Deep Combiner for Independent and Correlated Pixel Estimates

JONGHEE BACK, Gwangju Institute of Science and Technology, South Korea
BINH-SON HUA, VinAI Research, Vietnam and VinUniversity, Vietnam
TOSHIYA HACHISUKA, The University of Tokyo, Japan
BOCHANG MOON, Gwangju Institute of Science and Technology, South Korea

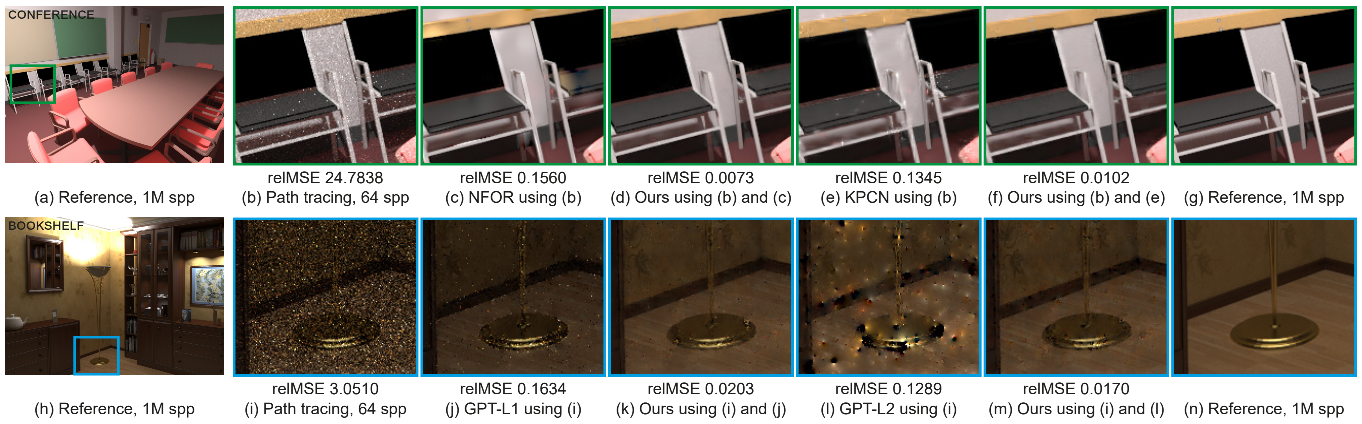


Fig. 1. Our framework allows us to combine two different types of images, independent pixel estimates (e.g., path traced images) and correlated pixel estimates (e.g., denoised images), and reduces remaining errors (residual noise or systematic errors) of the existing methods such as Nonlinearly weighted First-Order Regression (NFOR) [Bitterli et al. 2016], Kernel-Predicting Convolutional Networks (KPCN) [Bako et al. 2017], and Gradient-domain Path Tracing with L1 and L2 reconstruction (GPT-L1 and GPT-L2) [Kettunen et al. 2015]. The numbers are the relative mean squared error (relMSE) [Rousselle et al. 2011].

Monte Carlo integration is an efficient method to solve a high-dimensional integral in light transport simulation, but it typically produces noisy images due to its stochastic nature. Many existing methods, such as image denoising and gradient-domain reconstruction, aim to mitigate this noise by introducing some form of correlation among pixels. While those existing methods reduce noise, they are known to still suffer from method-specific residual noise or systematic errors. We propose a unified framework that reduces such remaining errors. Our framework takes a pair of images, one with independent estimates, and the other with the corresponding correlated estimates. Correlated pixel estimates are generated by various existing methods such as denoising and gradient-domain rendering. Our framework then combines the two images via a novel combination kernel. We model our combination kernel as a weighting function with a deep neural network that exploits the correlation among pixel estimates. To improve the robustness of our framework for outliers, we additionally propose an extension to handle multiple image buffers. The results demonstrate that our unified framework can successfully reduce the error of existing methods while treating them as black-boxes.

CCS Concepts: • **Computing methodologies** → **Ray tracing**.

Authors' addresses: Jonghee Back, Gwangju Institute of Science and Technology, South Korea; Binh-Son Hua, VinAI Research, Vietnam and VinUniversity, Vietnam; Toshiya Hachisuka, The University of Tokyo, Japan; Bochang Moon, Gwangju Institute of Science and Technology, South Korea.

Additional Key Words and Phrases: Combination Kernel, Monte Carlo Ray Tracing

ACM Reference Format:
Jonghee Back, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. 2020. Deep Combiner for Independent and Correlated Pixel Estimates. *ACM Trans. Graph.* 39, 6, Article 242 (December 2020), 12 pages. https://doi.org/10.1145/3414685.3417847

1 INTRODUCTION

Monte Carlo (MC) rendering [Kajiya 1986] has been recognized as a powerful tool for light transport simulation, which has been widely adopted in production rendering recently [Pharr 2018]. MC rendering can simulate a variety of lighting effects by randomly sampling light paths and averaging their contributions in every pixel. Pixel estimates in MC rendering are typically independent of each other. The main problem is that it outputs noisy pixel estimates, which also stem from its random nature. In general, a large number of samples (thus a considerable amount of computation time) are needed to reduce such noise to an imperceptible level.

A popular class of noise reduction methods in MC rendering is image-space denoising [Overbeck et al. 2009; Sen and Darabi 2012]. Its main advantage is that it can handle different types of random noise generated by complex lighting effects without suffering from the complexity of light transport. Learning-based denoising [Bako et al. 2017; Chaitanya et al. 2017; Gharbi et al. 2019; Xu et al. 2019] has achieved an impressive level of noise reduction recently. The denoising process typically introduces correlation among pixels since each denoised pixel is a weighted sum of independent pixel estimates. Common to all the existing denoising methods, however, is that they tend to leave residual noise or blur high-frequency details, especially when a small number of samples are used.

On a different line of research, gradient-domain rendering [Kettunen et al. 2015; Lehtinen et al. 2013] can also suppress MC noise by estimating image gradients using correlated path sampling. Gradient-domain rendering performs a screened Poisson reconstruction [Bhat et al. 2008] to obtain a final image from those estimated gradients. Similar to denoising, pixel estimates in the gradient-domain rendering are correlated. Unlike denoising, however, Poisson reconstruction allows us to achieve unbiased estimates while significantly reducing noise. The errors in gradient-domain rendering instead show up as dipole artifacts, which are the results of the diffusion of errors in inaccurate gradients. The reconstruction with the L1 norm can reduce the dipole artifacts, but it will reintroduce bias in the form of energy loss.

We propose a unified framework that can reduce the artifacts of correlated pixel estimates in the existing methods. Correlated pixel estimates in this paper cover various types of pixel estimates, such as those in denoising, gradient-domain rendering, and even correlated sampling methods. We observe that such existing methods generally introduce positive correlation across pixels. This observation leads us to design a combination kernel, which takes the independent and correlated pixel estimates as inputs and produces an improved output by weight-averaging the inputs while taking account of the correlation. The combined estimator allows us to reduce the remaining errors in the existing correlated pixel estimators. To summarize, our contributions are:

- Unified framework that improves the output of the existing methods, where different forms of errors exist;
- Weighted combination kernel that exploits spatially-varying correlations across pixels;
- Multi-buffered kernel that improves the robustness against outliers in independent and correlated pixel estimates;
- Deep neural network which estimates the weights of our kernel robustly for different types of correlated pixel estimates.

We demonstrate that our approach is able to boost various denoising/correlated sampling methods by reducing their specific types of errors. These results are achieved through a unified framework, without tailoring our method to a specific technique.

2 RELATED WORK

We classify the related work into three categories. We design our method to be orthogonal to any of those; Our method can take the output of any of those existing methods as an input to further improve the accuracy. A reader may refer to Zwicker et al. [2015] for a comprehensive overview of other denoising methods and Hua et al. [2019] for a survey of gradient-domain rendering.

Denoising for Independent Pixel Estimates. Image denoising for independent pixel estimates has a long history [McCool 1999]. A common approach is to take well-established image filters designed for reducing noise in photography. Such examples include a cross-bilateral filter [Li et al. 2012; Sen and Darabi 2012], non-local means [Rousselle et al. 2012, 2013], wavelet thresholding [Overbeck et al. 2009] and non-local Bayes [Boughida and Boubekeur 2017]. They all modified the existing image filters to incorporate rendering-specific information (e.g., per-pixel variance). As an alternative to the filtering formulation, Moon et al. [2014] investigated a local regression theory where pixel colors are locally-fitted on a feature space spanned by G-buffers (normals, textures, and depths). This regression approach has recently been improved by using higher-order polynomials [Moon et al. 2016] or non-local means weighting [Bitterli et al. 2016].

Recently, many methods take advantage of machine learning approaches to derive data-driven filters. Kalantari et al. [2015] pioneered such an approach by using a neural network to output parameters of the existing image filters. Bako et al. [2017] proposed a denoising framework that exploits a convolutional neural network where its last layer produces denoising weights. Chaitanya et al. [2017] suggested using a recurrent neural network to incorporate temporal coherence in interactive rendering. Gharbi et al. [2019] found that a simpler network can be used for learning a splatting filter kernel, and Xu et al. [2019] adopted the concept of an adversarial network to replace a numerical error metric for training.

While some modern methods have demonstrated significant noise reduction, common to all is that the denoised images tend to have over-blurred high-frequency details or residual noise such as low-frequency noise. We aim at reducing such remaining errors with a post-processing method. The inputs to our framework are the noisy input (e.g., the path traced image) and the corresponding denoised output, and additional rendering-specific features (e.g., normals, textures, and depths). Note that pixel estimates after denoising are correlated since denoising methods introduce some correlation among pixels. For this application, our framework does not require us to take any additional sample, since correlated estimates are produced from independent estimates by denoising. We model a denoising method as a black-box function that outputs correlated but filtered pixel estimates from independent pixel estimates.

Correlated Sampling. Since denoising methods output correlated pixel estimates in the end, another logical approach is to modify the rendering process itself to *directly output* correlated pixel estimates. Unlike denoising methods, this approach can avoid introducing bias by carefully controlling the sampling process in rendering. Keller and Heidrich [2001] presented such an approach by tiling the same sampling patterns across the image. Bekaert et al. [2002] introduced a path-reuse technique that shares the sampled light transport path across multiple pixels. Rather than paths themselves, Sadeghi et al. [2009] proposed to reuse the same sequence of random numbers for each pixel. Bauszat et al. [2017] generalized the path-reuse technique to improve the robustness in rendering glossy surfaces. They all tend to suppress random noise when compared to independent pixel estimates. While correlated sampling can achieve unbiased estimates with less noise, the output images often suffer from a different kind of errors, *structured* noise. The existing denoising methods cannot remove such structured noise effectively since they all assume independent pixel estimates as their input. Our framework addresses this problem of structured noise for the first time by explicitly modeling correlated pixel estimates as our input.

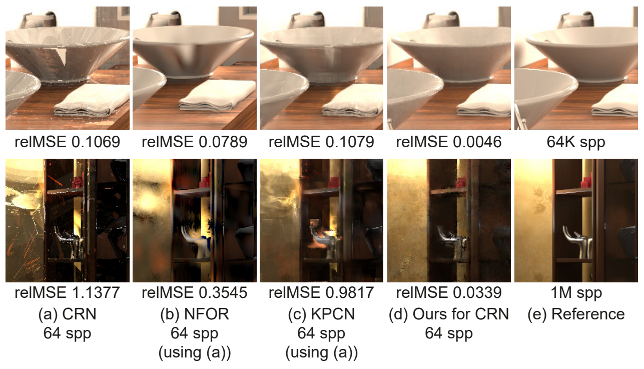


Fig. 2. Equal-sample comparisons with NFOR and KPCN. The denoising results ((b) and (c)) are generated by NFOR and KPCN, respectively, where we used the CRN image (a) as their input. Our results (d) with 64 spp are generated using two inputs; a path traced image with 32 spp and a CRN result with 32 spp. The quality improvement from NFOR and KPCN over their input (a) is not significant, as these methods assume independent pixel estimates. With the same number of samples, our method produces much-improved results (d) by explicitly taking the correlation into account.

Gradient-Domain Rendering as a Hybrid. A promising alternative is gradient-domain rendering [Lehtinen et al. 2013]. Gradient-domain rendering can be thought of as a hybrid of image denoising and correlated sampling. In the first step of gradient-domain rendering, image-space gradients are estimated by a form of correlated sampling called *shift mapping*. Shift mapping is carefully designed to reduce noise in the gradient estimates while remaining unbiased. In the second step, a screened Poisson reconstruction is conducted to output the final image using the gradient estimates and independent pixel estimates. Kettunen et al. [2015] later showed that this process could be seen as low-pass filtering via a minimization problem. Unlike the denoising methods, this process is tightly coupled with estimated gradients to achieve *unbiased* denoising when formulated as an L2 minimization.

While being unbiased, the L2 minimization tends to produce structured noise (*dipole artifacts*). A biased reconstruction with L1 minimization is thus suggested as a practical alternative [Kettunen et al. 2015; Lehtinen et al. 2013]. While the L1 minimization is successful at suppressing dipole artifacts, it causes a loss of brightness (bias) in the resulting images. Despite some recent advances [Guo et al. 2019; Kettunen et al. 2019], the images from gradient-domain rendering can still suffer from either bias or structured noise, especially when gradient estimates are erroneous.

Since our method also takes both correlated and independent estimates to produce the final image, it can also be interpreted as a reconstruction algorithm in gradient-domain rendering. Unlike gradient-domain rendering, however, we make minimal assumptions regarding the input, and thus the correlated pixel estimates do not need to be gradient estimates. As a result, our method can use *both* input and output of gradient-domain reconstruction as *our input*, in order to reduce the remaining errors in the reconstructed image (e.g., dipole artifacts for L2 and energy loss for L1).

3 STATISTICAL MODEL AND MOTIVATION

We formally describe our statistical models and show a motivating example to explain our approach. We consider independent pixel estimates and correlated pixel estimates differently.

3.1 Independent Pixel Estimates

Let us consider a model where pixels are independently estimated in an unbiased manner. An image that follows this model can be generated by an unbiased MC rendering technique such as path tracing. Under this model, each pixel estimate y_c is expressed as

$$y_c = \mu_c + \epsilon_c, \tag{1}$$

where μ_c is the ground truth value with a center pixel c, and ϵ_c is the error. Since y_c is an unbiased estimate of μ_c, the expected error is zero (i.e., $E[\epsilon_c] = 0$). Having independent pixel estimates means that there is no correlation among errors in neighboring pixels. Considering a set of neighboring pixels Ω_c around the pixel c, it means that $\text{cov}(y_c, y_i) = 0$ for any $i \in \Omega_c$. Note that the neighboring set Ω_c does not include the pixel c. This model has also been widely used as a model of the input to the existing denoising methods.

3.2 Correlated Pixel Estimates

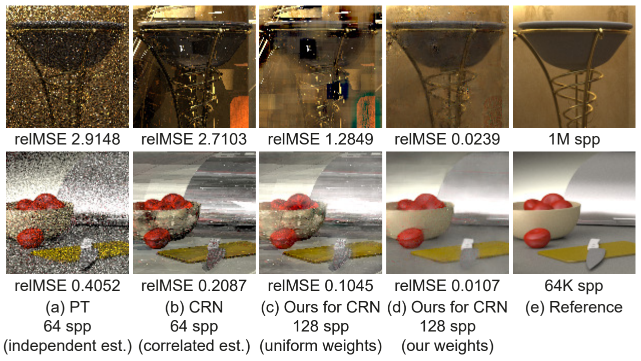
Let us consider another model where pixel estimates are locally *correlated*. As we mentioned in Section 2, the correlation between pixel estimates can be introduced in various ways. Unlike the model for independent pixel estimates where we consider each pixel estimate independently (Eq. 1), we model the *difference* between two correlated pixel estimates z_c and z_i as

$$z_c - z_i = \mu_c - \mu_i + \epsilon_{ci}, \tag{2}$$

where ϵ_{ci} is the error from the difference $\mu_c - \mu_i$. Similar to the model for independent pixel estimates, this model assumes that the expected value of ϵ_{ci} is zero ($E[\epsilon_{ci}] = 0$) and thus $z_c - z_i$ is unbiased. This model, however, assumes that $\text{cov}(z_c, z_i) \neq 0$ for any $i \in \Omega_c$. In other words, the pixel estimates z_c and z_i are correlated under this model.

This model holds exactly for unbiased methods (e.g., gradient-domain reconstruction with L2 minimization and correlated sampling) since $E[z_c - z_i] = \mu_c - \mu_i$. For biased methods, we generally have $E[z_c - z_i] \neq \mu_c - \mu_i$ since each pixel value is biased. The bias of this difference $E[z_c - z_i] - (\mu_c - \mu_i)$, however, is typically smaller than the bias of individual pixel value (e.g., energy loss in Fig. 1 (j)). We thus assume that this model approximately holds for the biased methods.

Fig. 2 shows an example of the correlated pixel estimates generated by sharing a common random number (CRN) sequence across pixels. One could attempt to reduce the structured noise in the CRN image using state-of-the-art denoisers such as a first-order regression (NFOR [Bitterli et al. 2016]) and a learning-based denoiser (KPCN [Bako et al. 2017]). For KPCN, we train its network using CRN images. As can be seen, the existing denoisers (NFOR (b) and KPCN (c)) fail to reduce structured error present in correlated images (i.e., (a)) since none of them considers the correlation among pixel estimates. This observation motivates us to design a new technique that can reduce such artifacts by considering the correlation.



relMSE 2.9148
relMSE 2.7103
relMSE 1.2849
relMSE 0.0239
1M spp
relMSE 0.4052
relMSE 0.2087
relMSE 0.1045
relMSE 0.0107
64K spp
(a) PT 64 spp (independent est.)
(b) CRN 64 spp (correlated est.)
(c) Ours for CRN 128 spp (uniform weights)
(d) Ours for CRN 128 spp (our weights)
(e) Reference

Fig. 3. Comparisons between combination results with uniform (c) and our weighting (d). Both methods use the same inputs (a) and (b). The results with uniform weighting (c) still have some structured artifacts since neighboring pixels with a low correlation are simultaneously utilized for our combination process. On the other hand, our weighted kernel (d) considers heterogeneous correlation by assigning different weights to neighboring pixels according to its correlation. As a result, the results (d) from our weighted kernel show much improved quality.

3.3 Combined Estimator

For a given pixel c, let us consider taking a sum of the independent pixel estimate y_i and the difference of correlated pixel estimates $z_c - z_i$ as

$$y_i + (z_c - z_i) = (\mu_i + \epsilon_i) + (\mu_c - \mu_i + \epsilon_{ci}) = \mu_c + \epsilon_i + \epsilon_{ci}. \tag{3}$$

Since $E[\mu_c + \epsilon_i + \epsilon_{ci}] = \mu_c$ under our assumptions ($E[\epsilon_i] = 0$ and $E[\epsilon_{ci}] = 0$), this sum serves as yet another estimate of pixel c in addition to its independent estimate y_c. Since one can construct a similar estimate $y_i + (z_c - z_i)$ for any $i \in \Omega_c$, let us consider combining all the possible estimates via a weighted sum:

$$\frac{1}{W_c}\left(w_c y_c + \sum_{i\in\Omega_c} w_i(y_i + z_c - z_i)\right), \tag{4}$$

where $w_c, w_i > 0$ are positive weights allocated to pixel c and i and $W_c = w_c + \sum_{i\in\Omega_c} w_i$. We show that this estimator can be derived from a least-squares minimization problem in the next section. The main challenge is to estimate w_c and w_i that minimize the error of the combined estimator. It is challenging since $\text{cov}(z_c, z_i)$ can significantly vary per pixel and also per method. We leverage a deep learning model that estimates w_c and w_i to address the challenge.

4 COMBINATION KERNEL

Our method takes both independent pixel estimates y and correlated pixel estimates z as inputs, and outputs a weighted sum of the two pixel estimates. We show how the weighted sum (Eq. 4) that we mentioned in the last section is actually minimizing the weighted sum of squared errors (Sec. 4.1). For determining proper weights, we utilize a deep neural network (Sec. 4.2). We also provide an error analysis of our estimator to further study the behavior of the estimator (Sec. 4.3). Finally, we extend our estimator to utilize multiple image buffers to improve the robustness against outliers (Sec. 4.4).

4.1 Derivation

Given the statistical models (Eq. 1 and 2) for our inputs, our goal is to estimate the ground truth color μ_c at each center pixel c. Let us use β_c and β_i to denote unknown parameters that we estimate, which are corresponding to the ground truth μ_c and μ_i at center pixel c and neighboring pixel i, respectively. We then define the residual (error) of each pixel in y and z as

$$\begin{aligned} r(y_c) &\equiv y_c - \beta_c, \\ r(y_i) &\equiv y_i - \beta_i, \\ r(z_c - z_i) &\equiv (z_c - z_i) - (\beta_c - \beta_i)\,. \end{aligned} \tag{5}$$

Given these residuals, we estimate β_c and β_i which minimize the sum of squared residuals:

$$J_c = \frac{1}{2} w_c r^2(y_c) + \sum_{i\in\Omega_c} w_i r^2(y_i) + \sum_{i\in\Omega_c} w_i r^2(z_c - z_i), \tag{6}$$

where w_c and w_i are given positive weights for pixel c and i, respectively. The function above can be minimized by setting its gradients with respect to the parameters zero:

$$\begin{aligned} \frac{\partial J_c}{\partial \beta_c} &= w_c(y_c - \beta_c) + 2\sum_{i\in\Omega_c} w_i\{(z_c - z_i) - (\beta_c - \beta_i)\} = 0, \\ \frac{\partial J_c}{\partial \beta_i} &= -w_i\,(y_i - \beta_i) + w_i\{(z_c - z_i) - (\beta_c - \beta_i)\} = 0. \end{aligned} \tag{7}$$

By rearranging the second equation with respect to $\beta_i = \frac{1}{2}(y_i - z_c + z_i + \beta_c)$ and plugging this into the first equation, we reach to a combination kernel in a closed-form:

$$\hat{\beta}_c = \frac{1}{W_c}\left(w_c y_c + \sum_{i\in\Omega_c} w_i y_i + \sum_{i\in\Omega_c} w_i\,(z_c - z_i)\right), \tag{8}$$

where W_c is a normalization term (i.e., $W_c = w_c + \sum_{i\in\Omega_c} w_i$). Note that the derived formula (Eq. 8) is indeed equivalent to the weighted sum estimator we already introduced (i.e., Eq. 4). In other words, using $\hat{\beta}_c$ as the output for pixel c minimizes the weighted sum of residuals for *given* w_c and w_i. Note that, since this combination kernel is localized, we can run our kernel per pixel independently to produce all the pixel colors in the final image.

Importance of w_c and w_i. While our formulation allows us to use any positive weights w_c and w_i, the weights should be determined so that the estimation error of our output, $(\hat{\beta}_c - \mu_c)^2$, can be minimized. To highlight the importance of determining proper weights, Fig. 3 shows an example result of our combination kernel. We have generated independent pixel estimates by varying random sequences per pixel, which is a standard approach for MC rendering. For correlated pixel estimates, we render an image sampled with CRN for all the pixels. The baseline approach ((c) of the figure) uses uniform weights (e.g., $w_c = w_i = 1$). In this case, some neighboring pixels that have a low correlation with its center pixel are equally leveraged to get a combined pixel color, which leads to a minor quality improvement compared to the input image. It implies that the correlation among pixel colors can vary significantly, and the uniform weighting cannot handle such a spatially varying nature of correlation. On the other hand, proper weights make a significant improvement over the results with uniform weights by considering locally varying correlations. How to determine the kernel weights

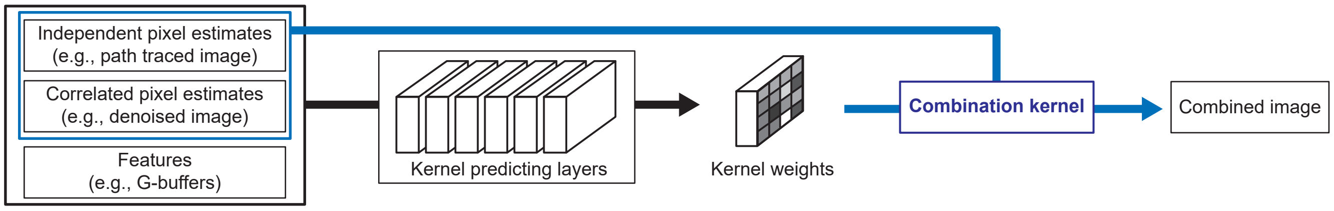

Independent pixel estimates (e.g., path traced image)
Correlated pixel estimates (e.g., denoised image)
Features (e.g., G-buffers)
Kernel predicting layers
Kernel weights
Combination kernel
Combined image

Fig. 4. Overview of our framework. We use a kernel-predicting network that takes both independent and correlated pixel estimates as network inputs. We also pass rendering-specific features (normal, texture, and depth buffers) to the network as additional inputs. The deep-learning network generates the parameters (i.e., weights) for our combination kernel so that our kernel combines the independent and correlated pixel estimates using the weights.

in the direction of reducing the estimation error is discussed in the following section.

4.2 Estimating Kernel via a Neural Network

To evaluate our estimator (Eq. 8), it is ideal for estimating the optimal weights that minimize the error of $\hat{\beta}_c$. To this end, we employ supervised learning that computes estimated optimal parameters for a given reconstruction kernel using training dataset with reference images. This approach is inspired by the work of Kalantari et al. [2015] that estimates optimal parameters for a denoising filter.

Specifically, we train a network to minimize a loss function $\mathcal{L}$ with respect to the parameters $\theta_i \equiv [w_c, w_i]$:

$$\hat{\theta}_i = \underset{\theta_i}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(I_i, g\left(y_i, z_i, f_i; \theta_i\right)\right), \tag{9}$$

where I_i and f_i are a reference color and feature vector (e.g., normal, texture, and depth) at pixel i. The function g is our combined estimator in Eq. 8.

Fig. 4 illustrates an overview of this process, where we use a deep convolutional network to estimate the parameters. Our network architecture is built upon the kernel-predicting convolutional network (KPCN) [Bako et al. 2017], where the last convolution layer produces kernel weights for a reconstruction. Our network, however, is used for entirely different purpose than denoising.

We use six convolution layers, each of which has one hundred of 5×5 convolution kernels. The activation function is the rectified linear unit (ReLU). For the loss function, we use the relative mean squared error [Rousselle et al. 2011]. Note that KPCN allows us to use an arbitrary feature vector, and thus one can exploit additional information (e.g., additional G-buffers from secondary rays) as the network input. Nevertheless, we limit our additional feature f_i to be same as the feature used in KPCN [Bako et al. 2017], to evaluate our main contribution, combined estimator, in a fair manner.

4.3 Error Analysis

We analyze the mean squared error (MSE) of our combined output $\hat{\beta}_c$ (Eq. 8), $E[(\hat{\beta}_c - \mu_c)^2]$, which can be decomposed into the variance $\text{var}(\hat{\beta}_c)$ and squared bias $(E[\hat{\beta}_c] - \mu_c)^2$. To simplify our analysis, we assume that the weights w_c and w_i are given and fixed. Strictly speaking, the weights for our kernel are computed using the independent and correlated pixel estimates, and thus these are also random variables. Therefore our derived errors in this section are an approximation. We further assume that the independent pixel estimates y and the correlated pixel estimates z are independent of each other. Given the assumptions, the bias has the following form (its derivations in Appendix A.1):

$$E[\hat{\beta}_c] - \mu_c \approx \frac{1}{W_c} \sum_{i \in \Omega_c} w_i E\left[\epsilon_{ci}\right], \tag{10}$$

where $E[\epsilon_{ci}] = E[z_c - z_i] - (\mu_c - \mu_i)$. Unlike the statistical model in Eq. 2, here we assume that $E[\epsilon_{ci}]$ can be non-zero. If unbiased correlated pixel estimates (e.g., GPT-L2) are given as our input, the bias is zero regardless of the weights w_i. Otherwise, we would like the weights to be aware of the presence of this bias.

The variance $\text{var}(\hat{\beta}_c)$ can be derived using some elementary manipulations (see Appendix A.2), and it has the form:

$$\begin{aligned} \text{var}\left(\hat{\beta}_c\right) \approx \frac{1}{W_c^2} \Bigg[& w_c^2 \text{var}(y_c) + \sum_{i \in \Omega_c} w_i^2 \text{var}(y_i) + \Bigg(\sum_{i \in \Omega_c} w_i\Bigg)^2 \text{var}(z_c) \\ & + \sum_{i,j \in \Omega_c} w_i w_j \text{cov}(z_i, z_j) - 2\Bigg(\sum_{i \in \Omega_c} w_i\Bigg) \sum_{i \in \Omega_c} w_i \text{cov}\left(z_c, z_i\right) \Bigg]. \end{aligned} \tag{11}$$

Both expressions (Eq. 10 and 11) provide a theoretical interpretation on our combination kernel. For example, the variance expression above indicates that the variance of our result can decrease when including the neighboring pixels i that have a high covariance $cov(z_c, z_i)$ between z_c and z_i. The bias expression shows that the weight w_i should be small when the bias of $z_c - z_i$ is large.

As an alternative to the deep learning approach, one may consider using this error analysis to estimate optimal weights that minimize the MSE. However, we have found that this alternate approach is challenging since MSE representations include ground truth colors and variances. The gap between the expected version $E[(\hat{\beta}_c - \mu_c)^2]$ and actual error $(\hat{\beta}_c - \mu_c)^2$ can also be very large for a pixel c in which outlier samples are included. Our deep learning approach addresses this challenge by learning how to determine the weights to minimize the actual MSE in the training data.

4.4 Multi-Buffered Combination Kernel

We extend our kernel (Eq. 8) to a multi-buffered combination kernel so that our estimator becomes robust against outliers (i.e., spike noise). Technically, our combination kernel can down-weight neighboring pixels i that contain outliers, but the noise can remain if the center pixel c itself is an outlier as the correlation between the pixel c and any of the neighboring pixels will be small in this case.

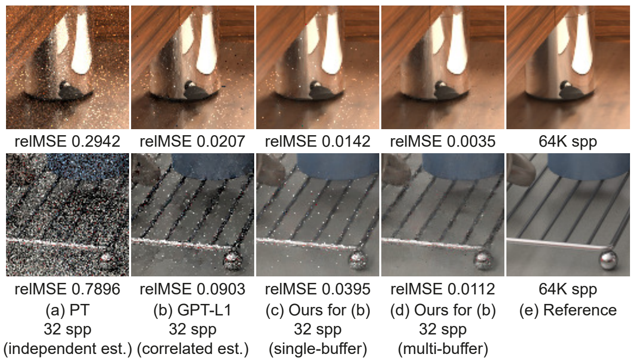

relMSE 0.2942
relMSE 0.0207
relMSE 0.0142
relMSE 0.0035
64K spp
relMSE 0.7896
relMSE 0.0903
relMSE 0.0395
relMSE 0.0112
64K spp
(a) PT 32 spp (independent est.)
(b) GPT-L1 32 spp (correlated est.)
(c) Ours for (b) 32 spp (single-buffer)
(d) Ours for (b) 32 spp (multi-buffer)
(e) Reference

Fig. 5. Results of our single and multi-buffered combination kernels for GPT-L1 reconstruction (b). The single-buffered kernel (c) improves the GPT-L1 results, but it still leaves spike noise. Our extension to a multi-buffered kernel (d) resolves this issue and produces much improved results.

To mitigate this issue, we adopt sample binning [Back et al. 2018]. Sample binning splits a set of samples into mutually exclusive subsets per pixel, and the average of the samples in each subset is stored at each bin. Unless outliers are present in all the bins, we are able to down-weight the sub-averages that contain outliers.

Let us denote $y_{c,b}$ and $y_{i,b}$ are the sub-averages stored at the b-th bin ($b = 1, ..., B$) from independent pixel estimates, and also $z_{c,b}$ and $z_{i,b}$ are the sub-averages from correlated pixel estimates. For the multi-buffered pixel estimates, we extend the least-squares residuals for the single-buffer (Eq. 5) into:

$$\begin{aligned} r(y_{c,b}) &\equiv y_{c,b} - \beta_c, \\ r(y_{i,b}) &\equiv y_{i,b} - \beta_i, \\ r(z_{c,b} - z_{i,b}) &\equiv (z_{c,b} - z_{i,b}) - (\beta_c - \beta_i). \end{aligned} \tag{12}$$

The objective function using the extended residuals is as follows:

$$J_c = \sum_{b=1}^{B} \left[\frac{1}{2} w_{c,b} r^2\left(y_{c,b}\right) + \sum_{i \in \Omega_c} w_{i,b} \left(r^2\left(y_{i,b}\right) + r^2\left(z_{c,b} - z_{i,b}\right) \right) \right] \tag{13}$$

where $w_{c,b}$ and $w_{i,b}$ are the kernel weights allocated to each pixel c and i at the b-th bin, respectively. The least-squares solution that minimizes this objective function can be derived analogously to the single-buffered combination kernel (Eq. 8) and we reach to our combination kernel for multi-buffered inputs

$$\hat{\beta}_c = \frac{1}{W_c} \sum_{b=1}^{B} \left[w_{c,b} y_{c,b} + \sum_{i \in \Omega_c} w_{i,b} y_{i,b} + \sum_{i \in \Omega_c} w_{i,b} \left(z_{c,b} - z_{i,b}\right) \right] \tag{14}$$

where $W_c = \sum_{b=1}^{B} \left(w_{c,b} + \sum_{i \in \Omega_c} w_{i,b}\right)$. This multi-buffered combination kernel is similar to its single-buffered one (Eq. 8), as its final formulation is simply to sum the single-buffered kernel for b-th independent and correlated pixel estimates. Since our process is localized within Ω_c, the total number of parameters ($w_{c,b}$ and $w_{i,b}$) per pixel is $B \times (|\Omega_c| + 1)$. We set $B = 4$ based on an empirical study. The details are in the supplementary document.

Given the formulation of our extended kernel, we need to modify our neural network to produce the extended weights. Fortunately, this extension is straightforward, thanks to the generality of KPCN [Bako et al. 2017]. Specifically, we amend the last convolution layer in the network so that it produces $B \times (|\Omega_c| + 1)$ weights instead of $(|\Omega_c| + 1)$ weights. B-pairs of independent and correlated pixel estimates (instead of the single-pair of pixel estimates) are also fed into the network. Fig. 5 compares the results from single and multi-buffered kernels. The single-buffered kernel (Eq. 8) reduces the remaining errors in the GPT-L1 results, but still leaves some spike noise. On the other hand, our multi-buffered kernel (Eq. 14) removes the spike noise more effectively.

Fig. 6. Training and validation scenes.

5 EXPERIMENTAL SETUP

Network Input. We use path tracing to generate independent pixel estimates and various existing methods to generate correlated pixel estimates. For denoising methods (e.g., NFOR, KPCN, GPT-L1 and GPT-L2), correlated pixel estimates are generated by denoising the independent pixel estimates (i.e., the path-traced image). In this case, we *do not* need to take additional samples, since the actual input to both our framework and denoising methods is the same (i.e., noisy path-traced image). For correlated sampling such as CRN, we need to take additional samples to generate correlated pixel estimates. In this case, we used the same sample count for both independent and correlated pixel estimates (i.e., $N/2$ for independent estimates and $N/2$ for correlated estimates given the total N samples). In addition to the independent and correlated pixel estimates, we also put G-buffers (normal, texture, and depth buffers), obtained during rendering easily, to the network so that estimated kernel weights take account of the high-frequency details captured by the features.

For a multi-buffered combination kernel, B independent and correlated pixel estimates are passed to the network. For the independent pixel estimates and non-denoising methods (e.g., CRN), we split samples into B buffers uniformly and store the averages of the samples in each sub-buffer (e.g., $y_{c,b}$ for independent pixel estimates and $z_{c,b}$ for non-denoising methods). For the denoising methods (e.g., NFOR, KPCN, GPT-L1 and GPT-L2) that compute their denoising weights, we compute the denoising kernel using all the samples first, then share this kernel among all B buffers. Each b-th value at pixel c is computed as $z_{c,b} = \bar{w}_c y_{c,b} + \sum_{i \in \Omega_c} \bar{w}_i y_{i,b}$ without recomputing the weights separately. For the other denoising methods (e.g., Bayesian Collaborative Denoising (BCD) [Boughida and Boubekeur 2017]) that do not build its pixel-based weights explicitly, we apply the denoising into each of partially-averaged independent pixel estimates, and use the denoised images as correlated pixel estimates.

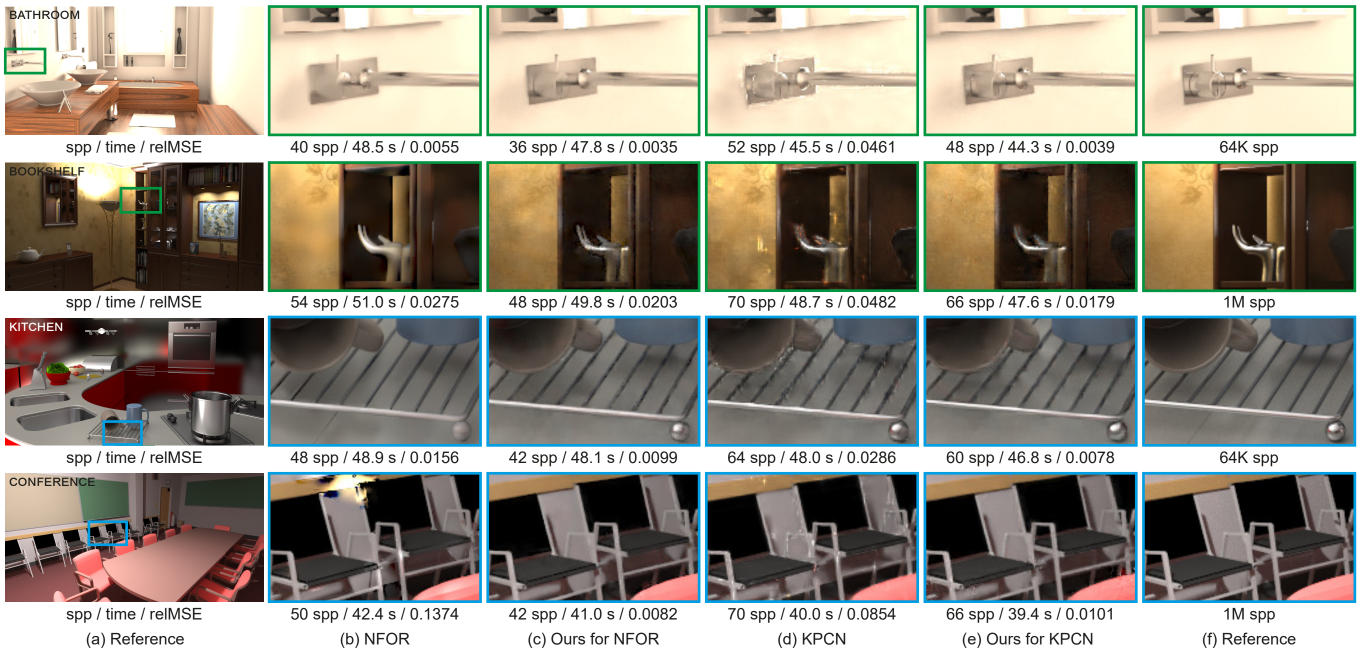


Fig. 7. Equal-time comparisons with NFOR and KPCN. Our framework takes the results of the denoisers and their inputs as our inputs, and reduces the errors in the correlated pixel estimates. For example, we reduce the bias in the over-blurred regions (e.g., the rack in KITCHEN for NFOR and the shadows in CONFERENCE for KPCN). Also, our combination kernel removes the splotches or spike noise for the CONFERENCE scene.

Training Details. We trained a single neural network that infers the parameters of our combination kernel using independent and correlated pixel estimates. We did not train a separate network per tested method to demonstrate the generality of our framework across different types of methods. The dataset contains input-reference pairs for six scenes (in Fig. 6) from random views. We used five types of correlated pixel estimates from CRN, GPT-L1 and GPT-L2 [Kettunen et al. 2015], NFOR [Bitterli et al. 2016], and KPCN [Bako et al. 2017]. Input images generated with 64 or 256 samples per pixel, and path tracing with $16K$ samples per pixel was used to produce the reference images. We extracted 276 image patches of 128×128 size randomly from 1280×720 images. Given the configuration, we rendered 240 images for each method, and thus the total number of images is 1200. We then split the dataset into training (70% of the data) and validation (30% of the frames) sets. Note that test scenes, used for comparing our method with existing approaches, are not included in the training dataset. We implemented our network using Tensorflow [Abadi et al. 2015] and trained it on a Linux machine with NVidia GTX 1080 Ti graphics card. The size of mini-batch was set to ten, and we trained our network with Adam optimizer [Kingma and Ba 2014] with a learning rate of 0.0001 for 50 epochs and it took 24 hours for training time. We use a combination window size of 15×15, and thus the number of our kernel weights per pixel is $4 \times (|\Omega_c| + 1) = 900$ given $B = 4$.

6 RESULTS AND DISCUSSIONS

We have tested our framework on a PC with Intel Xeon CPU E5-2687W and NVidia GTX 1080 Ti graphics card. Specifically, we demonstrate that our technique is orthogonal to different types of existing methods categorized in Sec. 2 while reducing heterogeneous errors of the methods. Particularly, we evaluate our method with different types of input with correlated pixels: 1) denoising (NFOR and KPCN) for independent pixel estimates, 2) images rendered by correlated sampling (CRN), and 3) images generated by gradient-domain sampling and reconstruction (GPT-L1 and GPT-L2).

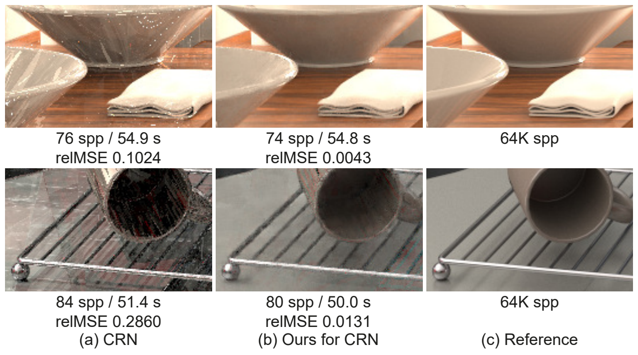


Fig. 8. Same-time comparisons with CRN, which introduces unbiased but structured noise. The reported spp for our method is the sum of the number of samples used for generating our independent and correlated pixel estimates. Our technique combines independent and correlated pixel estimates via a weighted combination kernel and removes the structured noise effectively while exploiting the inter-pixel correlation in the CRN images.

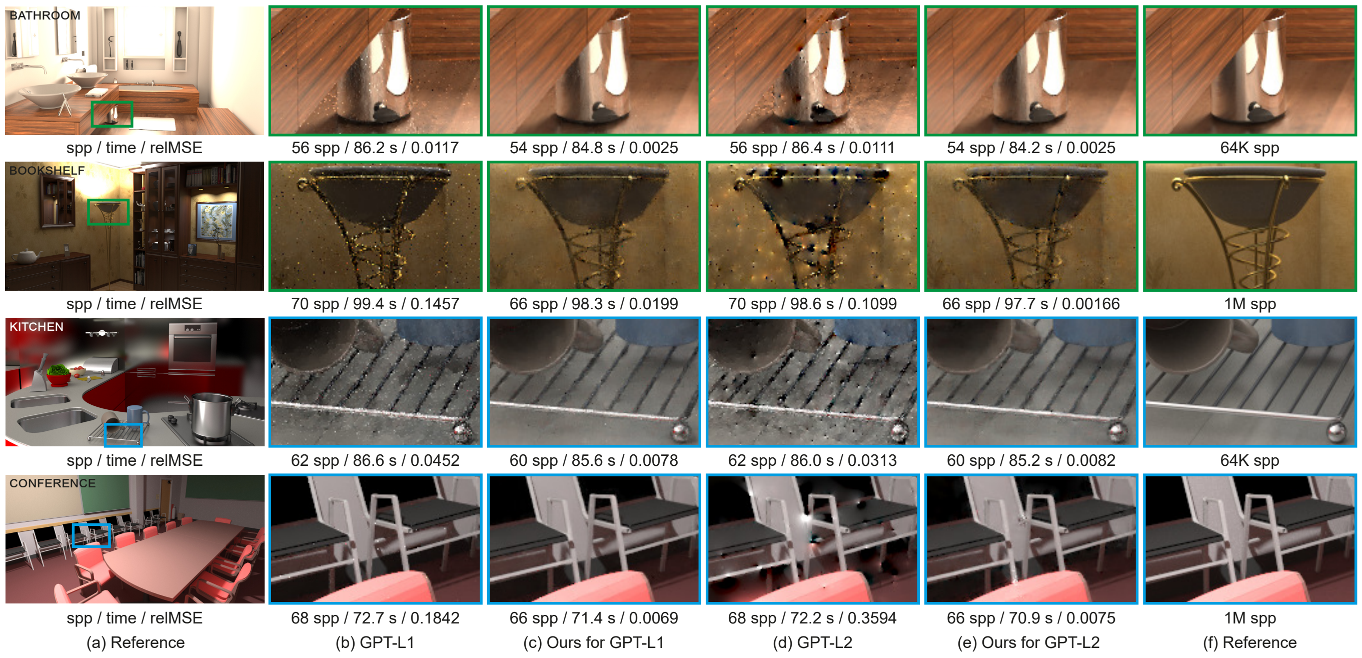


Fig. 9. Equal-time comparisons with the gradient-domain rendering. Our method reduces the dipole artifacts in GPT-L2 and spike noise in GPT-L1, and also restores the lost energy in GPT-L1 for the BOOKSHELF scene.

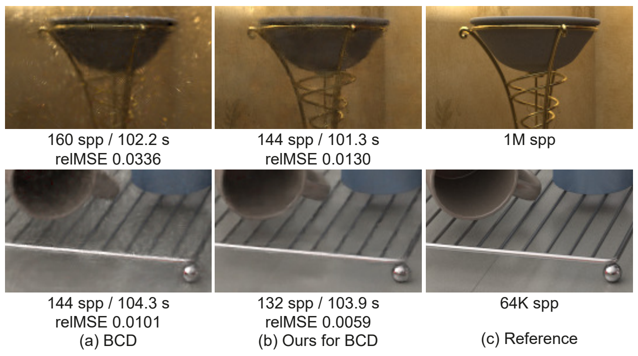


Fig. 10. Equal-time comparisons with Bayesian Collaborative Denoising (BCD). Despite not being trained on correlated pixel estimates from BCD, our combination for BCD can restore the high-frequency details while reducing the residual noise in the scenes.

Additionally, we test two methods, Bayesian Collaborative Denoising (BCD) [Boughida and Boubekeur 2017] and Progressive Photon Mapping (PPM) [Hachisuka et al. 2008], which were not trained with our network. For a fair comparison with KPCN, we retrained its network using our training data (Fig. 6).

We render four test scenes (BATHROOM, BOOKSHELF, KITCHEN, and CONFERENCE). All the scenes have challenging lighting settings where most of the parts are illuminated by indirect lighting. For the BOOKSHELF and CONFERENCE scenes, we test how our method enhances existing techniques as they exhibit spike noise, which remains a challenging problem for denoising techniques. Most parts in the KITCHEN scene are glossy, which introduces glossy interreflections and highlights. It can be challenging to denoising methods because G-buffers cannot capture such illumination effects.

We compare the performance of all methods using the relative mean squared error (relMSE) [Rousselle et al. 2011]. Each method has been executed ten times, and we report its average error. When our technique is compared with existing methods, we also report the total rendering time, which include both sampling and reconstruction time. Note that our framework incurs an overhead, which includes the inference time for combining independent and correlated pixel estimates. For example, the inference of our network (i.e., combining time) takes 1.4 seconds on average, given the tested 1280 × 720 image resolution.

Equal-Time Comparisons with Image Denoising. Fig. 7 shows that NFOR tends to blur high-frequency details at locations where the feature buffers fail to capture such information, e.g., the glossy highlights in BOOKSHELF and the rack in KITCHEN. Our method restores such details when taking the result of NFOR as our input. KPCN produces sharper images than the NFOR due to its non-linear weights learned by a neural network. However, it leaves some residual noise (e.g., the structure in BOOKSHELF) or blurs some high-frequency details (e.g., the shadows in the CONFERENCE). Our framework enhances the KPCN results while removing residual noise and reducing its bias. For challenging scenes such as BOOKSHELF and CONFERENCE, our multi-buffered combination kernel removes the outlier artifacts in NFOR and KPCN. We emphasize that the training scenes for KPCN and ours are the same.

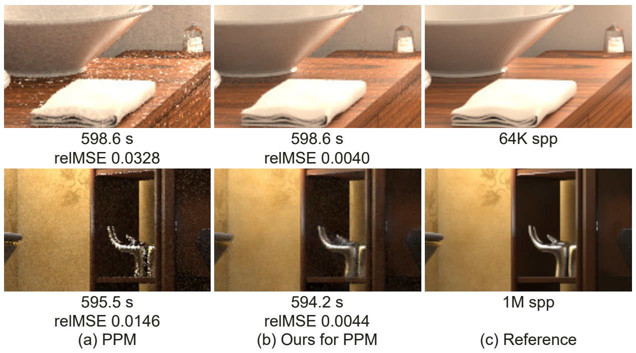


Fig. 11. Equal-time comparisons with Progressive Photon Mapping (PPM). Our method (b) combines independent pixel estimates from path tracing and correlated pixel estimates from PPM, and results in much-reduced noise in both high and low frequencies compared to the original PPM (a).

Equal-Time Comparisons with Correlated Sampling. As shown in Fig. 8, while CRN introduces inter-pixel correlation to reduce random noise, such correlated sampling suffers from structured artifacts. However, when the pair of the independent and correlated pixel estimates are combined via our kernel, the combined results have much fewer artifacts than the CRN results.

Equal-Time Comparisons with Gradient-Domain Rendering. In Fig. 9, we evaluate our method with GPT-L2 and GPT-L1. The GPT-L2 produces unbiased results, but the results are not visually pleasing due to its dipole artifact. When our technique is applied to the GPT-L2 reconstruction, the final output is biased. Nevertheless, we significantly improve its result both numerically and visually while removing the dipole artifacts. GPT-L1 reconstruction, recommended by [Kettunen et al. 2015], outputs biased but more visually pleasing results than the GPT-L2 results. However, this reconstruction suffers from energy loss, especially at locations where many outliers exist (see BOOKSHELF). Our technique, applied to the GPT-L1 reconstruction, restores noticeable energy loss and thus reduces its bias. Technically, the approximation error, $bias(z_c - z_i)$ of our statistical model for correlated pixel estimates (Eq. 2) can be much lower than the biases of the correlated pixel colors, since the biases, $bias(z_c)$ and $bias(z_i)$ at center pixel c and i, tend to be similar for the energy loss case. Furthermore, our method reduces the remaining spike noise in GPT-L1 results, thanks to our multi-buffered kernel.

Equal-Time Comparisons with Untrained Correlated Pixel Estimates. We evaluate how our framework generalizes to two unseen correlated pixel estimates in training: Bayesian Collaborative Denoising (BCD) [Boughida and Boubekeur 2017] and Progressive Photon Mapping (PPM) [Hachisuka et al. 2008]. As shown in Fig. 10, our combination for BCD restores blurred high-frequency details (see the BOOKSHELF scene) and reduces residual noise effectively.

Our method mainly aims to improve the results of the existing methods (e.g., NFOR) that use path tracing as the core light transport algorithm, but applying our combination kernel to photon mapping [Jensen 1996], can also be interesting since the biased light transport algorithm can introduce a correlation due to its density estimation. Fig. 11 shows same-time comparisons with a consistent

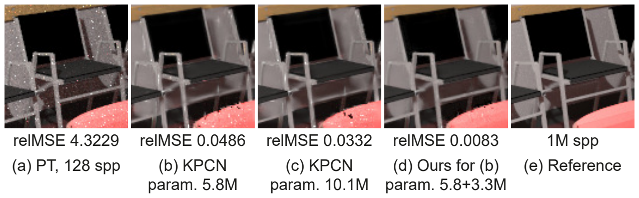


Fig. 12. A comparison with the modified KPCN (c) that has a larger number of parameters (10.1M) than that of our network combined with the original KPCN (total 9.1M). Our combination still outperforms the modified KPCN significantly, showing that our improved result is achieved by exploiting inter-pixel correlations, not just simply by the increased network capacity.

photon mapping algorithm (i.e., PPM). We set its kernel reduction ratio to 0.7 and use $1M$ photons per its photon generation pass. We use multiple primary rays per pixel for PPM mainly for the anti-aliasing and split the primary sample colors evenly to B buffers, which are the correlated input images to our network. We generate independent pixel estimates using path tracing, and for this process, we adjust its sample count per pixel so that its sampling time is equal to the time spent for our other input (i.e., correlated pixel estimates). As shown in Fig. 11, our combined results have much fewer residual errors (e.g., low-frequency artifacts) than the results of the previous technique, even without any additional training.

Network Capacity. In our experiments, the capacity of our network, denoted by the number of network parameters, is smaller than typical learning based denoisers, e.g., KPCN. However, when our approach is combined with a learning-based denoiser, the total capacity becomes larger than the capacity of the input denoiser. In Fig. 12, we compare our combination with a modified KPCN with a larger capacity to verify that our improvement does not directly come from the increased capacity. Specifically, we have retrained the KPCN with an increased number of kernels (from 100 to 140) in each layer so that the modified KPCN has a larger number of parameters (10.1M) than our combination (9.1M for both our network and the input denoiser). As can be seen, the modified KPCN results in an improved result compared to the original one, but this alone still has a much higher error than our method.

Numerical Convergence. Fig. 13 shows the convergence of four denoising methods (NFOR, KPCN, GPT-L1 and GPT-L2) with and without our technique, where relMSE values are reported over running times on a log-log scale. The convergence results indicate that our method consistently improves the numerical accuracy of the existing methods. For the CONFERENCE scene where severe outliers present, the existing methods have some fluctuations even if we average relMSE values from multiple runs (e.g., ten times). Our method has smaller variations over time thanks to the outlier handling with our multi-buffered kernel.

Relation to Image Boosting. When our technique is applied to a denoising, our method has a similarity with boosting [Milanfar 2013; Talebi et al. 2012] which reduces a denoising bias at the expense of

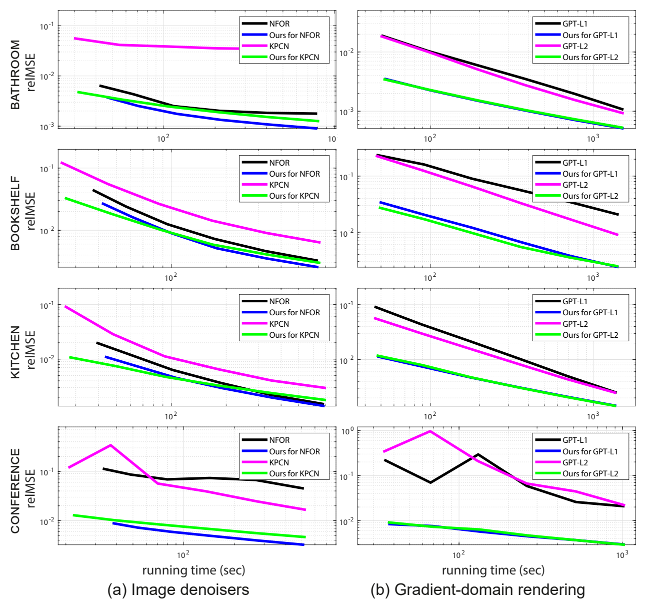


Fig. 13. Numerical convergences over time on a log-log scale. Our method consistently improves numerical accuracy of the existing methods.

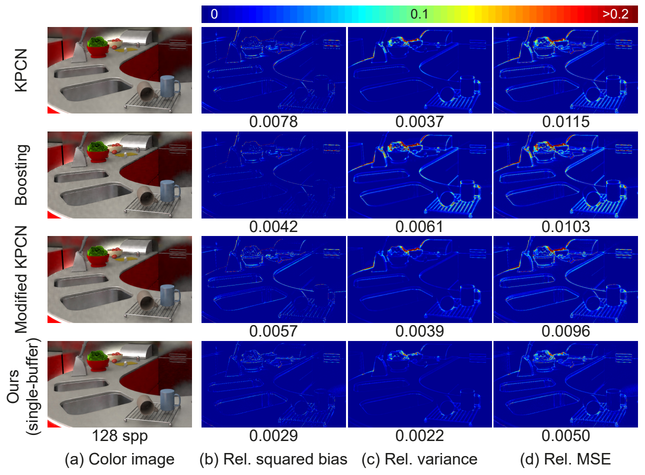


Fig. 14. Comparisons with image boosting techniques for KPCN. We visualize the relative errors that scale the squared bias (b), variance (c), and MSE (d) by a factor of $1/(\mu_c^2 + 0.01)$.

increasing a variance as a post-denoising. Specifically, our single-buffered kernel (Eq. 8) can be reformulated into:

$$\hat{\beta}_c = z_c + \frac{1}{W_c} \sum_{i \in \Omega_c'} w_i (y_i - z_i), \tag{15}$$

where we change the set Ω_c of the neighboring pixels into Ω_c' that includes the pixel c. Our reformulated equation can be interpreted as a form of image boosting, as it applies a kernel w_i to the denoising residuals, $y_i - z_i$, and adds the processed residuals to the input value z_c. Nonetheless, the main difference is in determining the weights

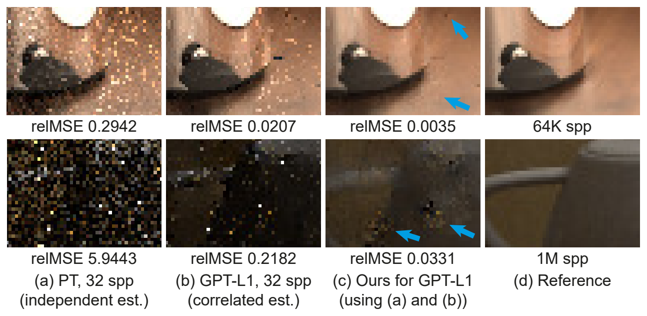

relMSE 0.2942
relMSE 0.0207
relMSE 0.0035
64K spp
relMSE 5.9443
relMSE 0.2182
relMSE 0.0331
1M spp
(a) PT, 32 spp (independent est.)
(b) GPT-L1, 32 spp (correlated est.)
(c) Ours for GPT-L1 (using (a) and (b))
(d) Reference

Fig. 15. Failure cases. Our combination kernel assumes enough correlation exists in a local image region, but our correlated input (b) from GPT-L1 reconstruction does not have enough correlation. Our method improves the existing method by handling both noise and bias, but our result (c) still contains noticeable noise due to the lack of enough correlation.

w_i. A typical form of boosting assigns the weights w_i to be the same with the weights of an input denoiser. Our method, however, does not restrict the weights to be the same as a denoising method, and a separate network estimates the parameters in order to reduce both the bias and variance in a denoised image. Fig. 14 visualizes the squared bias and variance of our single-buffered kernel and boosting that uses the same weights for the KPCN denoiser. In addition to simple boosting, we have also tested an extended KPCN (third row in Fig. 14), where we have retrained a modified KPCN network to include a boosting layer so that its predicted kernel can be further optimized.

As shown in the figure, boosting (second row) reduces the bias of its input denoiser (i.e., KPCN) but results in a higher variance. The modified KPCN (third row) produces numerically improved results due to its lower variance than boosting, but its variance is still slightly higher than that of the original KPCN (first row). Note that the bias term dominates the relative MSE in this scene. For scenes (e.g., the BOOKSHELF scene) where strong outliers exist, we found that both boosting variants introduced higher MSE values than an unmodified KPCN due to a noticeable increase in the variance term. On the other hand, our technique reduces both errors effectively and produces more accurate results than the tested alternatives. Moreover, our approach can be exploited in more general scenarios (not just a post-denoising).

Limitations and Future Work. Our method relies on the assumption that there is enough correlation in correlated pixel estimates. Fig. 15 shows failure cases of our assumption, where positive correlations among pixels are not enough for reducing the residual variance effectively. In this case, our method leaves some remaining noise. To analyze our technique more thoroughly with the cases where our statistical assumption breaks, we conduct a study while varying the inputs to the network, as shown in Fig. 16. The fifth and sixth columns ((e) and (f) in Fig. 16) show the network outputs when there is no inter-pixel correlation in the correlated pixel estimates z. Specifically, the results in Fig. 16 (e) are generated by making the z to be the same with the independent pixel estimates y, and the z in the other case (Fig. 16 (f)) is set with another path traced

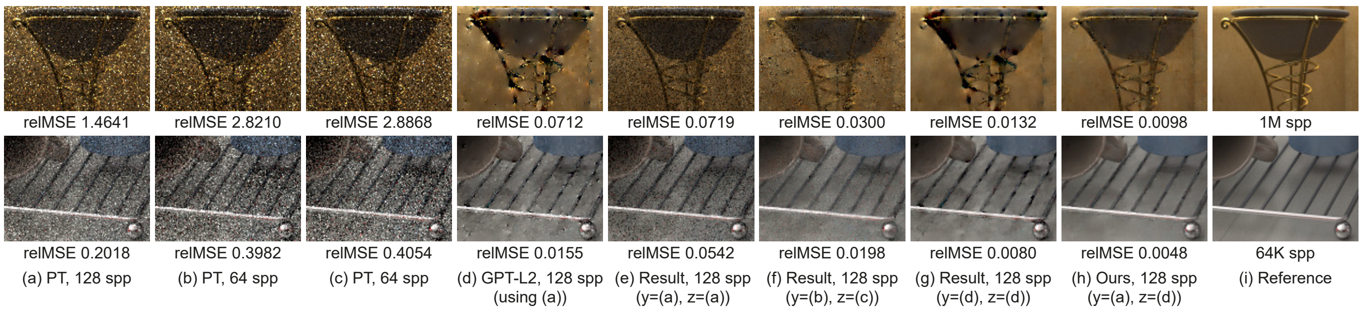


Fig. 16. Results by varying the independent y and correlated pixel estimates z, which are the inputs to our combination kernel. This study shows the worst cases of our combination kernel, where the key assumptions, inter-pixel independence and correlation in y and z break. The results in (e) and (f) show a case where there is no correlation among adjacent pixels in z. The outputs in (g) are generated from another worst scenario where independence assumption for y breaks. The combination kernel in the extreme cases produces high-frequency noise ((e) and (f)) or fails to remove correlated artifacts (g), and thus its results have higher numerical errors than our results (h) generated with valid inputs.

image. In both cases, the combination results leave very noticeable high-frequency noise due to the absence of inter-pixel correlation.

We also test a scenario where the independence assumption on the y breaks. Specifically, we generate the results in the seventh column ((g) in Fig. 16) by setting the y with a correlated image (e.g., GPT-L2 results). In this case, the combination kernel fails to effectively reduce the correlated artifacts, as our kernel does not model the covariance terms of adjacent pixel colors in the y. This study indicates that the performance of our technique heavily depends on the assumptions. This is mainly because our combination kernel is designed to exploit inter-pixel independence and correlation in the y and z, respectively. A related future research direction would be designing a sampling technique that introduces a strong correlation among adjacent pixels, as the enhanced sampling can make our combination stronger.

We propose a new combination kernel whose parameters rely on a deep learning method, and thus our result inherits its fundamental problem, i.e., generalization. We demonstrate our technique behaves well for the different types of correlated pixel estimates, even though our learning architecture is built upon the simple network [Bako et al. 2017]. It, however, does not indicate that our chosen network architecture is ideal. We would like to explore more powerful architectures (e.g., U-Net [Ronneberger et al. 2015]) to make our combinations more effective. Additionally, we would like to investigate a spatio-temporal extension of our kernel so that temporal correlation in adjacent frames can be exploited.

7 CONCLUSION

In this work, we propose a unified framework for combining independent and correlated pixels estimates in Monte Carlo rendering. Our key observation is that previous techniques introduce pixel correlation in their output images, either directly via correlated sampling or indirectly via image denoising. Our framework exploits such correlation, leading to improved performance when being applied as a post-processing step on existing correlated sampling and denoising methods. As a key technical contribution, we formulate a weighted combination kernel with a neural network that deals with heterogeneous inter-pixel correlations in correlated pixel estimates. We also extend our kernel to support multiple buffers, making it more robust to outliers. We design our framework to be orthogonal to correlated sampling and denoising techniques and demonstrate its robustness by applying it to a wide variety of methods. We show that our framework can be complementary to the prior methods while reducing their reconstruction errors.

ACKNOWLEDGMENTS

We appreciate the reviewers for their valuable comments (especially for a connection with image boosting). We also thank Wenzel Jakob for Mitsuba renderer [Jakob 2010], Benedikt Bitterli for public scenes [Bitterli 2016], and the following authors and artists for each scene: Mareck, Jay-Artist, Wig42, NovaZeeke, Eric Veach, Miika Aittala, Samuli Laine, Jaakko Lehtinen and Tiziano Portenier (training scenes in Fig. 6), nacimus (BATHROOM), Tiziano Portenier (BOOKSHELF), Anton Kaplanyan (KITCHEN), Anat Grynberg and Greg Ward (CONFERENCE). Bochang Moon is a corresponding author of the paper. This work was supported by NRF (MSIT, 2020R1A2C4002425) and Cross-Ministry Giga KOREA grants (MSIT, GK20P0300).

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.

Jonghee Back, Sung-Eui Yoon, and Bochang Moon. 2018. Feature Generation for Adaptive Gradient-Domain Path Tracing. *Computer Graphics Forum* 37, 7 (2018), 65–74.

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Trans. Graph.* 36, 4, Article 97 (2017), 14 pages.

Pablo Bauszat, Victor Petitjean, and Elmar Eisemann. 2017. Gradient-Domain Path Reusing. *ACM Trans. Graph.* 36, 6, Article 229 (2017), 9 pages.

Philippe Bekaert, Mateu Sbert, and John Halton. 2002. Accelerating Path Tracing by Re-Using Paths. In *Proceedings of the 13th Eurographics Workshop on Rendering (EGRW '02)*. 125–134.

Pravin Bhat, Brian Curless, Michael Cohen, and C. Lawrence Zitnick. 2008. Fourier Analysis of the 2D Screened Poisson Equation for Gradient Domain Problems. In

Proceedings of the 10th European Conference on Computer Vision: Part II (ECCV '08). 114–128.

Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/.

Benedikt Bitterli, Fabrice Rousselle, Bochang Moon, José A. Iglesias-Guitián, David Adler, Kenny Mitchell, Wojciech Jarosz, and Jan Novák. 2016. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. *Computer Graphics Forum* 35, 4 (2016), 107–117.

Malik Boughida and Tamy Boubekeur. 2017. Bayesian Collaborative Denoising for Monte Carlo Rendering. *Computer Graphics Forum* 36, 4 (2017), 137–153.

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article 98 (2017), 12 pages.

Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. 2019. Sample-Based Monte Carlo Denoising Using a Kernel-Splatting Network. *ACM Trans. Graph.* 38, 4, Article 125 (2019), 12 pages.

Jie Guo, Mengtian Li, Quewei Li, Yuting Qiang, Bingyang Hu, Yanwen Guo, and Ling-Qi Yan. 2019. GradNet: Unsupervised Deep Screened Poisson Reconstruction for Gradient-Domain Rendering. *ACM Trans. Graph.* 38, 6, Article 223 (2019), 13 pages.

Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. 2008. Progressive Photon Mapping. *ACM Trans. Graph.* 27, 5, Article 130 (2008), 8 pages.

Binh-Son Hua, Adrien Gruson, Victor Petitjean, Matthias Zwicker, Derek Nowrouzezahrai, Elmar Eisemann, and Toshiya Hachisuka. 2019. A Survey on Gradient-Domain Rendering. *Computer Graphics Forum* 38, 2 (2019), 455–472.

Wenzel Jakob. 2010. Mitsuba renderer.

Henrik Wann Jensen. 1996. Global Illumination using Photon Maps. In *Rendering Techniques '96*. Springer Vienna, Vienna, 21–30.

James T. Kajiya. 1986. The rendering equation. In *ACM SIGGRAPH '86*. 143–150.

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Trans. Graph.* 34, 4, Article 122 (2015), 12 pages.

Alexander Keller and Wolfgang Heidrich. 2001. Interleaved Sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*. 269–276.

Markus Kettunen, Erik Härkönen, and Jaakko Lehtinen. 2019. Deep Convolutional Reconstruction for Gradient-Domain Rendering. *ACM Trans. Graph.* 38, 4, Article 126 (2019), 12 pages.

Markus Kettunen, Marco Manzi, Miika Aittala, Jaakko Lehtinen, Frédo Durand, and Matthias Zwicker. 2015. Gradient-domain Path Tracing. *ACM Trans. Graph.* 34, 4, Article 123 (2015), 13 pages.

Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (2014).

Jaakko Lehtinen, Tero Karras, Samuli Laine, Miika Aittala, Frédo Durand, and Timo Aila. 2013. Gradient-domain Metropolis Light Transport. *ACM Trans. Graph.* 32, 4, Article 95 (2013), 12 pages.

Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012. SURE-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph.* 31, 6, Article 194 (2012), 9 pages.

Michael D. McCool. 1999. Anisotropic Diffusion for Monte Carlo Noise Reduction. *ACM Trans. Graph.* 18, 2 (1999), 171–194.

Peyman Milanfar. 2013. A Tour of Modern Image Filtering: New Insights and Methods, Both Practical and Theoretical. *IEEE Signal Processing Magazine* 30, 1 (2013), 106–128.

Bochang Moon, Nathan Carr, and Sung-Eui Yoon. 2014. Adaptive Rendering Based on Weighted Local Regression. *ACM Trans. Graph.* 33, 5, Article 170 (2014), 14 pages.

Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. 2016. Adaptive Polynomial Rendering. *ACM Trans. Graph.* 35, 4, Article 40 (2016), 10 pages.

Ryan S. Overbeck, Craig Donner, and Ravi Ramamoorthi. 2009. Adaptive Wavelet Rendering. *ACM Trans. Graph.* 28, 5, Article 140 (2009), 12 pages.

Matt Pharr. 2018. Guest Editor's Introduction: Special Issue on Production Rendering. *ACM Trans. Graph.* 37, 3, Article 28 (2018), 4 pages.

O. Ronneberger, P.Fischer, and T. Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI) (LNCS)*, Vol. 9351. Springer, 234–241.

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2011. Adaptive Sampling and Reconstruction Using Greedy Error Minimization. *ACM Trans. Graph.* 30, 6, Article 159 (2011), 12 pages.

Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2012. Adaptive Rendering with Non-local Means Filtering. *ACM Trans. Graph.* 31, 6, Article 195 (2012), 11 pages.

Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. 2013. Robust Denoising using Feature and Color Information. *Computer Graphics Forum* 32, 7 (2013), 121–130.

Iman Sadeghi, Bin Chen, and Henrik Wann Jensen. 2009. Coherent path tracing. *Journal of Graphics, GPU, and Game Tools* 14, 2 (2009), 33–43.

Pradeep Sen and Soheil Darabi. 2012. On Filtering the Noise from the Random Parameters in Monte Carlo Rendering. *ACM Trans. Graph.* 31, 3, Article 18 (2012), 15 pages.

Hossein Talebi, Xiang Zhu, and Peyman Milanfar. 2012. How to SAIF-ly boost denoising performance. *IEEE transactions on image processing* 22 (12 2012), 16.

Bing Xu, Junfei Zhang, Rui Wang, Kun Xu, Yong-Liang Yang, Chuan Li, and Rui Tang. 2019. Adversarial Monte Carlo Denoising with Conditioned Auxiliary Feature Modulation. *ACM Trans. Graph.* 38, 6, Article 224 (2019), 12 pages.

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. 2015. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Computer Graphics Forum* 34, 2 (2015), 667–681.

A APPENDIX

A.1 Bias of Our Combination Kernel

The expectation of $\hat{\beta}_c$ (Eq. 8) can be represented in the form:

$$\begin{aligned}
E[\hat{\beta}_c] &= E\left[\frac{1}{W_c}\left(w_c y_c + \sum_{i\in\Omega_c} w_i y_i + \sum_{i\in\Omega_c} w_i\,(z_c - z_i)\right)\right]\\
&\approx \frac{1}{W_c}\left(w_c E\,[y_c] + \sum_{i\in\Omega_c} w_i E\,[y_i] + \sum_{i\in\Omega_c} w_i (E\,[z_c - z_i])\right)\\
&= \frac{1}{W_c}\left(w_c \mu_c + \sum_{i\in\Omega_c} w_i \mu_i + \sum_{i\in\Omega_c} w_i\,(\mu_c - \mu_i + E\,[\epsilon_{ci}])\right)\\
&= \mu_c + \frac{1}{W_c}\sum_{i\in\Omega_c} w_i E\,[\epsilon_{ci}]\,.
\end{aligned}$$

In the equation above, $E[\epsilon_{ci}] = E[z_c - z_i] - (\mu_c - \mu_i)$. The equation in the second line is approximated from the one in the first line, by assuming that the weights w_c and w_i are constant. Consequently, the bias of our combination kernel, $E[\hat{\beta}_c] - \mu_c \approx \frac{1}{W_c}\sum_{i\in\Omega_c} w_i E\,[\epsilon_{ci}]$, which is a weighted sum of $E[\epsilon_{ci}]$.

A.2 Variance of Our Combination Kernel

The variance $\mathrm{var}(\hat{\beta}_c)$ of our estimator can be approximated under two assumptions (constant weights and independence between y and z) as the following:

$$\begin{aligned}
\mathrm{var}(\hat{\beta}_c) &= \mathrm{var}\left[\frac{1}{W_c}\left(w_c y_c + \sum_{i\in\Omega_c} w_i y_i + \sum_{i\in\Omega_c} w_i\,(z_c - z_i)\right)\right]\\
&\approx \frac{1}{W_c^2}\left[w_c^2 \mathrm{var}(y_c) + \sum_{i\in\Omega_c} w_i^2 \mathrm{var}(y_i) + \mathrm{var}\left(\sum_{i\in\Omega_c} w_i\,(z_c - z_i)\right)\right].
\end{aligned}$$

In the equation above, the last term contains the correlated random variables z_c and z_i, and thus this can be further decomposed by considering the covariance between the values:

$$\begin{aligned}
\mathrm{var}\left(\sum_{i\in\Omega_c} w_i (z_c - z_i)\right) &= \mathrm{var}\left(\sum_{i\in\Omega_c} w_i z_c\right) + \mathrm{var}\left(\sum_{i\in\Omega_c} w_i z_i\right)\\
&\quad - 2\mathrm{cov}\left(\sum_{i\in\Omega_c} w_i z_c, \sum_{i\in\Omega_c} w_i z_i\right)\\
&= \left(\sum_{i\in\Omega_c} w_i\right)^2 \mathrm{var}(z_c) + \sum_{i,j\in\Omega_c} w_i w_j \mathrm{cov}(z_i, z_j)\\
&\quad - 2\left(\sum_{i\in\Omega_c} w_i\right)\sum_{i\in\Omega_c} w_i \mathrm{cov}\,(z_c, z_i)\,.
\end{aligned}$$

By plugging this into the equation (Eq. 11), we have the final variance representation for our combination kernel.