In contrast to their work, ours does not require application of multiple denoisers or support for rendering half-sample count images. Furthermore, our method is inherently consistent and does not rely on the choice of denoiser.

2.3. Stein's Unbiased Risk Estimate

Proposed by Stein in 1981 [Ste81], SURE estimates the MSE of an estimator of the mean of a normally distributed multivariate random variable. Stein states that if $x \in \mathbb{R}^d$ is a measured vector, distributed normally with unknown mean $\mu \in \mathbb{R}^d$ and known covariance $\Sigma \in \mathbb{R}^{d\times d}$, and $F(x)$ is an estimator of μ that is at least weakly differentiable, then

$$\mathrm{SURE}(F,x) = \frac{1}{d}\left(\|F(x)-x\|^2 + 2\mathrm{tr}(J_F(x)\cdot\Sigma) - \mathrm{tr}(\Sigma)\right) \qquad (1)$$

is an unbiased estimate of the mean squared error of $F(x)$, where $J_F(x)$ is the Jacobian matrix of F at x and tr is the trace. If Σ is diagonal, or equal to $\sigma^2 I$, then the equation reduces to its more commonly seen forms.

SURE has appeared multiple times in computer graphics literature, being used to guide adaptive sampling and to optimize parameters of denoising functions [LWC12, RMZ13, CFS*18, XC20].

3. Denoising Error Estimation

In this section, we detail our initial investigation into the error properties of denoised images. We begin with an empirical evaluation of denoised images by comparing them with ground-truth images, and end with a practical way of estimating the mean squared error (MSE) of these images using SURE.

3.1. Evaluating Error of Denoised Images

Images were rendered in *pbrt-v4* [PJH16] for a collection of scenes featuring a variety of different objects, materials, and lighting environments. We then denoised these images using Intel® Open Image Denoise (OIDN) [oid] and compared to ground-truth images rendered at very high samples-per-pixel (spp) counts, 65K or higher. We computed various error metrics and compared images qualitatively as well. Comparing with images denoised using the NVIDIA OptiX™ AI-Accelerated Denoiser [CKS*17] and Radeon™ Image Filter denoiser [rad], we found that the OIDN denoiser generally outperformed the others in terms of image quality.

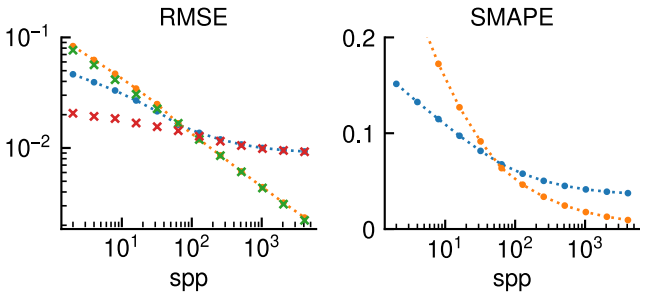

Figure 2: *Plot of the actual error (orange and blue) and estimated error (green and red) computed using variance for input images (green) and SURE for denoised images (red), and using the San Miguel outdoor scene which is also in the bottom row of Figure 9. RMSE: root mean squared error (left). SMAPE: symmetric mean absolute percentage error (right).*

For a handful of scenes, we found denoising introduced noticeable error and often the denoised image appeared to have less visual fidelity than the input, even when the input had a modest sample count of around a few hundred samples per pixel. This was still the case, albeit to a lesser degree, when auxiliary features (albedo and normals) were included in the denoising input. In common between these scenes was the presence of high-frequency details, such as from vegetation or bump maps, and a resultant blurring from denoising, an example of which is illustrated in Figure 1.

Plotting the root mean squared error (RMSE) of the rendered and denoised images with log axis scaling, we observed a common pattern, which is exemplified in Figure 2. Compared with the rendered input image, the denoised image has significantly less error at very low sample counts, but as the sample count increases, denoising results in diminishing improvement until eventually the rendered input image has less error than its denoised counterpart. For some scenes, this happens only at very high sample counts (greater than 10K), while for the handful of scenes previously mentioned this can require only a few hundred samples or less.

3.2. Error Estimation with SURE

To make an *a priori* estimate of the MSE of denoised images, without knowledge of the ground-truth image, we resort to SURE. For simplicity, we assume the sample covariance matrix to be diagonal, meaning that there are no correlations between variables. This assumption is true except between RGB channels of the same samples, but our initial investigation showed this misassumption to have no impact at high sample counts.

As we have no closed-form expression for the partial derivatives of the denoiser $F(x)$, we apply a Monte Carlo SURE first-order approximation [RBU08] to estimate $\mathrm{tr}(J_F(x)\cdot\Sigma)$ in Eq. 1. This is

$$\mathrm{tr}(J_F(x)\cdot\Sigma) \approx \frac{1}{\varepsilon K}\sum_{k=1}^{K} b_k^T\left(F(x+\varepsilon b_k) - F(x)\right), \qquad (2)$$

where b_k are normally distributed random vectors with mean zero and covariance Σ, ε is some small positive number (e.g. 10^{-4}), and K is the number of MC iterations (1 or 4 in our work).

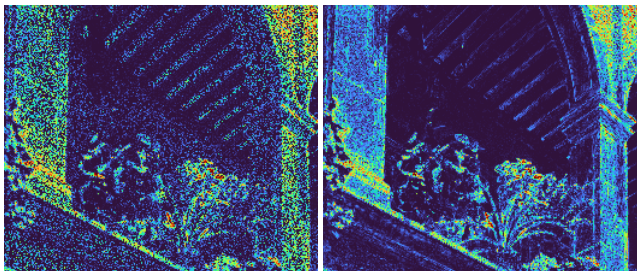

Figure 3: *Per-pixel squared error estimate of a denoised image using SURE (left), and its actual squared error (right).*

As SURE is an unbiased estimator, it produces an accurate estimate of the whole image MSE as illustrated in Figure 2. The accuracy of the estimate increases with the sample count, and this is likely due to the distribution of the sample mean approaching a normal distribution, as concluded by Li et al. [LWC12].

The variance of SURE is generally too high to make accurate estimates of the squared error of individual pixels, an example of which is shown in Figure 3. Despite this we note that local averages are visibly well correlated with the actual error of the denoised images, and that these estimates, when compared with variance estimates of the input, could form the basis of a decision between the pixels of input and denoised images.

4. Progressive Denoising

Our goal of *progressive denoising* can be summarized as attempting to solve the following problem: Let $x \sim \mathbb{N}(\mu, \frac{\sigma^2}{n})$ be an image produced by MC rendering with n samples, also an unbiased estimate of the ground truth image μ. We assume it to be normally distributed and this assumption is true in the limit of n going to infinity, provided σ^2 is finite. Let y be the denoised counterpart of the image x and also a biased estimate of μ:

$$y = f(x, G_x; \Theta_f),$$

where the denoising function f is parametrized by Θ_f, and G_x are auxiliary image features of x such as albedo and normals. Let $z = x + \alpha(y - x)$, where α is a per-pixel mixing parameter, be a linear interpolation of x and y. We wish to find a function that outputs the per-pixel mixing parameter:

$$\alpha = h(x, y, \ldots; \Theta_h) \tag{3}$$

and solves the following constrained optimization problem:

$$\arg\min_{\Theta_h} \mathcal{L}(z, \mu) \quad \text{subject to} \quad \lim_{n\to\infty} z = \mu \tag{4}$$

where $\mathcal{L}$ is some loss or error function, and Θ_h are the parameters of the function h being sought.

4.1. Theoretically Optimal Solution

An expression for α (in the one-dimensional case) that minimizes the expectation of the squared error of z, can be derived by following the same principles as SURE, such as assuming the estimator f to be at least weakly differentiable. In the one-dimensional case, covariance Σ is variance σ^2 and the SURE relation (1) becomes $\text{SURE} = \|y - x\|^2 + 2\sigma^2 f'(x) - \sigma^2$, for which the expected value is $\mathbb{E}[\text{SURE}] = \mathbb{E}[\|y - \mu\|^2]$. An expression for the squared error when using the mixing parameter α is then

$$\begin{aligned}
&\mathbb{E}[\|x + \alpha(y - x) - \mu\|^2] \\
&= \mathbb{E}[\|x - \mu + \alpha(y - \mu) - \alpha(x - \mu)\|^2] \\
&= \mathbb{E}[\|x - \mu\|^2] + \alpha^2 \mathbb{E}[\|y - \mu\|^2] + \alpha^2 \mathbb{E}[\|x - \mu\|^2] \\
&\quad + (2\alpha - 2\alpha^2)\mathbb{E}[(x - \mu)(y - \mu)] - 2\alpha\mathbb{E}[\|x - \mu\|^2] \\
&= \sigma^2(1 + \alpha^2 - 2\alpha) + \alpha^2 \mathbb{E}[\text{SURE}] \\
&\quad + 2(\alpha - \alpha^2)\mathbb{E}[(x - \mu)(y - \mu)], \qquad (5)
\end{aligned}$$

where we have the variance $\sigma^2 = \mathbb{E}[\|x - \mu\|^2]$ since μ is the ground truth image (which is the expected value of an unbiased MC rendering). We then apply Stein's Lemma [Ste81] to rewrite the last term:

$$\begin{aligned}
\mathbb{E}[(x - \mu)(y - \mu)] &= \mathbb{E}[y(x - \mu)] - \mu\mathbb{E}[x - \mu] \\
&= \mathbb{E}[f(x)(x - \mu)] = \sigma^2 \mathbb{E}[f'(x)]. \qquad (6)
\end{aligned}$$

Substituting Eq. 6 into 5 eliminates the unknown μ, yielding

$$\begin{aligned}
&\mathbb{E}[\|x + \alpha(y - x) - \mu\|^2] \qquad (7) \\
&= \sigma^2(1 + \alpha^2 - 2\alpha) + \alpha^2 \mathbb{E}[\text{SURE}] + 2(\alpha - \alpha^2)\sigma^2 \mathbb{E}[f'(x)]. \qquad (8)
\end{aligned}$$

Taking the derivative with respect to α and equating this to 0, we find the following solution for α by employing the SURE relation:

$$\alpha = \frac{\sigma^2 - \sigma^2 \mathbb{E}[f'(x)]}{\mathbb{E}[\text{SURE}] + \sigma^2 - 2\sigma^2 \mathbb{E}[f'(x)]} = \frac{\sigma^2 - \sigma^2 \mathbb{E}[f'(x)]}{\|y - x\|^2}. \tag{9}$$

If we swap x with y, we find a similar solution for $y + \beta(x - y)$:

$$\beta = \frac{\text{SURE} - \sigma^2 \mathbb{E}[f'(x)]}{\|y - x\|^2}. \tag{10}$$

It is difficult to apply Eqs. 9 and 10 directly, due to the high degree of noise in the estimates of σ^2, SURE and $\mathbb{E}[f'(x)]$ as exemplified by Figure 3. Despite these equations not being used directly in our work, we extract two interesting properties from them:

1. The shared role of σ^2 and SURE in calculating α and β.
2. The non-linear relationship between the quantities involved.

In Section 4.3, we detail how we incorporate these properties into our neural network and its training, and in Section 6.1 we demonstrate their impact.

4.2. Practical Solution

Our approach to constructing a function $h(x, y, \ldots; \Theta_h)$ that solves the constrained optimization problem of Eq. 4 can be broken down into two steps. The first step is aimed at tackling the minimization part of the problem, and the second step is directed at the limit constraint.

We begin by training a neural network to learn a function h_{NN} that predicts a preliminary per-pixel mix between the MC rendered and denoised images, so that

$$z = x + \alpha(y - x), \quad \alpha = h_{\text{NN}}(x, y, \ldots; \Theta_h). \tag{11}$$

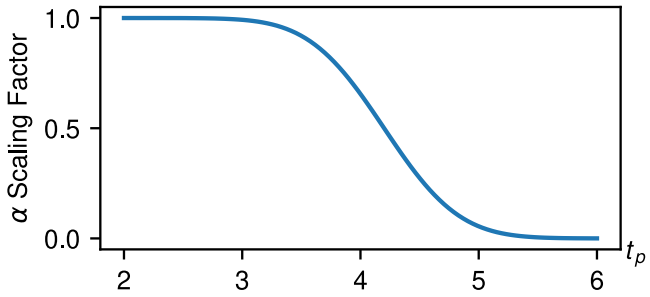

Figure 4: *Scaling of the mixing parameter α, Eq. 18.*

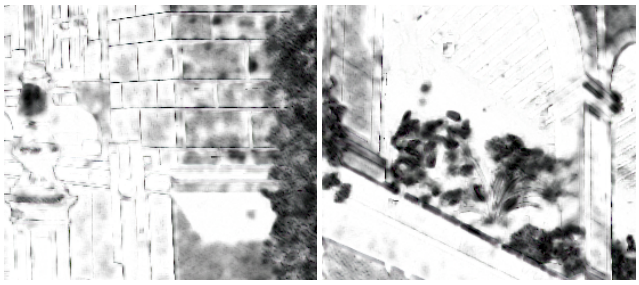

Figure 5: *Examples of the per-pixel mixing parameter α′, the result of Eq. 18. The images show α′ for Figures 1 and 3 (left and right).*

We discuss the choice of network architecture for h_{NN}, its inputs, and details regarding its training in Section 4.3.

The preliminary value for z, while assumed to fulfil the minimization part of the problem, does not guarantee satisfaction of the limit constraint. Fortunately, since in the limit of many samples we know x becomes normally distributed, we can apply statistical tests and compute quantiles estimating confidence intervals for the outputs of the network. In early work [Pur87, TJ97], confidence intervals were used for adaptive sampling in MC rendering. In a sense, we suggest to use confidence intervals for adaptive denoising.

We compute the weighted average of the non-denoised radiance values x within each pixel's 11-by-11 neighbourhood $\mathcal{N}_p$, the variance of this weighted average, and the weighted average of the mixed radiance values from the network as follows:

$$\bar{x}_p = x_{M_p} + \sum_{q\in\mathcal{N}_p} \kappa_{p,q} x_q, \tag{12}$$

$$\mathrm{Var}[\bar{x}_p] = \mathrm{Var}[x_{M_p}] + \sum_{q\in\mathcal{N}_p} \kappa_{p,q}^2 Var[x_q] \tag{13}$$

$$\bar{z}_p = z_{M_p} + \sum_{q\in\mathcal{N}_p} \kappa_{p,q}(x_q + \alpha_q(y_q - x_q)) \tag{14}$$

$$\kappa_{p,q} = \frac{1}{\sigma_s\sqrt{2\pi}} \exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}\right) \tag{15}$$

$$M_p = \arg\max_{q\in\mathcal{N}_p} \mathrm{Var}[x_q]\,. \tag{16}$$

To compensate for insufficient accuracy of the sample variance estimates at low sample counts, we use the index M_p to weigh the average towards the neighbouring pixel with the highest variance. For the filter parameter σ_s, we chose $\min(\sqrt{Var[\bar{x}]}, 1)\times 10^2$, where $\bar{x}$ is the image mean.

These values are used to compute the t-statistic, which corresponds to a quantile estimating the confidence interval of the radiance values predicted by the network. Applying a sigmoid function S to the magnitude of the t-statistic, as if we wanted to estimate the significance of the predicted radiance values, we find a scaling factor for α that we use to get our final mixing parameter α'. This ensures that if the estimated confidence interval is large, α' is reduced, so that we in this case rely more on the original output from the MC rendering than on the denoised version. This is done as follows:

$$t_p = \frac{\bar{z}_p - \bar{x}_p}{\sqrt{\mathrm{Var}[\bar{x}_p] + \varepsilon}} \tag{17}$$

$$\alpha'_p = (1 - S[a(|t_p| - c)])\,\alpha_p\,, \tag{18}$$

where we use the cumulative distribution function (CDF) of the standard normal distribution as S while $c = 4.2$ is a shift parameter and $a = 2$ is a strength parameter, the values of which were determined via a parameter search over the training dataset. This parameter search was not crucial. Our motivation for doing so was to avoid optimizing our method for the test dataset. Another option for S is the logistic function, which seems to work equally well. The α scaling factor is plotted as a function of t_p in Figure 4. When this significance-related value is close to one, the network likely found a radiance value that is not attributable to random noise. We therefore let the scaling of α follow this value.

The scaling of the mixing parameter does not take much effect until the magnitude of the quantile $|t_p|$ exceeds $c - 3/a$. Since the values are weighted averages of many pixels, $Var[\bar{x}_p]$ is small and therefore the predicted value need not deviate a lot for scaling to apply. As the sample count increases, the necessary deviation becomes smaller guaranteeing convergence to the ground truth. Figure 5 shows two examples of the scaled mixing parameter.

4.3. Network Architecture

The function $h_{\mathrm{NN}}(x, y, \ldots; \Theta_h)$ consists of a small residual convolutional neural network, 6 layers deep, each layer with 48 feature channels, 3×3 kernels (5×5 for the first layer), ReLU activation, and using 'same' padding. See Figure 6. Two *residual blocks* [HZRS16] form part of our network, a similar network composition was used by Vogels et al. [VRM*18].

To mirror the theoretical results in Section 4.1, our network does not directly predict α. Instead, the network outputs three values (x_1, x_2, x_3) per pixel and per channel that are passed into the following non-linear function that mimics Eqs. 9 and 10:

$$f_{\mathrm{act}}(x_1, x_2, x_3) = \frac{x_1 - x_2}{\max(x_3, \varepsilon)}, \tag{19}$$

where $\varepsilon = 10^{-6}$ is to avoid the singularity. We demonstrate in our experiments (Section 6.1) that this approach has a positive impact when applied to the network's output.

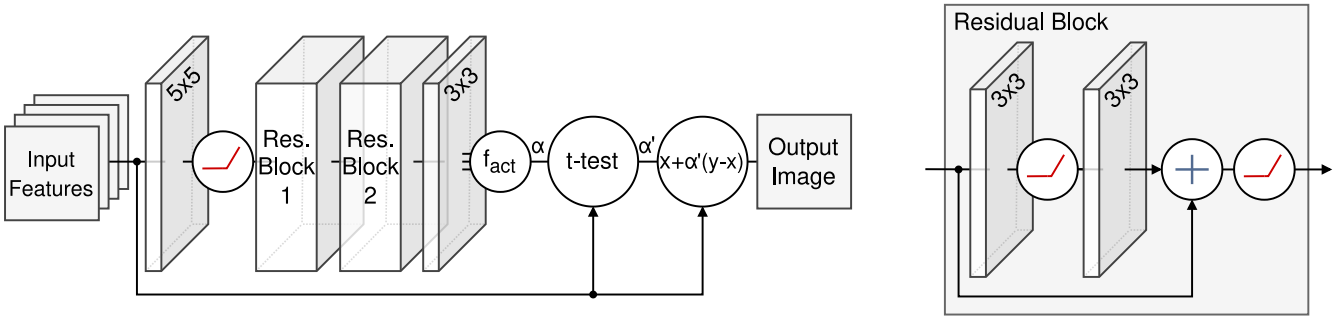

Figure 6: *Overview of our network $h_{NN}(x,y,\ldots;\Theta_h)$ for mixing an unbiased MC rendering x with its denoised counterpart y using per-pixel mixing parameters α'. The initial 5×5 convolutional layer is followed by two residual blocks and a final convolutional layer. ReLU activation follows all but the final convolution, which uses f_{act} (see Sections 4.1 and 4.3). Input features include the rendered and denoised images, along with estimates of their error (Section 4.3). The use of residual blocks is inspired by the networks of Vogels et al. [VRM*18].*

As inputs for the network, we have the MC rendered and denoised radiances x and y, also abbreviated as HDR and DEN, as well as the estimates of their squared error: VAR and SURE, respectively. We also include input features corresponding to the values of $\sigma^2 f'(x)$ and $\|y-x\|^2$, which are calculated when computing SURE. The non-radiance quantities are transformed by $\arctan(x)$, rather than $\log(1+x)$ as x may be negative, to ensure that those network inputs are bounded.

During training we randomly swap HDR, VAR with DEN, SURE and vice versa, effectively swapping the above calculation for α to one for $\beta = 1-\alpha$. This is to ensure our training does not converge on a model whose outputs always compute a value of 1 for α, as the denoised input (DEN) typically has lower error. We posit that when combined with the network output's non-linear function, this may lead the network to learn quantities in its output that are invariant to the swapping. We have found that when the random swapping is not performed, the networks being trained failed to converge to the desired solution.

5. Dataset and Training Procedure

5.1. Denoiser Model

The network described in Section 4.3 takes a denoised image and its SURE as input. So, aside from using a pre-trained denoiser from OIDN, we also train our own models for comparison. These are based on the network architecture of Chaitanya et al. 2017 [CKS*17] as implemented in the OIDN training toolkit, but trained from scratch [oid]. This architecture consists of a U-Net with 4 encoder and decoder stages using 2×2 max pooling and nearest-neighbour upsampling, and convolutional layers with 3×3 kernels and ReLU activation.

As inputs to our denoisers we always include radiance (HDR), and optionally include diffuse albedo (ALB), shading normals (NRM), and variance of the radiance channels (VAR). Our variance is computed during rendering by first estimating the sample variance using an online approach, and then dividing by the number of samples to get the variance of the sample mean. Radiance and albedo quantities are transformed by $\log(1+x)$, and variance by $\arctan(x)$.

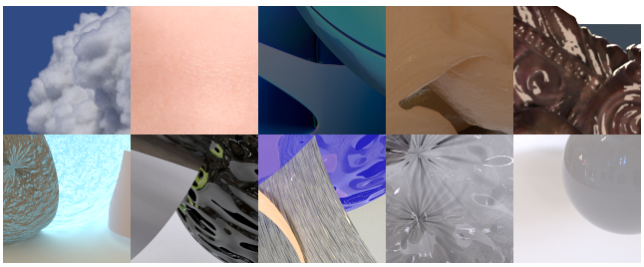

Figure 7: *Example images from the training dataset. Top row: Crops generated from available pbrt-v4 scenes. Bottom row: Randomly generated scenes rendered in mitsuba.*

Figure 8: *The test dataset used in our experiments.*

5.2. Dataset Generation

Our training dataset consists of 1348 unique 128×128 crops from a collection of scenes, rendered using between 2 and 4096 samples per pixel, for a grand total of 11150 images. Ground-truth counterparts were rendered with 32k or 131k samples per pixel. 5% of the dataset was set aside as validation during training. Of the 1348 unique crops, 844 originate from publicly available *pbrt* scenes, while the remaining 504 were generated procedurally and rendered in *mitsuba* [NDVZJ19]. Figure 7 shows examples of some of the images in our training dataset. For our test dataset, we rendered 10 scenes, shown in Figure 8, from among publicly available *pbrt* scenes (not overlapping with scenes used in the training dataset) at sample counts of between 32 and 8192 samples per pixel.

inputs	model	error		
		RMSE	SMAPE	FLIP
HDR	den	0.1964	0.0344	0.0816
	pro-den	0.0784	0.0284	0.0726
HDR, VAR	den	0.1022	0.0292	0.0736
	pro-den	0.0587	**0.0277**	**0.0713**
HDR, ALB, NRM	den	0.1247	0.0393	0.0947
	pro-den	**0.0523**	0.0298	0.0789
HDR, ALB, NRM, VAR	den	0.0971	0.0326	0.0838
	pro-den	0.0536	0.0278	0.0742
	mc-render	0.0841	0.0669	0.1049

Table 1: *Comparison of our approach (pro-den) with simple denoising (den) for different input feature combinations.*

inputs	model	error		
		RMSE	SMAPE	FLIP
HDR, ALB, NRM, VAR	den-kpcn	0.1342	0.0362	0.0990
	pro-den	**0.0640**	**0.0285**	**0.0788**
	mc-render	0.0841	0.0669	0.1049

Table 2: *Our approach (pro-den) applied to a denoiser based on a kernel predicting network (den-kpcn). The improvement is similar as to when applied to the U-Net based denoisers of Table 1.*

5.3. Training Hyperparameters

When training our networks (both our h_{NN} and the denoiser models), we use a batch size of 12, symmetric mean absolute percentage error (SMAPE) as a loss function, and the 'One Cycle' learning rate schedule with a maximum learning rate of $2 \cdot 10^{-5}$.

6. Results

We performed a series of experiments to characterize the performance of our method in comparison with other denoising approaches and to investigate the impact of our choices. We also compare with similar work and highlight the different performance characteristics of *progressive denoising* in the limit of many samples. We make our comparisons with respect to the following three error metrics: RMSE, SMAPE, and FLIP [ANSA21]. Error measures reported in Tables 1–4 are averages across the scenes/viewports shown in Figure 8, and in Tables 1–3 across sample counts ranging from 32 to 8192.

Comparison with Denoising. We trained multiple U-Net based denoisers (Section 5.1), each with a different combination of the input features, for 2500 epochs over our training dataset. For each of those denoisers we trained our h_{NN} network (Section 4.3). Results are listed in Table 1 and demonstrate that our progressive denoising approach improves upon each error metric in all cases. Visual comparisons illustrating preserved details are shown in Figure 9.

In addition to the U-Net based denoisers, we also tested how our method performs when applied to a denoiser based on a different network architecture, in this case a kernel-predicting convolutional network (KPCN) [BVM*17]. We configured this denoiser to predict 5×5 kernels, and its training was otherwise equal to that of the other networks. The results, listed in Table 2, show improvement

inputs	model	error		
		RMSE	SMAPE	FLIP
HDR	oidn	0.0499	0.0278	0.0735
	pro-den	**0.0394**	0.0251	**0.0685**
HDR, ALB, NRM	oidn	0.0586	0.0265	0.0743
	pro-den	0.0489	**0.0248**	0.0705
	mc-render	0.0841	0.0669	0.1049

Table 3: *Applying our method (pro-den) to a pre-trained denoiser, Intel Open Image Denoise (oidn). Despite the already high-quality of OIDN, our method is still able to lower the overall error.*

for each metric and indicate that our method is not limited to base denoisers of a single architecture.

Application to an Existing Denoiser. We trained our h_{NN} network for use with the pre-trained OIDN denoiser, which achieves higher quality than our own trained models, for 2500 epochs over 100% of our training dataset. Input images were first denoised and their SURE were computed. Here, our method was also able to achieve lower error in our test dataset and with respect to multiple metrics. Results are detailed in Table 3, and illustrated in Figures 1 and 10. Additional results, with per-scene and per-sample count errors, are available in the supplemental document.

Comparison with Deep Combiner [BHHM20]. We applied the pre-trained 'single buffer' network of deep combiner (DC) to our test dataset, denoised by one of the networks (HDR, ALB, NRM) from Table 1. Like our method, DC was able to improve upon the quality of the denoised images, but its performance characteristics were notably different to those of our method. As clearly exemplified by Figure 11, while DC continually provides a small improvement over the denoised image, in the limit of many samples its output has higher RMS error than the rendered image. In contrast, the output of our method converges to the ground truth along with the rendered image as the sample count increases. At low sample counts, DC performs better than our method and we suggest this is due to our method being constrained to per-pixel mixing, where as DC combines its inputs by filtering neighbouring pixel residuals using 15×15 kernels. However, there is nothing preventing us from applying our method to the output from the DC approach in order to improve the performance of our method at low sample counts while retaining consistency.

6.1. Network Ablation

To investigate the contribution of some individual element constituting our method, we performed a series of ablation experiments for which we train a network with some element removed and compare it to the unmodified method. The tested elements were the inclusion of SURE as a network input, the indirect approach to computing the per-pixel mixing parameter α by using f_{act}, and the t-statistic based scaling of α. The results, shown in Table 4, indicate that the removal of any one of these elements lowers the quality of the output image, in terms of RMS error, by some degree.

All our results were computed on desktop PC equipped with an AMD Ryzen Threadripper 3970X 32-core CPU, an NVIDIA GeForce RTX 3090 GPU, and 64 GB of RAM. Excluding file IO, a

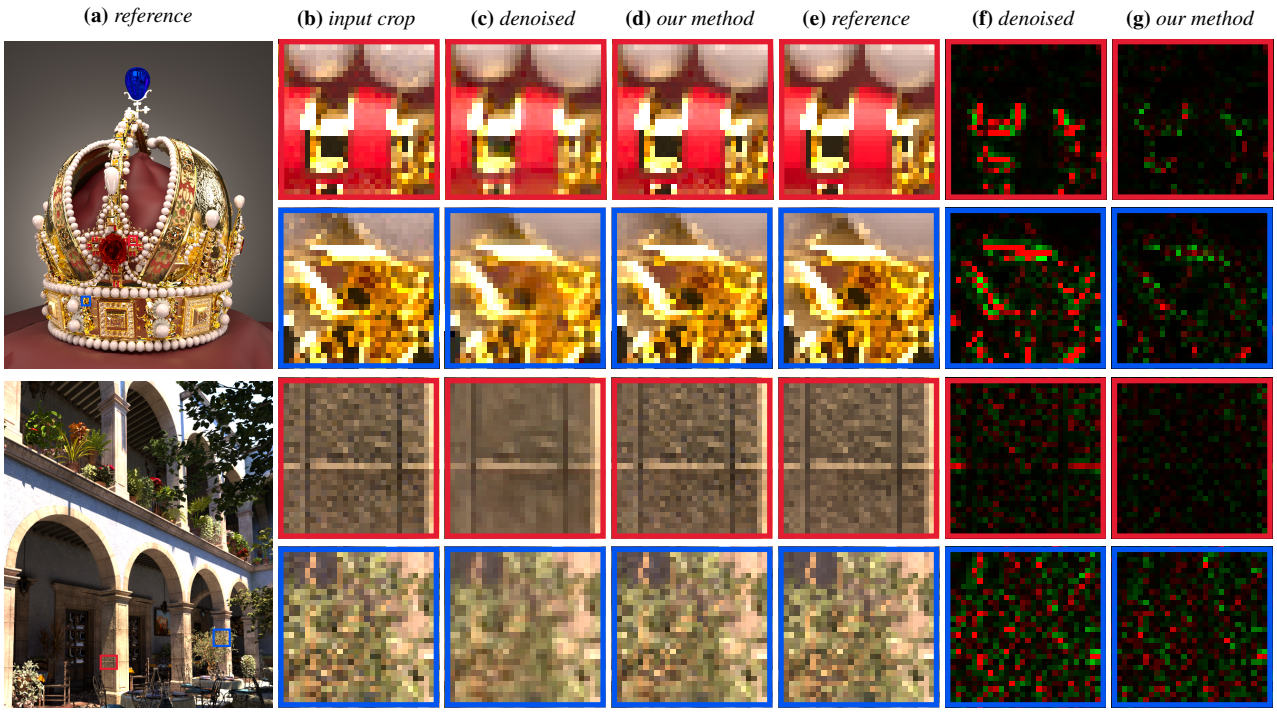

Figure 9: *Visual comparison of our trained models (HDR, ALB, NRM, VAR of Table 1): denoising (c,f) versus our method (d,g). In the difference images (f,g), negative difference is red, positive is green, and values are uniformly scaled for visual clarity. RMSE of the denoised and progressively denoised images are 0.139 and 0.043, respectively, for the top scene (1024spp input), and 0.028 and 0.018 for the bottom scene (256spp input).*

	Samples Per Pixel				
	32	128	512	2048	8192
w/o SURE	0.1216	0.0796	0.0499	0.0288	0.0151
w/o f_{act}	0.1238	0.0821	0.0487	0.0247	0.0123
w/o t-test	0.1224	0.0801	0.0487	0.0247	0.0124
pro-den	**0.1185**	**0.0754**	**0.0462**	**0.0239**	**0.0123**
mc-render	0.2331	0.1162	0.0581	0.0287	0.0136

Table 4: *Network ablation experiments. Average RMSE of the 10 scenes in our testset. The removal of any of these elements from our method (pro-den) results in lessened quality improvement, although the magnitude of this difference is heavily dependent on the input. In the w/o f_{act} case, the network directly predicts α rather than values for x_1, x_2, and x_3.*

1920x1080 image took 7 ms to denoise, 34 ms to compute SURE, and 42 ms to run the *progressive denoising* steps of Section 4.2, for a total average time of 83 ms. The absolute training time for each model was approximately 10 hours.

7. Discussion, Limitations, and Future Work

Regarding the choice of denoising input features, we found the benefits of including certain features, particularly albedo and normals, to be scene dependent. Anecdotally, the inclusion of variance as a denoiser input seems to be responsible for the biggest increase in quality. Counterintuitively, including additional features sometimes results in worse denoising quality in certain image areas.

During training, we swap the rendered and denoised images (as mentioned in Section 4.3). We do this to avoid that the network converges to a state of simply returning the denoised image, which always happened when the swapping was not performed. When swapping is used in training, the resulting network has the advantage that we can swap the arguments and still get the right blending factor ($\beta = 1 - \alpha$). In the ablation experiment results of Table 4, the exclusion of the SURE input feature leads to degraded performance, particularly at high sample counts, which is when the SURE estimate is most accurate as illustrated in Figure 2. We believe this demonstrates that the proposed random swapping guides the network to learn a swap invariant association between an image and its error estimate, as without it the network is no longer able to compare unbiased error estimates and likely relies on some sub-optimal learned criteria to explain the observed discrepancy.

The neighbourhood size of the filter used to compute the weighted average of the non-denoised radiance values, as part of the t-statistic computation, is an important parameter, as is the shift to the neighbouring pixel with the highest variance. At lower sample counts, it is difficult to accurately estimate variance for a single pixel, particularly when sampled paths do not connect with a light source and return a radiance value of 0, resulting in a variance estimate of 0. To overcome this, we instead estimate the variance of a weighted average of its neighborhood. We found the 11-by-11

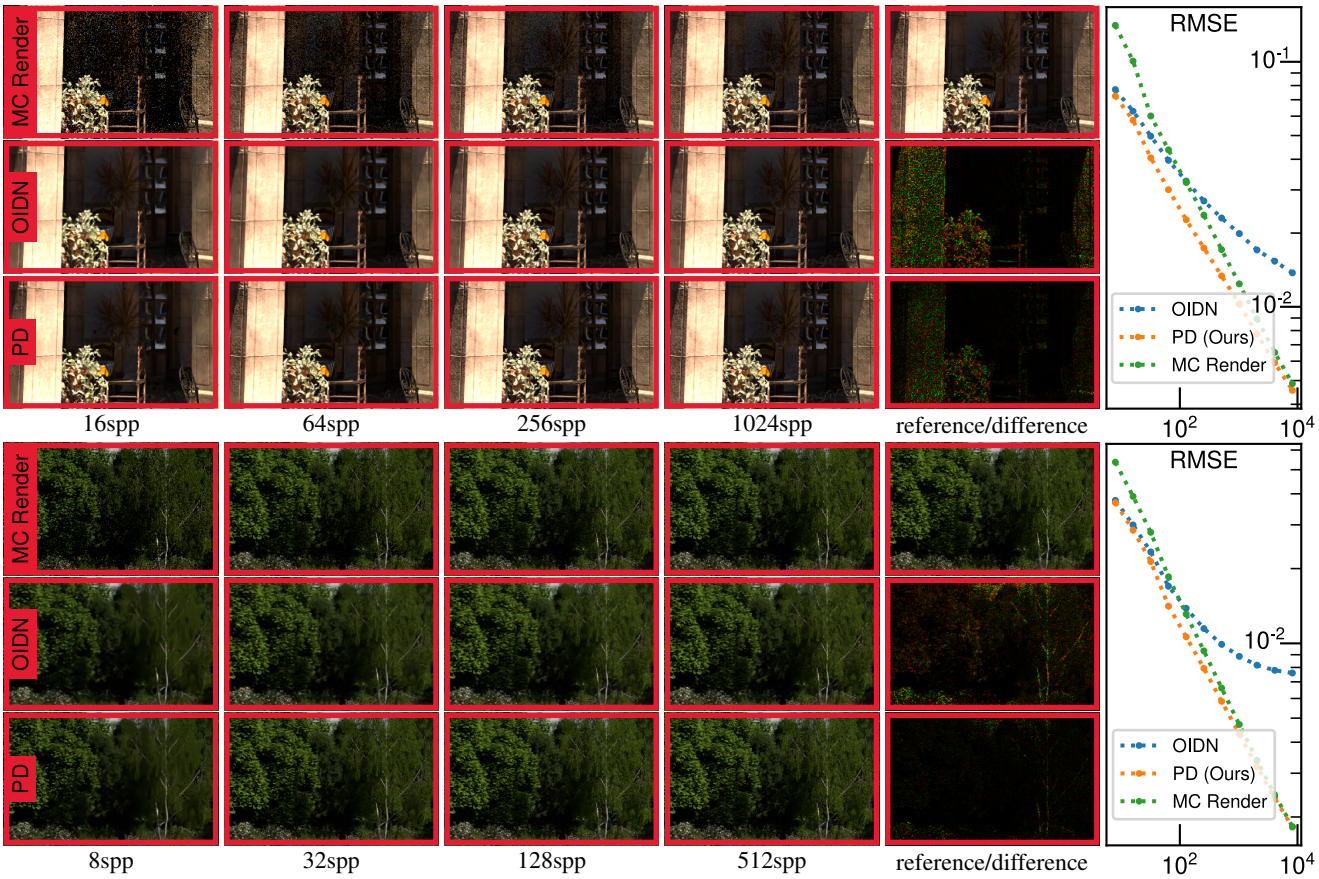

Figure 10: *Two progressive series of MC renderings illustrating the performance of our method (PD) applied onto the OIDN output, and their convergence plots. Reference images were rendered using 32K samples per pixel (spp) and difference images taken with the result in the fourth column are scaled by 16 and have negative and positive differences colored red and green. By incorporating error information about the rendered and denoised images, our method substantially improves the denoised result, especially as the number of samples increases.*

size to be a good compromise between gathering sufficient statistics and the performance impact of a larger kernel, and this is illustrated in Figure 12. This still resulted in some artifacts, seen as noisy splotches, and we found placing more weight on the neighboring pixel of highest variance resolved this issue while keeping the method convergent. At very low sample counts however, these artifacts may still be present and are a limitation of our method, as seen in Figure 13.

For scenes with extremely high sample variance, we have observed that the output of our method may converge to the ground truth slower than the MC rendered image, as shown in Figure 14. This is partly a consequence of our previously mentioned variance overestimation and method of α scaling. Despite this, our method is still guaranteed to be asymptotically unbiased in the limit of many samples.

We briefly investigated how our method performs with regard to temporal coherency by denoising sequences of still images rendered with different random seeds and at sample counts of 32, 256, and 2048 samples per pixel. When compared to the same sequences denoised without *progressive denoising*, we found no visual improvement in terms of temporal coherency. It thus remains an area of future work, to investigate how approaches used to construct temporally coherent denoisers could be combined with the method presented here.

Another area of future work would be to research a manner of computing a neural network's divergence that does not involve MC estimation. This may lead to better performance for our and other SURE based methods, although if it is possible, it would likely require specific support from machine learning frameworks.

8. Conclusion

We have presented a method useful for *progressive denoising*. Our approach is to find per-pixel mixing parameters using a neural network. The mix is between MC rendered and denoised images, and it is asymptotically unbiased in the limit of many samples. We include computed error estimates as inputs for our network, using SURE to estimate the squared error of denoised images. To guarantee convergence, we used estimation of confidence intervals based on the Student's t distribution. We provided a theoretical motivation for

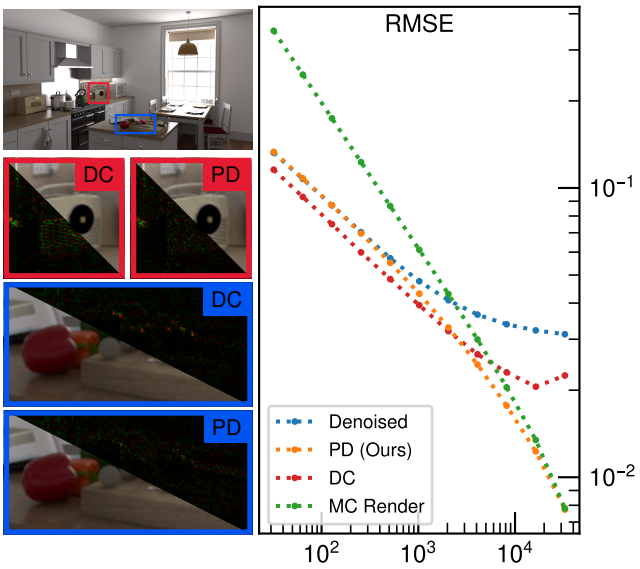

Figure 11: *Comparison of our method, Progressive Denoising (PD), with Deep Combiner (DC), kitchen test scene. While DC continually improves upon the denoised image, our method performs the best as the quality of the MC rendered input increases.*

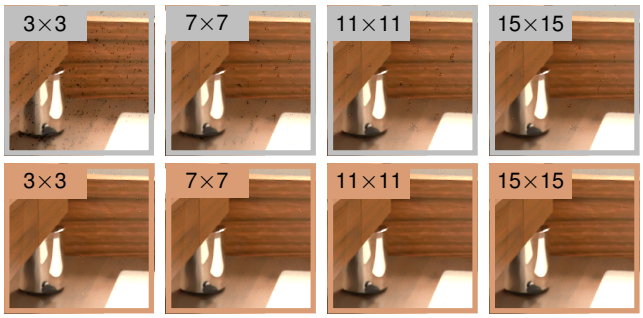

Figure 12: *Visual comparison of using different neighbourhood sizes for the weighted average calculation of Eq. 14, at 32 samples per pixel. While increasing the neighbourhood size reduces the visible artifacts (top row), it is not sufficient. Weighting the average towards the pixel with the highest variance (bottom row), removes most of these artifacts at low sample counts.*

our approach, and we demonstrated the improvement our method provides over our trained denoisers, as well as when applied to a popular existing denoiser.

The source code of our implementation is made available on GitHub (https://github.com/ArthurFirmino/progressive-denoising).

Acknowledgements

We would like to thank Shilin Zhu for discussion regarding the implementation of our procedural scene generator. We also extend our gratitude to Matt Pharr for the PBRT renderer [PJH16], Wenzel Jakob for the Mitsuba renderer [NDVZJ19], and their

Figure 13: *Limitation of our method at very low sample counts, 2spp in this example, arising from insufficiently accurate sample variance estimates.*

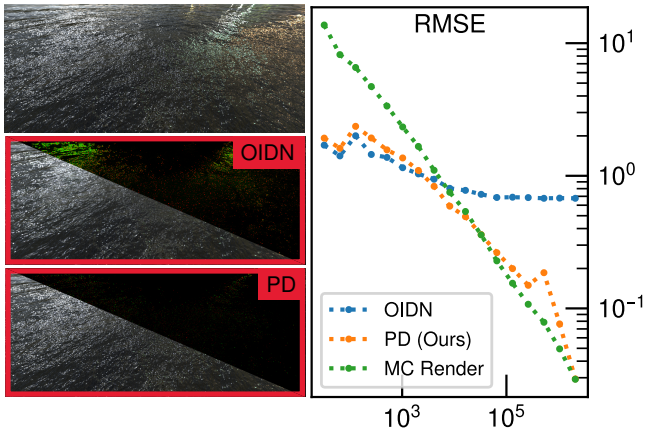

Figure 14: *Convergence plot of our method for a scene with extremely high variance. At between 10^5 and 10^6 samples, Progressive Denoising converges slower than the MC rendered image, but still provides improvement over the denoised image.*

respective teams. Intel for open sourcing their denoiser training framework [oid], the scene repositories by Matt Pharr [Pha20]and Benedikt Bitterli [Bit16], and the individual authors of each scene (names available in the scene repositories). This research is a part of PRIME which is funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska Curie grant agreement No 956585.

References

[ANSA21] ANDERSSON P., NILSSON J., SHIRLEY P., AKENINE-MÖLLER T.: Visualizing the error in rendered high dynamic range images. In *Eurographics Short Papers* (May 2021). doi:10.2312/egs.20211015. 7

[BAC*18] BURLEY B., ADLER D., CHIANG M. J.-Y., DRISKILL H., HABEL R., KELLY P., KUTZ P., LI Y. K., TEECE D.: The design and evolution of Disney's Hyperion renderer. *ACM Transactions on Graphics 37*, 3 (2018), 33:1–33:22. doi:10.1145/3182159. 2

[BHHM20] BACK J., HUA B.-S., HACHISUKA T., MOON B.: Deep combiner for independent and correlated pixel estimates. *ACM Transactions on Graphics 39*, 6 (2020), 242:1–242:12. doi:10.1145/3414685.3417847. 2, 3, 7

[Bit16] BITTERLI B.: Rendering resources, 2016. https://benedikt-bitterli.me/resources/. 10

[BRM*16] BITTERLI B., ROUSSELLE F., MOON B., IGLESIAS-GUITIÁN J. A., ADLER D., MITCHELL K., JAROSZ W., NOVÁK J.: Nonlinearly weighted first-order regression for denoising Monte Carlo renderings. *Computer Graphics Forum 35*, 4 (2016), 107–117. doi:10.1111/cgf.12954. 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 Transactions on Graphics 36*, 4 (2017), 97:1–97:14. doi:10.1145/3072959.3073708. 2, 7

[CFS*18] CHRISTENSEN P., FONG J., SHADE J., WOOTEN W., SCHUBERT B., KENSLER A., FRIEDMAN S., KILPATRICK C., RAMSHAW C., BANNISTER M., ET AL.: RenderMan: An advanced path-tracing architecture for movie rendering. *ACM Transactions on Graphics 37*, 3 (2018), 30:1–30:21. doi:10.1145/3182162. 2, 3

[CJ16] CHRISTENSEN P. H., JAROSZ W.: The path to path-traced movies. *Foundations and Trends® in Computer Graphics and Vision 10*, 2 (2016), 103–175. doi:10.1561/0600000073. 2

[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 Transactions on Graphics 36*, 4 (2017), 98:1–98:12. doi:10.1145/3072959.3073601. 2, 3, 6

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1 (January 1986), 51–72. doi:10.1145/7529.8927. 2

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. *Computer Graphics (SIGGRAPH '84) 18*, 3 (July 1984), 137–145. doi:10.1145/800031.808590. 2

[HY21] HUO Y., YOON S.-E.: A survey on deep learning-based Monte Carlo denoising. *Computational Visual Media 7*, 2 (2021), 169–185. doi:10.1007/s41095-021-0209-9. 2

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)* (2016), pp. 770–778. doi:10.1109/CVPR.2016.90. 5

[JC95] JENSEN H. W., CHRISTENSEN N. J.: Optimizing path tracing using noise reduction filters. In *Proceedings of WSCG 1995* (1995), pp. 134–142. URL: http://hdl.handle.net/11025/16029. 2

[Kaj86] KAJIYA J. T.: The rendering equation. *Computer Graphics (SIGGRAPH '86) 20*, 4 (August 1986), 143–150. doi:10.1145/15922.15902. 1

[KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering Monte Carlo noise. *ACM Transactions on Graphics 34*, 4 (2015), 122:1–122:12. doi:10.1145/2766977. 2

[KCK*18] KŘIVÁNEK J., CHEVALLIER C., KOYLAZOV V., KARLÍIK O., JENSEN H. W., LUDWIG T.: Realistic rendering in architecture and product visualization. In *ACM SIGGRAPH 2018 Courses* (2018), p. 10. doi:10.1145/3214834.3214872. 2

[KCSG18] KULLA C., CONTY A., STEIN C., GRITZ L.: Sony Pictures Imageworks Arnold. *ACM Transactions on Graphics 37*, 3 (2018), 29:1–29:18. doi:10.1145/3180495. 2

[KS13] KALANTARI N. K., SEN P.: Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum 32*, 2pt1 (May 2013), 93–102. doi:10.1111/cgf.12029. 2

[LR90] LEE M. E., REDNER R. A.: A note on the use of nonlinear filtering in computer graphics. *IEEE Computer Graphics and Applications 10*, 3 (May 1990), 23–29. doi:10.1109/38.55149. 2

[LWC12] LI T.-M., WU Y.-T., CHUANG Y.-Y.: SURE-based optimization for adaptive sampling and reconstruction. *ACM Transactions on Graphics 31*, 6 (2012), 194:1–194:9. doi:10.1145/2366145.2366213. 2, 3, 4

[NDVZJ19] NIMIER-DAVID M., VICINI D., ZELTNER T., JAKOB W.: Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics 38*, 6 (2019), 203:1–203:17. doi:10.1145/3355089.3356498. 6, 10

[oid] Intel® Open Image Denoise. https://www.openimagedenoise.org/index.html. Accessed: 2021-09-08. 3, 6, 10

[Pha20] PHARR M.: pbrt-v4-scenes. https://github.com/mmp/pbrt-v4-scenes, 2020. Accessed: 2022-01-13. 10

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016. doi:10.1016/C2013-0-15557-2. 3, 10

[Pur87] PURGATHOFER W.: A statistical method for adaptive stochastic sampling. *Computers & Graphics 11*, 2 (1987), 157–162. doi:10.1016/0097-8493(87)90029-X. 2, 5

[rad] Radeon™ Image Filtering Library. https://gpuopen.com/radeon-image-filtering-library/. Accessed: 2021-10-03. 3

[RBU08] RAMANI S., BLU T., UNSER M.: Monte-Carlo SURE: A black-box optimization of regularization parameters for general denoising algorithms. *IEEE Transactions on Image Processing 17*, 9 (2008), 1540–1554. doi:10.1109/TIP.2008.2001404. 3

[RMZ13] ROUSSELLE F., MANZI M., ZWICKER M.: Robust denoising using feature and color information. *Computer Graphics Forum 32*, 7 (October 2013), 121–130. doi:10.1111/cgf.12219. 2, 3

[RW94] RUSHMEIER H. E., WARD G. J.: Energy preserving non-linear filters. In *Proceedings of SIGGRAPH 1994* (1994), pp. 131–138. doi:10.1145/192161.192189. 2

[SD12] SEN P., DARABI S.: On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Transactions on Graphics 31*, 3 (2012), 18:1–18:15. doi:10.1145/2167076.2167083. 2

[Ste81] STEIN C. M.: Estimation of the mean of a multivariate normal distribution. *The Annals of Statistics* (1981), 1135–1151. doi:10.1214/aos/1176345632. 3, 4

[TJ97] TAMSTORF R., JENSEN H. W.: Adaptive sampling and bias estimation in path tracing. In *Eurographics Workshop on Rendering Techniques (EGWR '97)* (1997), Springer, pp. 285–295. doi:10.1007/978-3-7091-6858-5_26. 2, 5

[VRM*18] VOGELS T., ROUSSELLE F., MCWILLIAMS B., RÖTHLIN G., HARVILL A., ADLER D., MEYER M., NOVÁK J.: Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics 37*, 4 (2018), 124:1–124:15. doi:10.1145/3197517.3201388. 2, 5, 6

[WW19] WONG K.-M., WONG T.-T.: Deep residual learning for denoising Monte Carlo renderings. *Computational Visual Media 5*, 3 (2019), 239–255. doi:10.1007/s41095-019-0142-3. 2

[XC20] XING Q., CHEN C.: Path tracing denoising based on SURE adaptive sampling and neural network. *IEEE Access 8* (2020), 116336–116349. doi:10.1109/ACCESS.2020.2999891. 3

[XZW*19] XU B., ZHANG J., WANG R., XU K., YANG Y.-L., LI C., TANG R.: Adversarial Monte Carlo denoising with conditioned auxiliary feature modulation. *ACM Transactions on Graphics 38*, 6 (2019), 224:1–224:12. doi:10.1145/3355089.3356547. 2

[ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHI R., ROUSSELLE F., SEN P., SOLER C., YOON S.-E.: Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Computer Graphics Forum 34*, 2 (2015), 667–681. doi:10.1111/cgf.12592. 2

[ZZXY21] ZHENG S., ZHENG F., XU K., YAN L.-Q.: Ensemble denoising for Monte Carlo renderings. *ACM Transactions on Graphics 40*, 6 (2021), 274:1–274:17. doi:10.1145/3478513.3480510. 2, 3