Self-Supervised Post-Correction for Monte Carlo Denoising

Jonghee Back
Gwangju Institute of Science and Technology
South Korea
jongheeback@gm.gist.ac.kr

Binh-Son Hua
VinAI Research
Vietnam
binhson.hua@gmail.com

Toshiya Hachisuka
University of Waterloo
Canada
toshiya.hachisuka@uwaterloo.ca

Bochang Moon
Gwangju Institute of Science and Technology
South Korea
moonbochang@gmail.com

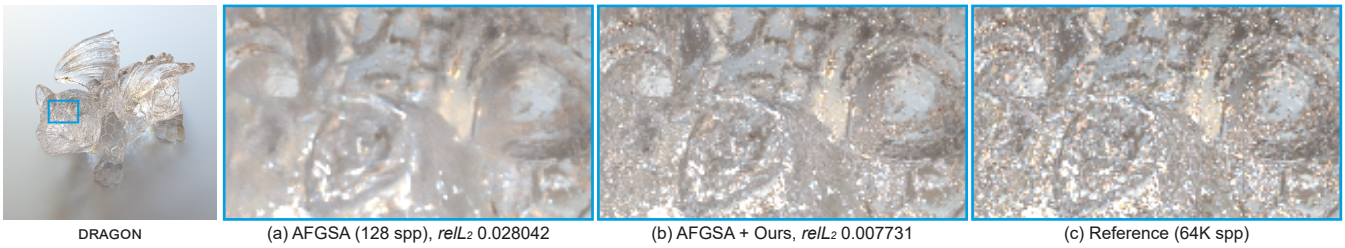

Figure 1: Our post-correction result for a state-of-the-art image denoiser (AFGSA [Yu et al. 2021]). Our self-supervised optimization uses only test images, and it visually and numerically improves the existing learning-based technique by restoring the fine details. We used the relative L_2 ($relL_2$) [Rousselle et al. 2011] as an error metric.

ABSTRACT

Using a network trained by a large dataset is becoming popular for denoising Monte Carlo rendering. Such a denoising approach based on supervised learning is currently considered the best approach in terms of quality. Nevertheless, this approach may fail when the image to be rendered (i.e., the test data) has very different characteristics than the images included in the training dataset. A pre-trained network may not properly denoise such an image since it is unseen data from a supervised learning perspective. To address this fundamental issue, we introduce a post-processing network that improves the performance of supervised learning denoisers. The key idea behind our approach is to train this post-processing network with self-supervised learning. In contrast to supervised learning, our self-supervised model does not need a reference image in its training process. We can thus use a noisy test image and self-correct the model on the fly to improve denoising performance. Our main contribution is a self-supervised loss that can guide the post-correction network to optimize its parameters without relying on the reference. Our work is the first to apply this self-supervised learning concept in denoising Monte Carlo rendered estimates. We demonstrate that our post-correction framework can boost supervised denoising via our self-supervised optimization. Our implementation is available at https://github.com/CGLab-GIST/self-supervised-post-corr.

CCS CONCEPTS

• **Computing methodologies** → **Ray tracing**.

KEYWORDS

self-supervised learning, self-supervised loss, Monte Carlo denoising, self-supervised denoising

ACM Reference Format:
Jonghee Back, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. 2022. Self-Supervised Post-Correction for Monte Carlo Denoising. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings (SIGGRAPH '22 Conference Proceedings), August 7–11, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3528233.3530730

1 INTRODUCTION

Monte Carlo (MC) image denoising is a general approach that can effectively reduce the variance of noisy pixel estimates [Zwicker et al. 2015]. A traditional denoiser takes path-traced images as input and produces more accurate pixel estimates by replacing a noisy pixel estimate with a weighted sum of neighboring pixels [Bitterli et al. 2016]. Sophisticated neural networks for image denoising [Bako et al. 2017; Xu et al. 2019; Yu et al. 2021] have been actively studied and demonstrated effective at denoising MC estimates.

A widely adopted approach for optimizing learning-based denoisers is to pretrain a neural network with a training dataset and then use the trained network to infer pixel estimates at runtime.

SIGGRAPH '22 Conference Proceedings, August 7–11, 2022, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9337-9/22/08…$15.00
https://doi.org/10.1145/3528233.3530730

This approach is said to be made robust by carefully selecting training images that are *similar* to the test images. However, it is not practical to assume that the training dataset covers all the possible test images (i.e., a noisy input image at runtime) since rendered images can vary significantly according to various scene configurations such as geometries, materials, and illumination. As a result, it is not uncommon to see that a pre-trained denoiser fails to denoise a test image in practice (e.g., Fig. 1).

One can also consider correcting (potentially non-ideal) visual artifacts of denoised estimates as a post-process. Back et al. [2020] presented such a post-processing model that corrects a denoised image with another pre-trained neural network. While this post-processing model can remove visual artifacts from denoising, it still shares the same problem as supervised learning that we need to train this model with a dataset that contains denoised-reference image pairs.

To address this fundamental limitation of supervised learning, we introduce a combination of two different learning mechanisms, *self-supervised* correction for *supervised learning*, in the context of MC denoising. We propose a self-supervised post-correction network trained only with test images on the fly. The training data for the supervised denoiser and our self-supervised learning are mutually exclusive, and thus both techniques are complementary from an optimization perspective. Our main technical contributions are as follows.

- We propose a new self-supervised loss that enables us to optimize a post-correction framework using only a test image pair, i.e., a noisy image and its denoised output (Sec. 4.1).
- We present a practical implementation for this post-correction network where the network can be effectively optimized using our self-supervised loss (Sec. 4.2).

We demonstrate that our post-correction can improve state-of-the-art learning-based denoising techniques, especially when the denoisers receive complex test images with detailed high-frequency information, as shown in Fig. 1.

2 RELATED WORK

Pretraining-based optimization. Designing an effective denoising network has been actively explored since it allows for modeling a complex non-linear relationship from noisy input images to their ground truth. Kalantari et al. [2015] exploited a multilayer perceptron to adapt the parameters of a cross-bilateral filter per pixel. Chaitanya et al. [2017] used a recurrent neural network that reduces temporal noise in animation. Bako et al. [2017] employed a convolutional neural network that infers denoising weights per pixel, which was later extended for an animated sequence [Vogels et al. 2018]. Gharbi et al. [2019] devised a framework that takes radiance samples as input, and Xu et al. [2019] introduced a generative adversarial network for denoising. Kettunen et al. [2019] and Guo et al. [2019] presented specialized neural frameworks that take image gradients as well as primal pixel colors for gradient-domain renderings [Hua et al. 2019; Kettunen et al. 2015; Lehtinen et al. 2013]. Recently, Back et al. [2020] proposed a post-reconstruction network that boosts denoising results, and Yu et al. [2021] exploited the self-attention mechanism for improving denoising quality by selecting suitable neighboring pixels.

These methods use different neural architectures for denoising, but the common to all the techniques is that they pretrain the networks using a dataset that does not include test images available only for their inference. We present a post-correction framework that optimizes our neural network by taking their unseen data (i.e., test images) into account.

MSE-based optimization. A conventional approach for optimizing image denoising is to adjust denoising parameters per pixel. Errors can vary significantly across pixels, especially for MC rendered images with heterogeneous variances. Overbeck et al. [2009] transformed a rendered image via wavelets and truncated wavelet coefficients using the variance of pixel colors, and Sen and Darabi [2012] exploited mutual information to estimate optimal parameters for a cross-bilateral filter. Li et al. [2012] exploited Stein's unbiased risk estimator, which produces unbiased estimates for denoising errors, and optimized cross-bilateral and non-local means filters. Rousselle et al. [2012] proposed a dual-buffer approach for estimating denoising errors of non-local means filters, and this approach was exploited later for cross non-local means [Rousselle et al. 2013] and regression-based denoising [Bitterli et al. 2016]. Moon et al. [2014; 2016] estimated squared bias and variance of the denoising using local regression and adapted their denoising kernels across pixels. Zheng et al. [2021] formulated an estimate to blend multiple denoised images and predicted optimal per-pixel weights for the denoised images. A comprehensive overview of such classical optimization for image denoising is available [Zwicker et al. 2015].

A vital benefit of this conventional approach is that a denoising process can be specialized for each input image since their parameters are optimized using the test image to be rendered at runtime. It inspires us to design our post-correction technique that enhances denoised estimates via a deep neural network trained using test input. While our method exploits a similar optimization scheme (i.e., MSE-based optimization using test images), the main distinction is that we train a deep neural network, unlike the classical methods.

Self-supervised learning in other problems. Training a neural network using only test images has been actively studied for computer vision problems such as image restoration [Heckel and Hand 2019; Quan et al. 2020; Ulyanov et al. 2018], image decomposition [Gandelsman et al. 2019], and image fusion [Uezato et al. 2020]. In computer graphics, training a neural network that adjusts sampling density for importance sampling without pretraining was explored in [Müller et al. 2019; Zheng and Zwicker 2019]. We exploit self-supervised learning but apply this learning mechanism for MC denoising and propose a specialized neural network that can correct denoised estimates.

3 BACKGROUND AND MOTIVATION

We provide an overview of the existing post-correction technique [Back et al. 2020] that enhances denoised estimates using a pre-trained neural network. We then motivate our self-supervised learning that can perform such a correction without any pretraining.

Supervised post-correction for image denoising. The deep combiner (DC) [Back et al. 2020] combines independent and correlated pixel estimates $\mathbf{y}$ and $\mathbf{z}$ to produce an improved image $\hat{\boldsymbol{\mu}}$ that estimates the ground truth $\boldsymbol{\mu}$. While the DC is not limited to denoisers,

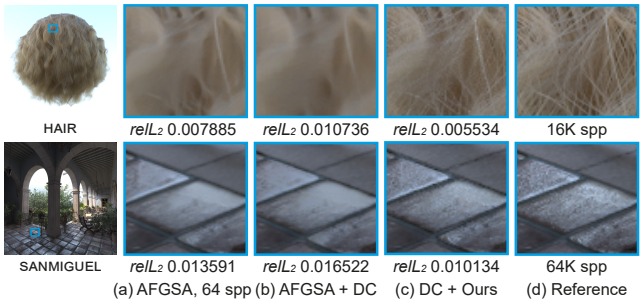

Figure 2: Our post-correction results for the supervised deep combiner (DC) with a denoiser, AFGSA. The DC has been trained without the denoising results of AFGSA, and it fails to boost the unseen denoiser. Our technique (c) takes the DC result (b) as input and restores the lost details.

one can use a noisy image as the independent input $\mathbf{y}$ to the DC, and the corresponding denoised image by a denoiser as the correlated input $\mathbf{z}$ to the DC, to remove artifacts due to denoising. They consider the following statistical models for the two inputs ($\mathbf{y}$ and $\mathbf{z}$);

$$\mathbf{y}_c = \boldsymbol{\mu}_c + \mathbf{e}_c, \tag{1}$$

$$\mathbf{z}_c - \mathbf{z}_i = \boldsymbol{\mu}_c - \boldsymbol{\mu}_i + \mathbf{e}_{ci}, \tag{2}$$

where $\mathbf{y}_c$, $\mathbf{z}_c$ and $\boldsymbol{\mu}_c$ are the c-th pixel colors in $\mathbf{y}$, $\mathbf{z}$ and $\boldsymbol{\mu}$, and $\mathbf{e}_c$ and $\mathbf{e}_{ci}$ are the error terms. The expectations of the error terms ($\mathbf{e}_c$ and $\mathbf{e}_{ci}$) assume to be zero vectors. Similarly, $\mathbf{z}_i$ and $\boldsymbol{\mu}_i$ are the i-th pixel colors. We shall treat the pixel colors as 3×1 vectors.

Given the models, a combination function $f_c(\mathbf{y}, \mathbf{z})$ at pixel c is defined as

$$f_c(\mathbf{y}, \mathbf{z}) = \frac{1}{\sum_{i \in \Omega_c} w_i} \left\{ \sum_{i \in \Omega_c} w_i \mathbf{y}_i + \sum_{i \in \Omega_c} w_i (\mathbf{z}_c - \mathbf{z}_i) \right\}, \tag{3}$$

which produces the pixel estimate $\hat{\boldsymbol{\mu}}_c = f_c(\mathbf{y}, \mathbf{z})$ as a weighted average of independent colors $\mathbf{y}_i$ and correlated color differences $(\mathbf{z}_c - \mathbf{z}_i)$ within a local window Ω_c centered at pixel c.

This post-correction process (Eq. 3) is controlled by the combination weights w_i ($w_i > 0$) that should be adjusted per pixel since the variances of $\mathbf{y}_i$ and $\mathbf{z}_c - \mathbf{z}_i$ can vary locally. To this end, the DC pretrains a neural network that produces optimal per-pixel weights that minimize a supervised loss:

$$\mathcal{L}(\hat{\boldsymbol{\mu}}_c) = \frac{||\hat{\boldsymbol{\mu}}_c - \boldsymbol{\mu}_c||^2}{\bar{\mu}_c^2 + 0.01}, \tag{4}$$

which uses a relative L_2 error [Rousselle et al. 2011] that penalizes the errors in bright areas by leveraging the intensity $\bar{\mu}_c$ (i.e., the average of the $\boldsymbol{\mu}_c$) of the ground truth color $\boldsymbol{\mu}_c$. This supervised loss relies on the ground truth $\boldsymbol{\mu}_c$ for pixel c, which is replaced by a reference value rendered with a large number of samples in practice.

This process is a form of *supervised learning*, which is possible only when one can access the reference values. Therefore, it can be conducted only for pretraining a neural network using a training dataset including the references. The DC is thus pretrained for a specific set of scenes, images, and denoisers (to produce $\mathbf{z}$ given $\mathbf{y}$) as a supervised learning model.

On the contrary, our *self-supervised* learning trains a neural network without relying on the $\boldsymbol{\mu}$. Therefore, our model can be trained *on the fly* for each input $\mathbf{y}$ and $\mathbf{z}$ at runtime without needing to know the corresponding $\boldsymbol{\mu}$. The key technical problem is that we need to design a self-supervised loss that effectively optimizes a neural network using only the imperfect input data (e.g., $\mathbf{y}$ and $\mathbf{z}$ for the post-correction scenario).

Problem statement. Pretraining a post-correction network by supervised learning can be ideal when a training dataset effectively covers all the possible runtime scenarios. Unfortunately, it is not practical to assume that we can prepare such an ideal training set since the test data (i.e., an image we want to render at runtime) is commonly considered unseen. Fig. 2 shows such failure cases where post-correction actually deteriorates the correlated input generated by an untrained denoiser. It motivates our self-supervised learning that is free from this fundamental issue of supervised learning since we train the network on the fly at runtime. This flexibility allows us to even take *the output of supervised post-correction* as our *input* to further improve their results, as shown in Fig. 2.

4 SELF-SUPERVISED POST-CORRECTION

We propose a new self-supervised framework (Fig. 3) that post-corrects the results of a supervised learning model using their input and output without any pretraining. To fulfill this objective, we derive a mean squared error (MSE) based self-supervised loss in Sec. 4.1 and present a post-correction neural network guided by the loss in Sec. 4.2.

4.1 Self-Supervised Loss

Our goal is to have a self-supervised model that guides a post-correction network at runtime. We cannot use the actual error $||\hat{\boldsymbol{\mu}}_c - \boldsymbol{\mu}_c||^2$ since it can be obtained only when we can access the ground truth $\boldsymbol{\mu}_c$. We instead estimate the *expectation* of the actual error $E||\hat{\boldsymbol{\mu}}_c - \boldsymbol{\mu}_c||^2$ using only the test input analogously in classical denoisers [Li et al. 2012; Rousselle et al. 2012]. We adopt the dual-buffer scheme [Bitterli et al. 2016] that splits MC pixel estimates into two sub-buffers and estimates denoising errors for a noisy buffer using another noisy buffer, which can be considered a two-fold cross-validation. Lehtinen et al. [2018] also used the dual-buffer scheme to optimize a neural network mainly for generic image denoising. Unlike Lehtinen et al., we propose to optimize a network on the fly. As a result, our runtime optimization can complement the pretraining-based denoising.

Specifically, we take dual-buffered pairs $(\mathbf{y}^a, \mathbf{y}^b)$ and $(\mathbf{z}^a, \mathbf{z}^b)$. Splitting noisy estimates $\mathbf{y}$ into two sub-buffers is trivial [Rousselle et al. 2012], and the denoised estimates $\mathbf{z}^a$ and $\mathbf{z}^b$ can be generated by applying a denoiser to $\mathbf{y}^a$ and $\mathbf{y}^b$, respectively. The resulting pair $(\mathbf{y}^a, \mathbf{z}^a)$ is independent of another one $(\mathbf{y}^b, \mathbf{z}^b)$.

Given those sub-buffered inputs, let us apply post correction to each image pair independently to obtain estimates $\hat{\boldsymbol{\mu}}^a$ and $\hat{\boldsymbol{\mu}}^b$. We define a supervised loss that optimizes this correction process for

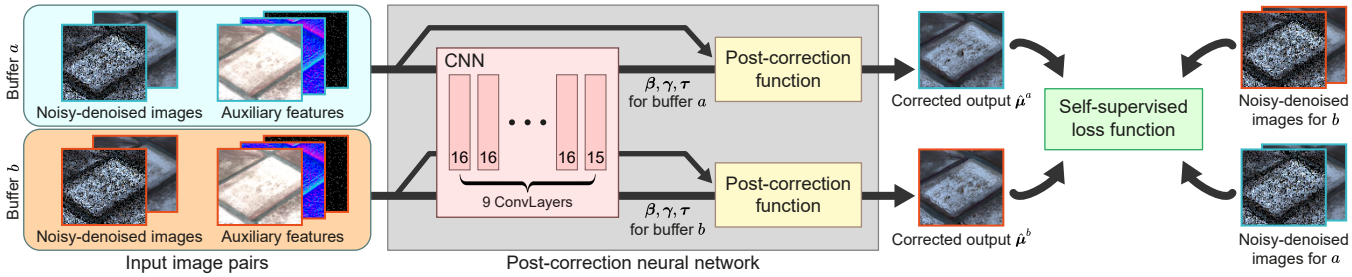

Figure 3: We split an image pair (noisy-denoised images) together with auxiliary features into dual buffers (a and b), and each buffer is fed into a convolutional neural network (CNN) in turn so that the parameters of our post-correction function (β, γ and τ in Eqs. 10 and 11) can be determined. We generate two output images ($\hat{\mu}^a$ and $\hat{\mu}^b$) from the dual-buffered inputs independently, and this correction process is optimized using our self-supervised loss (Eq. 9) at runtime.

the first pair as

$$\mathcal{L}(\hat{\boldsymbol{\mu}}_c^a) = \frac{E||\hat{\boldsymbol{\mu}}_c^a - \boldsymbol{\mu}_c||^2}{\bar{\mu}_c^2 + 0.01}. \tag{5}$$

To derive a self-supervised loss $\hat{\mathcal{L}}(\hat{\boldsymbol{\mu}}_c^a)$, we can estimate the unknown $E||\hat{\boldsymbol{\mu}}_c^a - \boldsymbol{\mu}_c||^2$ by an unbiased estimate:

$$E||\hat{\boldsymbol{\mu}}_c^a - \boldsymbol{\mu}_c||^2 \approx ||\hat{\boldsymbol{\mu}}_c^a - \mathbf{y}_c^b||^2 - \mathbf{1}^T\hat{\sigma}^2(\mathbf{y}_c^b), \tag{6}$$

where $\hat{\sigma}^2(\mathbf{y}_c^b)$ is an unbiased estimate of the variance $\sigma^2(\mathbf{y}_c^b)$, i.e., the sample variance of the pixel color $\mathbf{y}_c^b$, and $\mathbf{1}$ is a vector of ones of size 3×1. Please see the supplementary report for a detailed derivation. By plugging the unbiased estimate of the MSE (Eq. 6) into $\mathcal{L}(\hat{\boldsymbol{\mu}}_c^a)$ (Eq. 5), the loss for $\hat{\boldsymbol{\mu}}_c^a$ is approximated as

$$\mathcal{L}(\hat{\boldsymbol{\mu}}_c^a) \approx \frac{||\hat{\boldsymbol{\mu}}_c^a - \mathbf{y}_c^b||^2 - \mathbf{1}^T\hat{\sigma}^2(\mathbf{y}_c^b)}{\bar{\mu}_c^2 + 0.01}. \tag{7}$$

This approximate loss still depends on the reference value, i.e., the $\bar{\mu}_c$ in the denominator, and thus the unknown should be estimated. For the estimation, it is desirable to choose a value that is statistically independent of the output estimate $\hat{\boldsymbol{\mu}}_c^a$ since it allows us to ignore the (noisy) variance-related term $\mathbf{1}^T\hat{\sigma}^2(\mathbf{y}_c^b)/(\bar{\mu}_c^2 + 0.01)$. Note that this omission does not affect an optimization when its gradient with respect to the output estimate $\hat{\boldsymbol{\mu}}_c^a$ is zero.

In this respect, we have two intuitive candidates $\bar{y}_c^b$ and $\bar{z}_c^b$, i.e., the intensities of $\mathbf{y}_c^b$ and $\mathbf{z}_c^b$, for the unknown $\bar{\mu}_c$. In practice, the biased value $\bar{z}_c^b$ is often more accurate than the unbiased one $\bar{y}_c^b$ since a denoiser often reduces the error of their noisy input $\mathbf{y}_c^b$ drastically. Hence, we choose the $\bar{z}_c^b$ for the denominator (Eq. 7) and result in our self-supervised loss for $\hat{\boldsymbol{\mu}}_c^a$:

$$\hat{\mathcal{L}}(\hat{\boldsymbol{\mu}}_c^a) = \frac{||\hat{\boldsymbol{\mu}}_c^a - \mathbf{y}_c^b||^2}{(\bar{z}_c^b)^2 + 0.01}. \tag{8}$$

The loss $\hat{\mathcal{L}}(\hat{\boldsymbol{\mu}}_c^b)$ for the other buffer $\hat{\boldsymbol{\mu}}_c^b$ can be derived in the same manner. This self-supervised loss allows us to optimize the parameters θ of a neural network that produces post-corrected estimates $\hat{\boldsymbol{\mu}}^a$ and $\hat{\boldsymbol{\mu}}^b$, and it can be represented as

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{3N}\sum_{c=1}^{N} 0.5\left(\hat{\mathcal{L}}(\hat{\boldsymbol{\mu}}_c^a) + \hat{\mathcal{L}}(\hat{\boldsymbol{\mu}}_c^b)\right), \tag{9}$$

where N is the number of pixels for the input images.

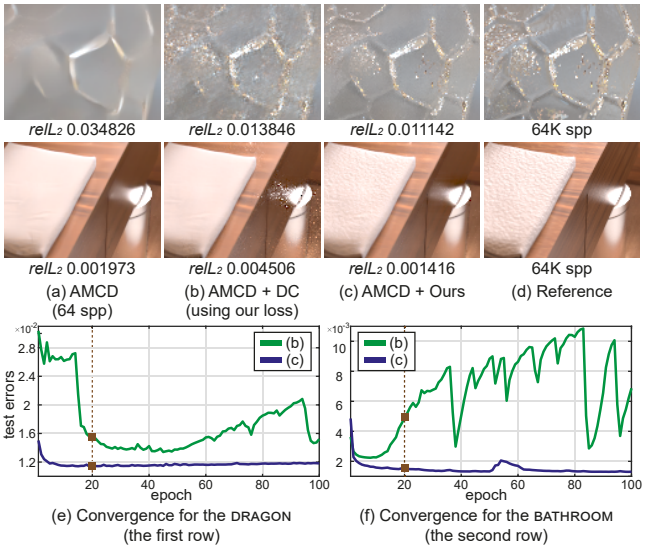

Figure 4: Comparisons between post-correction neural networks of DC (b) and our technique (c) for a recent denoiser, AMCD (a). Both networks are optimized using our self-supervised loss, and we generate their results ((b) and (c)) using 20 epochs. Our lightweight network, which has 79.5× smaller parameters than the DC, is robust against overfitting (see (e) and (f)), and it leads to more stable post-correction results.

4.2 A Practical Post-Correction Neural Network

We explain a practical implementation of the post-correction neural network. A straightforward option is to directly employ an existing model [Back et al. 2020] while replacing its supervised loss with our self-supervised loss. However, we found that this choice results in overfitting to the noise in the input $\mathbf{y}$ since the training data (only a test image pair) is insufficient for training the complex network with millions of trainable parameters (see Fig. 4). Moreover, training such a complex neural network at runtime can be expensive (e.g., 41.9 seconds for the example results in Fig. 4).

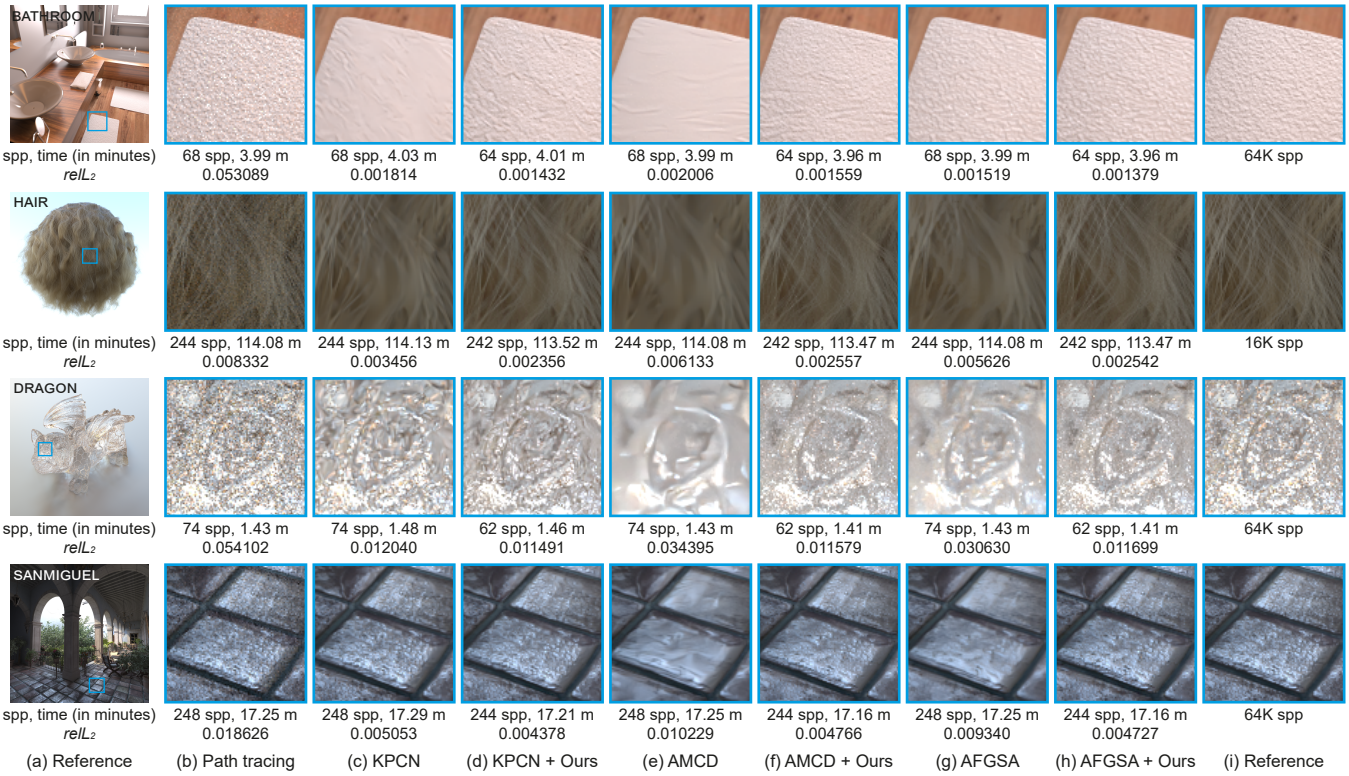

Figure 5: Equal-time comparisons of the denoising methods with and without our post-correction. Our self-supervised optimization helps the supervised-learning methods improve denoising quality when they fail to preserve fine details.

We propose to use a lightweight framework (Fig. 3) that relies on the same convolutional neural network of the previous method [Back et al. 2020] but with much smaller parameters. Our network size is only 0.02M which is 79.5× smaller than the original one with 1.59M [Back et al. 2020]. As an additional optimization to the baseline, we reformulate the existing localized combination (Eq. 3) into

$$g_c(\mathbf{y},\mathbf{z}) = \frac{\sum_{i\in\Omega_c} w_i \left\{ \mathbf{y}_i + \boldsymbol{\beta}_c^z \circ (\mathbf{z}_c - \mathbf{z}_i) + \boldsymbol{\beta}_c^\rho \circ (\boldsymbol{\rho}_c - \boldsymbol{\rho}_i) + \boldsymbol{\beta}_c^n \circ (\mathbf{n}_c - \mathbf{n}_i) \right\}}{\sum_{i\in\Omega_c} w_i} \tag{10}$$

where the symbol $\circ$ is the element-wise product. Note that we replace $\mathbf{z}_c - \mathbf{z}_i$ in the original combination (Eq. 3) with $\boldsymbol{\beta}_c^z \circ (\mathbf{z}_c - \mathbf{z}_i) + \boldsymbol{\beta}_c^\rho \circ (\boldsymbol{\rho}_c - \boldsymbol{\rho}_i) + \boldsymbol{\beta}_c^n \circ (\mathbf{n}_c - \mathbf{n}_i)$ using albedo and normal buffers ($\boldsymbol{\rho}$ and $\mathbf{n}$, respectively). Specifically, $\boldsymbol{\rho}_c$ and $\mathbf{n}_c$ are the albedo and normal values of size 3×1 at pixel c and $\boldsymbol{\beta}_c^z$, $\boldsymbol{\beta}_c^\rho$, and $\boldsymbol{\beta}_c^n$ of size 3×1 are the scale parameters that control a relative importance for $(\mathbf{z}_c - \mathbf{z}_i)$, $(\boldsymbol{\rho}_c - \boldsymbol{\rho}_i)$ and $(\mathbf{n}_c - \mathbf{n}_i)$.

We set $\boldsymbol{\beta}_c^z$ to one vector and the others ($\boldsymbol{\beta}_c^\rho$ and $\boldsymbol{\beta}_c^n$) to zero vectors, this formulation is equivalent to the previous combination kernel (Eq. 3), derived by assuming that $E[\mathbf{e}_{ci}]$ is zero vector. In practice, the assumption can be invalid since the $\mathbf{z}_c$ and $\mathbf{z}_i$ are biased pixel estimates generated by a denoiser. We also exploit the rendering-specific information (e.g., albedo and normal values) for compensating the approximation error, and this bias compensation is controlled by the network through the parameters ($\boldsymbol{\beta}_c^z$, $\boldsymbol{\beta}_c^\rho$, and $\boldsymbol{\beta}_c^n$).

We can define the weight w_i (in Eq. 10) in a cross-bilateral form:

$$w_i = \begin{cases} \exp\left(-\dfrac{\log_e(1 + ||\mathbf{y}_c - \mathbf{y}_i||^2)}{(\gamma_c^y)^2 + \epsilon} - \dfrac{\log_e(1 + ||\mathbf{z}_c - \mathbf{z}_i||^2)}{(\gamma_c^z)^2 + \epsilon} \right) \\ \times \exp\left(-\dfrac{||\boldsymbol{\rho}_c - \boldsymbol{\rho}_i||^2}{(\gamma_c^\rho)^2 + \epsilon} - \dfrac{||\mathbf{n}_c - \mathbf{n}_i||^2}{(\gamma_c^n)^2 + \epsilon} - \dfrac{(v_c - v_i)^2}{(\gamma_c^v)^2 + \epsilon} \right) & \text{if } i \neq c, \\ \tau_c \quad \text{otherwise.} \end{cases} \tag{11}$$

γ_c^y, γ_c^z, γ_c^ρ, γ_c^n and γ_c^v are the bandwidth parameters for the image pair ($\mathbf{y}$ and $\mathbf{z}$) and auxiliary features including albedo $\boldsymbol{\rho}$, normal $\mathbf{n}$ and visibility buffers $\mathbf{v}$, respectively. We found that taking a logarithm for the squared color differences is needed for a stable learning since the colors have a high dynamic range unlike the other features. Note that we treat the weight $w_c = \tau_c$ for the center pixel c separately in Eq. 11 to avoid that the center weight becomes always one by the cross-bilateral weighting like as [Işık et al. 2021]. The ϵ is set to 0.0001 to avoid the division by zero.

As a result, the post-correction function $g_c(\mathbf{y},\mathbf{z})$ (Eq. 10) requires a set of the scale parameters ($\boldsymbol{\beta}_c \equiv \{\boldsymbol{\beta}_c^z, \boldsymbol{\beta}_c^\rho, \boldsymbol{\beta}_c^n\}$) and bandwidth parameters ($\boldsymbol{\gamma}_c \equiv \{\gamma_c^y, \gamma_c^z, \gamma_c^\rho, \gamma_c^n, \gamma_c^v\}$) and center weight τ_c per pixel c. The per-pixel parameters are generated by a neural network (Fig. 3) with trainable parameters θ, analogously as in some prior work [Bako et al. 2017; Kalantari et al. 2015]. We train the network parameters using our self-supervised learning (Eq. 9) using the dual-buffered input pairs ($(\mathbf{y}^a, \mathbf{z}^a)$ and $(\mathbf{y}^b, \mathbf{z}^b)$ in Sec. 4.1). As we employ the auxiliary buffers ($\boldsymbol{\rho}$, $\mathbf{n}$, and $\mathbf{v}$), we also split those buffers and feed those into the network. Once the optimal network

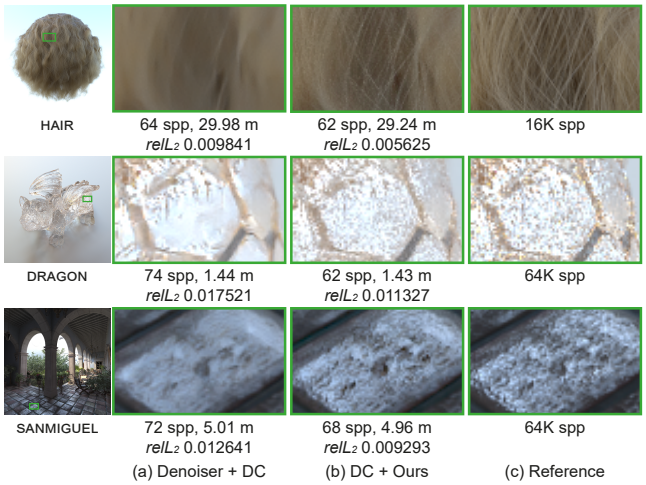

Figure 6: Same-time comparisons between DC for denoiser (i.e., AFGSA) and our technique for the DC. Our runtime optimization enables us to improve the supervised technique (DC) by correcting their outputs through our post-correction lightweight neural network trained by self-supervised learning.

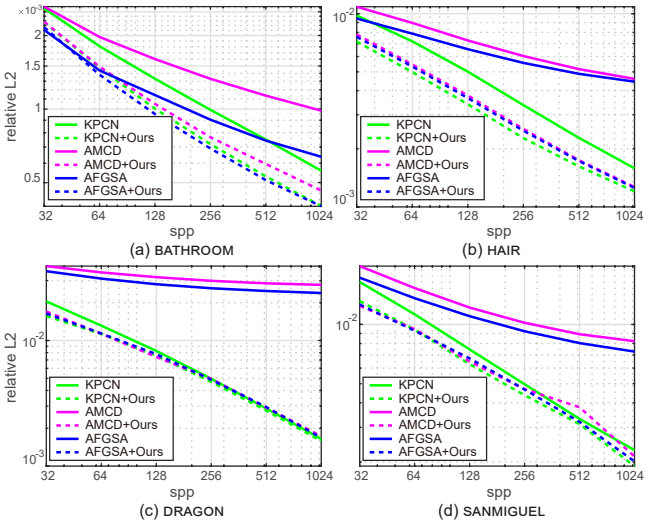

Figure 7: Numerical convergence plots (in log-log scale) for denoising techniques with and without our post-correction.

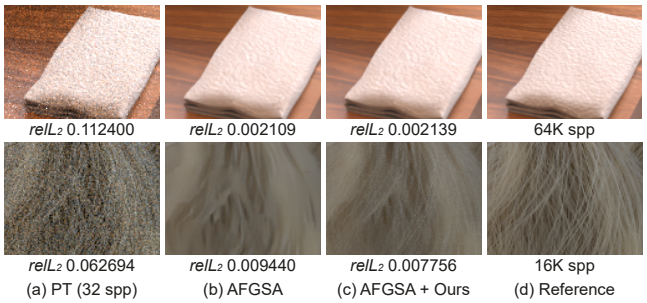

Figure 8: Failure cases occur when the denoised estimates (b) are post-corrected using our self-supervised loss that relies on the noisy input (a).

parameters θ^* are estimated, we infer the two output estimates by the function $g_c(\mathbf{y}=\mathbf{y}^a, \mathbf{z}=\mathbf{z}^a)$ and $g_c(\mathbf{y}=\mathbf{y}^b, \mathbf{z}=\mathbf{z}^b)$ using the dual buffers. Our final estimates are computed as the average of the two estimates. Fig. 4 shows that our optimization makes the self-supervised learning robust against overfitting and results in more accurate post-correction results than the baseline (DC) driven by the self-supervised loss.

Network and training details. Our network (Fig. 3) consists of 9 layers, each of which uses convolutional filters with 3×3 kernel size. The last layer uses 15 filters (i.e., the number of the parameters for $\boldsymbol{\beta}_c$, $\boldsymbol{\gamma}_c$ and τ_c), and the other layers use 16 filters. For the activation function in the last layer, we use the tanh function for $\boldsymbol{\beta}_c$ and the softplus for $\boldsymbol{\gamma}_c$ and τ_c. We use the ReLU for the other layers. We have implemented our self-supervised framework using Tensorflow [Abadi et al. 2015]. We have extracted 128×128 patches from the input color and auxiliary images for our runtime training and trained the network for 20 epochs using the Adam optimizer [Kingma and Ba 2014]. We have set the learning rate to $0.01\sqrt{\frac{1}{3N}\sum_{c=1}^{N}\mathbf{1}^T\hat{\sigma}^2(\mathbf{y}_c)}$ where $\hat{\sigma}^2(\mathbf{y}_c)$ is the estimated variance using dual-buffers (i.e., $\hat{\sigma}^2(\mathbf{y}_c)=(\mathbf{y}_c^a-\mathbf{y}_c^b)\circ(\mathbf{y}_c^a-\mathbf{y}_c^b)/4$). The batch size has been set to 16, and we have used Xavier uniform initializer [Glorot and Bengio 2010]. We have set the window size $|\Omega_c|$ to 19×19.

5 RESULTS AND DISCUSSION

We applied our self-supervised post-correction to the state-of-the-art denoising methods, KPCN [Bako et al. 2017], AMCD [Xu et al. 2019], and AFGSA [Yu et al. 2021]. We tested the pre-trained models released by the authors to generate their denoised estimates. We compared our technique also with the supervised deep combiner (DC) [Back et al. 2020]. We have fully retrained the DC using 800 training images generated using eight public scenes [Bitterli 2016] for a fairer comparison. We used the PBRT renderer [Pharr et al. 2016] to generate test images, and all the tests were conducted using an Intel Xeon CPU E5-2687W and NVIDIA GeForce RTX 3090 GPU. We used the four test scenes (BATHROOM, HAIR, DRAGON, and SANMIGUEL).

Equal-time comparisons. Fig. 5 compares denoised estimates and their post-corrected results using our technique. The reported times for our method include the self-supervised learning times as we train a neural network on the fly. As shown in the figure, the denoisers do not always show good denoising results for all the test scenes. For example, KPCN produces relatively high-quality results with preserved details for the DRAGON and SANMIGUEL but over-blurs some details for the BATHROOM and HAIR. AFGSA preserves the high-frequency information for the BATHROOM but produces over-smoothed artifacts for the other scenes, similarly to AMCD. It indicates that a learning-based denoiser, even with extensive pre-training, would not be ideal for all the possible scenarios since

the test images at runtime can be arbitrarily different from the training images in practice. Our technique uses unseen data (i.e., the test images) for learning a post-correction network on the fly and improves the denoising results when supervised techniques fail (e.g., over-blurred artifacts).

As shown in Fig. 6, the supervised post-correction (DC) does not effectively restore lost high-frequency details. Because of the flexibility of self-supervised learning, our framework can take the post-corrected estimates from the DC as input to our method and *further* corrects the results. It demonstrates that our post-correction using self-supervised learning can complement different kinds of supervised techniques.

Numerical convergence. Fig. 7 shows numerical convergences of the tested methods. Our method helps the denoising techniques to produce more numerically accurate results, primarily when these methods fail. For example, AMCD and AFGSA do not effectively reduce their errors by increasing the sample size. The figure shows that our self-supervised gain becomes more significant for the two methods with large sample counts. Technically, the self-supervised loss can become more accurate with a larger sample size since it relies on noisy unbiased estimates (e.g., $\mathbf{y}_c^b$ in Eq. 7).

Ablation studies for our combination function. We reformulate the previous combination (Eq. 3) into a new localized function (Eq. 10) that explicitly exploits rendering-specific features (e.g., albedo and normal values) for bias compensation. We infer the combination weights w_i using a cross-bilateral function (Eq. 11). Table 1 shows the post-correction results of our self-supervised network with and without the two adaptations (Eqs. 10 and 11). As shown in Table 1, the cross-bilateral weighting (setting B and D in the table) allows the self-supervised correction to be more accurate than those without the weighting (setting A and C). Thanks to the additional use of auxiliary features for bias compensation, our combination (Eq. 10) with the cross-bilateral weighting (setting D) outperforms the previous combination (Eq. 3) with the same weighting scheme (setting B) except for a case (AFGSA for the DRAGON). We include the post-corrected images of the different configurations in the supplemental report.

Runtime overhead. Table 2 shows the breakdowns for the runtime overhead, excluding the sampling times for generating the noisy path-traced estimates. The denoising time for generating the dual-buffered input depends on a chosen denoising technique. On the other hand, the training and inference times (12.52 secs in total) are only affected by the resolutions of the test images. This computational overhead is non-negligible for a few sample counts. However, since our training time does not depend on the sample size, the overhead becomes minor and minor as the sample count increases. As a result, our post-correction can be effective for offline rendering scenarios where the sample count is moderate to large, as shown in the equal-time comparisons (Fig. 5).

Limitations and future work. One limitation of our post-correction framework is that the self-supervised loss is only an estimate of the unknown supervised loss, and thus it contains its own noise (mainly due to the use of unbiased but noisy estimates, e.g., $\mathbf{y}_c^b$ in Eq. 7). The noisy self-supervised loss guides our post-correction to improve existing supervised methods for most cases (see Fig. 7), but a failure case can occur when we correct denoised estimates with much-higher quality than noisy estimates. As shown in Fig. 8, our method fails to improve the denoised estimates of AFGSA for the BATHROOM, e.g., 1.4% worse than the input. We improve the numerical accuracy of the denoised estimates for the HAIR scene, but our result leaves some residual noise.

For animated sequences, one might consider reusing the neural network trained by the previous frames as a starting point

Table 1: Ablation studies for our self-supervised post-correction with 128 spp. We show the $relL_2$ errors of the original combination (Eq. 3) and our revised combination (Eq. 10) with and without the cross-bilateral weighting (Eq. 11). Brown and cyan colors highlight the best and second-best results.

Scenes	Denoisers	Setting A	Setting B	Setting C	Setting D
BATHROOM	KPCN	0.002903	0.001015	0.002200	**0.000991**
	AFGSA	0.002699	0.001127	0.001842	**0.000942**
HAIR	KPCN	0.004383	0.003899	0.004412	**0.003429**
	AFGSA	0.004185	0.004005	0.004379	**0.003713**
DRAGON	KPCN	0.008340	0.008314	0.008217	**0.007471**
	AFGSA	0.008223	**0.007715**	0.007863	0.007731
SANMIGUEL	KPCN	0.008211	0.006973	0.008577	**0.006228**
	AFGSA	0.007512	0.006848	0.010412	**0.006547**

∗ Four settings with different design choices
A: original combination (Eq. 3)
B: original combination (Eq. 3) w/ cross-bilateral weighting (Eq. 11)
C: refined combination (Eq. 10)
D: refined combination (Eq. 10) w/ cross-bilateral weighting (Eq. 11)

Table 2: Runtime breakdowns (in secs) for a test image of size 1K×1K.

Step	KPCN + Ours	AMCD + Ours	AFGSA + Ours
Buffer splitting	3.02	0.04	0.02
Post-correction	12.52	12.52	12.52
Total	15.54	12.56	12.54

Table 3: Our results for the SANMIGUEL scene (in Fig. 5), where we change the viewing direction over five frames. We use 64 spp for all the frames. We train our neural network for the current frame using the trained network from the previous frame (with reuse), and it produces more accurate results than using the random initialization per frame (without reuse).

Methods	1st frame	2nd frame	3rd frame	4th frame	5th frame
KPCN	0.011741	0.011547	0.011513	0.011533	0.011410
Without reuse	0.009758	0.009623	0.009699	0.009815	0.009468
With reuse	-	0.008868	0.008671	0.009008	0.008431
AFGSA	0.014113	0.014044	0.014011	0.014100	0.014073
Without reuse	0.010020	0.009790	0.009292	0.010004	0.009275
With reuse	-	0.008593	0.008332	0.008113	0.008094

for the current frame without initializing the network parameters randomly. Table 3 shows that this simple change improves our post-correction accuracy (e.g., 8.5% to 23.3% improvement over the random initialization). Nevertheless, it would be desirable to incorporate temporal coherency in our framework to suppress flickering artifacts. We leave this investigation as future work. It would also be interesting to develop an extended self-supervised loss for gradient-domain rendering [Hua et al. 2019; Lehtinen et al. 2013] where estimated image gradients are available as additional unbiased input.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments and suggestions, and we also appreciate the following authors and artists for each scene: nacimus (BATHROOM), Cem Yuksel (HAIR), Christian Schüller (DRAGON) and Guillermo M. Leal Llaguno (SANMIGUEL). Bochang Moon is the corresponding author of the paper. This work was supported by the National Research Foundation of Korea (NRF) funded by the Korea government (MSIT) (No. 2020R1A2C4002425) and Ministry of Culture, Sports and Tourism and Korea Creative Content Agency (No. R2021080001).

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, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. 2020. Deep Combiner for Independent and Correlated Pixel Estimates. *ACM Trans. Graph.* 39, 6, Article 242 (2020), 12 pages.

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.

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.

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.

Yosef Gandelsman, Assaf Shocher, and Michal Irani. 2019. "Double-DIP": Unsupervised Image Decomposition via Coupled Deep-Image-Priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 9)*. PMLR, 249–256.

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.

Reinhard Heckel and Paul Hand. 2019. Deep Decoder: Concise Image Representations from Untrained Non-convolutional Networks. *International Conference on Learning Representations* (2019).

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.

Mustafa Işık, Krishna Mullia, Matthew Fisher, Jonathan Eisenmann, and Michaël Gharbi. 2021. Interactive Monte Carlo Denoising Using Affinity of Neural Features. *ACM Trans. Graph.* 40, 4, Article 37 (2021), 13 pages.

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.

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.

Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. 2018. Noise2Noise: Learning Image Restoration without Clean Data. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 2965–2974.

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.

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.

Thomas Müller, Brian Mcwilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. *ACM Trans. Graph.* 38, 5, Article 145 (2019), 19 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, Wenzel Jakob, and Greg Humphreys. 2016. *Physically based rendering: From theory to implementation.* Morgan Kaufmann.

Yuhui Quan, Mingqin Chen, Tongyao Pang, and Hui Ji. 2020. Self2Self With Dropout: Learning Self-Supervised Denoising From Single Image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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.

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.

Tatsumi Uezato, Danfeng Hong, Naoto Yokoya, and Wei He. 2020. Guided Deep Decoder: Unsupervised Image Pair Fusion. In *Computer Vision – ECCV 2020*. 87–102.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2018. Deep Image Prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Thijs Vogels, Fabrice Rousselle, Brian Mcwilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Trans. Graph.* 37, 4, Article 124 (2018), 15 pages.

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.

Jiaqi Yu, Yongwei Nie, Chengjiang Long, Wenju Xu, Qing Zhang, and Guiqing Li. 2021. Monte Carlo Denoising via Auxiliary Feature Guided Self-Attention. *ACM Trans. Graph.* 40, 6, Article 273 (2021), 13 pages.

Quan Zheng and Matthias Zwicker. 2019. Learning to Importance Sample in Primary Sample Space. *Computer Graphics Forum* 38, 2 (2019), 169–179.

Shaokun Zheng, Fengshi Zheng, Kun Xu, and Ling-Qi Yan. 2021. Ensemble Denoising for Monte Carlo Renderings. *ACM Trans. Graph.* 40, 6, Article 274 (2021), 17 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.