Neural James-Stein Combiner for Unbiased and Biased Renderings

JEONGMIN GU, Gwangju Institute of Science and Technology, South Korea
JOSE A. IGLESIAS-GUITIAN, Universidade da Coruña - CITIC, Spain
BOCHANG MOON, Gwangju Institute of Science and Technology, South Korea

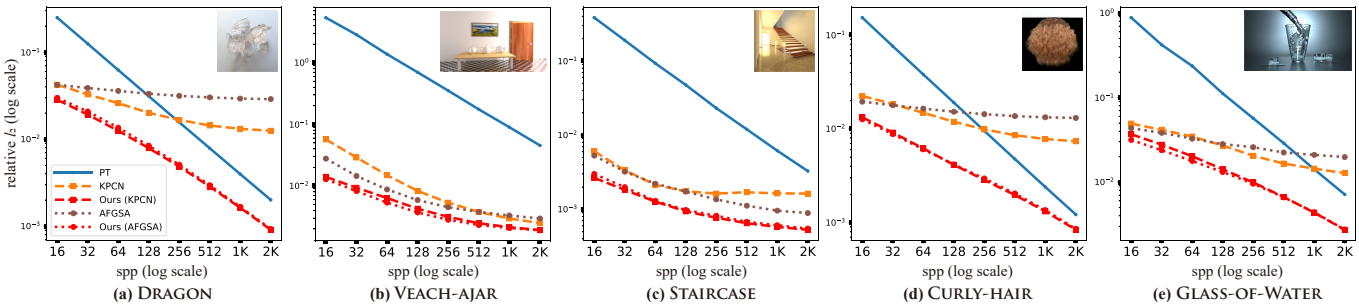

Fig. 1. Numerical convergence of the learning-based denoisers, KPCN [Bako et al. 2017] and AFGSA [Yu et al. 2021], with and without our technique. We report the relative l_2 errors [Rousselle et al. 2011] of the tested techniques from 16 to 2K samples per pixel (spp). The state-of-the-art denoisers show much lower errors than their input, i.e., path tracing (PT), for small sample counts (e.g., 16), but their improvements become minor (c) or disappear ((a), (d), and (e)) for larger sample counts due to their slow convergence rates, except for the VEACH-AJAR that contains fireflies. Our technique helps the denoisers have lower errors than their unbiased inputs, and this dominance property leads to significantly improved convergence rates of the input denoisers.

Unbiased rendering algorithms such as path tracing produce accurate images given a huge number of samples, but in practice, the techniques often leave visually distracting artifacts (i.e., noise) in their rendered images due to a limited time budget. A favored approach for mitigating the noise problem is applying learning-based denoisers to unbiased but noisy rendered images and suppressing the noise while preserving image details. However, such denoising techniques typically introduce a systematic error, i.e., the denoising bias, which does not decline as rapidly when increasing the sample size, unlike the other type of error, i.e., variance. It can technically lead to slow numerical convergence of the denoising techniques. We propose a new combination framework built upon the James-Stein (JS) estimator, which merges a pair of unbiased and biased rendering images, e.g., a path-traced image and its denoised result. Unlike existing post-correction techniques for image denoising, our framework helps an input denoiser have lower errors than its unbiased input without relying on accurate estimation of per-pixel denoising errors. We demonstrate that our framework based on the well-established JS theories allows us to improve the error reduction rates of state-of-the-art learning-based denoisers more robustly than recent post-denoisers.

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

Additional Key Words and Phrases: James-Stein estimator, James-Stein combiner, Monte Carlo rendering, learning-based denoising

ACM Reference Format:
Jeongmin Gu, Jose A. Iglesias-Guitian, and Bochang Moon. 2022. Neural James-Stein Combiner for Unbiased and Biased Renderings. *ACM Trans. Graph.* 41, 6, Article 262 (December 2022), 14 pages. https://doi.org/10.1145/3550454.3555496

Authors' addresses: Jeongmin Gu, Gwangju Institute of Science and Technology, Gwangju, South Korea, jeong755@gm.gist.ac.kr; Jose A. Iglesias-Guitian, Universidade da Coruña - CITIC, A Coruña, Spain, j.iglesias.guitian@udc.es; Bochang Moon, Gwangju Institute of Science and Technology, Gwangju, South Korea, moonbochang@gmail.com.

1 INTRODUCTION

Unbiased Monte Carlo (MC) ray tracing, such as path tracing [Kajiya 1986] and bidirectional path tracing [Lafortune and Willems 1993], has been widely employed for offline rendering applications (e.g., production rendering [Pharr 2018]) where complex light phenomena need to be accurately simulated. While the unbiased methods are guaranteed to produce a rendering output without systematic errors (i.e., a bias), they often leave visually distracting noise due to the variance of the MC integration. As a result, it usually takes a long time to achieve a nearly converged image.

A well-known approach for alleviating MC rendering noise is to exploit image denoising that trades the variances of unbiased estimates with systematic errors (i.e., a denoising bias). A commonly adopted optimization for such image denoising is to exploit hand-crafted filters (e.g., [Bitterli et al. 2016; Sen and Darabi 2012]) and locally optimize their smoothing parameters (i.e., bandwidths). While this classical approach makes denoising errors zero with an infinite number of samples (i.e., consistency), their denoising performance is restricted to a chosen filter.

A recent alternative to the model-based methods is to employ a deep neural network that maps a noisy image to the ground truth without relying on a hard-coded filter. For example, learning-based denoisers [Bako et al. 2017; Yu et al. 2021] demonstrated that this flexibility could drastically improve the visual quality of unbiased images by suppressing random artifacts. However, such model-free techniques often struggle to maintain a fast convergence rate and can produce denoised images with even higher errors than their unbiased inputs (e.g., Fig. 1).

A compelling approach for remedying the slow convergence of learning-based denoisers corrects their denoising results guided by the mean-squared errors (MSEs) of the biased process. For example, one can blend the denoised estimates with unbiased input [Firmino et al. 2022] or other biased but consistent estimates [Zheng et al. 2021] per pixel. This error analysis-based approach can produce more accurate outputs than any of their inputs when the estimated per-pixel MSEs are correct, but a robust estimation of the per-pixel denoising errors can be challenging in practice. While a general post-correction method without the error analysis was recently explored in the deep combiner [Back et al. 2020], its combination model was not designed to improve the slow convergence of learning-based denoisers.

We present a post-correction framework, like the existing MSE-based correction method [Firmino et al. 2022; Zheng et al. 2021], that improves the convergence rates of learning-based denoisers. We, however, take a different route for improving the biased estimates. Our framework helps an input denoiser dominate its unbiased input (i.e., making lower errors than the unbiased estimates) irrespective of the errors of the biased input (e.g., even for severely biased images). This dominance property makes an input denoiser improve its convergence rate since the MSEs of the unbiased input bound the errors of the post-corrected denoised estimates. Our main contributions are as follows.

- We introduce the James-Stein (JS) estimator to the rendering field, and our method is the first to use this estimator for improving rendering efficiency to the best of our knowledge.
- We propose a localized combination framework built upon the JS estimator, which locally fuses a pair of unbiased and biased rendering estimates while forcing the combined output to dominate the unbiased input.

We demonstrate that assisting learning-based denoisers [Bako et al. 2017; Yu et al. 2021] to dominate their unbiased inputs can boost their denoising performance, as illustrated in Fig. 1. We also show that our post-correction can be more robust than existing MSE-based post-denoisers, thanks to a fundamental difference, i.e., the requirement of accurate estimation of per-pixel denoising errors.

2 RELATED WORK

This section discusses the related work of image denoising (i.e., our main application) that produces biased pixel estimates from their unbiased input estimates. The proposed method in this paper does not require modifying its input rendering techniques, and thus their unbiased input and biased output can be directly fed into our combination framework.

Image denoising has been widely adopted for unbiased MC renderings (e.g., path tracing), and it trades the variance of the noisy input with a systematic error (i.e., bias). Classical denoisers typically rely on predefined functional forms. Examples are wavelets [Overbeck et al. 2009], non-local means filters [Rousselle et al. 2012, 2013], joint bilateral filters [Li et al. 2012; Sen and Darabi 2012], and low-order polynomials [Bitterli et al. 2016; Moon et al. 2014, 2016]. A survey [Zwicker et al. 2015] includes an extensive overview of classical denoising techniques.

While the classical denoisers employ specific filters, the typical behavior of these approaches is to strike the optimal balance of denoising bias and variance by adjusting their denoising parameters per pixel. Nevertheless, it is often technically challenging to robustly estimate the MSEs of a denoiser (and thus optimal parameters) only from the noisy statistics (e.g., the sample mean and variance). Kalantari et al. [2015] alleviated the difficulty in parameter selection through a multi-layer perceptron that infers the bandwidths of classical filters such as joint bilateral filters. This adaptive optimization can allow simple denoising filters to be consistent (i.e., correct results in the limit), but their denoising performance can be restricted due to the hard-coded nature of the designed filters.

Training a deep neural network that learns a mapping from an unbiased input to the ground truth image has received substantial attention [Huo and Yoon 2021] since it relaxes the requirement of hand-crafted modeling for denoising filters. For example, Bako et al. [2017] presented a kernel-predicting convolutional network (KPCN) that infers the per-pixel weights of a general denoising kernel, and Vogels et al. [2018] reformulated the KPCN for animated sequences. Also, an advanced neural framework, the generative adversarial network [Goodfellow et al. 2014], was exploited to reduce MC rendering noise more effectively [Xu et al. 2019; Yu et al. 2021]. Moreover, it has been demonstrated that it can be beneficial to train a neural network that takes individual samples as input since the samples can have richer information than the sample means (i.e., pixel colors) [Gharbi et al. 2019; Munkberg and Hasselgren 2020].

Recent learning-based denoising has demonstrated great success, but their flexibility (i.e., without the dependency on hand-crafted filters) came at the expense of introducing a slow numerical convergence. Our framework allows state-of-the-art denoisers to improve their convergences and become consistent by assisting the denoised estimates to dominate their unbiased input.

Post-correction for image denoisers. Correcting denoised images in a post-denoising stage is appealing as denoisers typically leave residual errors (under- or over-blurred visual artifacts) in their output images. Back et al. [2020] proposed a general combination model which locally fuses independent and correlated pixel colors and demonstrated that this model could boost a denoised image by blending it with a path-traced image. Nonetheless, this post-correction was not designed to improve the convergence of the input denoisers, and thus the post-corrected denoisers can still have a slow convergence. Recent MSE-based post-denoisers [Firmino et al. 2022; Zheng et al. 2021] addressed the convergence problem of learning-based denoisers. Zheng et al. [2021] blended multiple denoising estimates per pixel to obtain an improved estimate, even mixing a classical but consistent denoiser with a learning-based one. Firmino et al. [2022] estimated the MSEs of a learning-based denoiser using Stein's unbiased risk estimator (SURE) [Stein 1981] and blended the biased estimates with their unbiased inputs per pixel. These techniques can be theoretically ideal since combined estimates can have lower errors than any of their inputs when the error analysis is correct. Nonetheless, it can be technically challenging to estimate per-pixel denoising errors accurately, and thus their per-pixel combinations can be sensitive to the estimation errors.

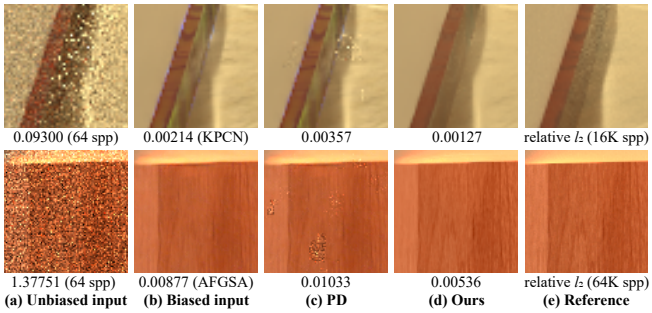

Fig. 2. Example results of an MSE-based post denoiser (PD [Firmino et al. 2022]) and ours, which combine the unbiased (a) and biased inputs (b). We use path tracing and learning-based denoisers (KPCN and AFGSA) to generate unbiased and biased images. PD does not improve the biased inputs while leaving some random artifacts propagated from the unbiased ones. On the other hand, our method robustly reduces the systematic errors in the denoised estimates. The zoomed areas come from the scenes, Staircase (top) and Veach-Ajar (bottom).

Our technique also aims to improve learning-based denoisers, but the key difference is that we enhance the convergence rates of the denoisers by bounding their errors through the James-Stein estimator that allows us to robustly combine a pair of unbiased and biased estimates without the knowledge of the per-pixel errors of the biased input.

The James-Stein estimator in other domains. The James-Stein estimator [Stein and James 1961] has been employed in various research fields. For example, Manton et al. [1998] presented a robust Kalman filter using the estimator for signal processing, and Hausser and Strimmer [2009] also proposed a James-Stein type estimator for inferring entropy and mutual information in the machine learning field. Wu et al. [2013] refined a classical non-local means filter using the JS estimator for general image denoising. As a recent example, the JS estimator was exploited for a learning-based classifier [Chakraborty et al. 2020]. In this paper, we introduce the usage of the JS estimator in the rendering domain for the first time.

3 BACKGROUND: THE JAMES-STEIN ESTIMATOR

This section introduces the underlying statistical foundations behind the James-Stein (JS) estimator [Stein and James 1961] and motivates our framework to combine unbiased and biased rendering estimates.

Suppose that we have a vector of unbiased estimates X of size p ($p \geq 3$) that follows a p-variate normal distribution, $X \sim N(\Theta, \sigma^2 I)$, where Θ is the ground truth to be estimated. The unbiased estimates are assumed to be independent and distributed with variance σ^2, and thus the covariance of size $p \times p$ becomes the diagonal matrix $\sigma^2 I$ with the identity matrix I of size p.

Given the statistical assumption, it is well-known that the sample mean is the maximum likelihood (ML) estimator for the unknown Θ. For example, in path tracing, the unbiased pixel estimates are generated by averaging radiance samples per pixel. The JS estimator takes the unbiased estimates as input and produces improved estimates:

$$\delta(X, Y) = Y + \left(1 - \frac{(p-2)\sigma^2}{\|X - Y\|^2}\right)(X - Y), \tag{1}$$

where Y is a fixed but arbitrary vector (e.g., biased estimates). Intuitively, this estimator shrinks X toward Y and the amount of shrinkage is controlled by a shrinkage factor $\eta = 1 - (p-2)\sigma^2/\|X - Y\|^2$.

The JS estimator produces biased estimates of the unknown Θ, but it was proved that this biased estimator dominates the unbiased ML estimator for any Θ when $p \geq 3$ [Stein and James 1961]. This property is known as Stein's paradox in statistics, as the dominance property was achieved using unrelated (independent) input [Efron and Morris 1977].

A technical caveat of the original estimator is that the shrinkage factor η can be negative and in this case its output estimates move away from the Y. It can be easily prevented by forcing the η to always have a positive value:

$$\delta^+(X, Y) = Y + \left(1 - \frac{(p-2)\sigma^2}{\|X - Y\|^2}\right)^+ (X - Y), \tag{2}$$

where $(\cdot)^+ = max(0, \cdot)$. This improved estimator is called the positive part JS estimator [Baranchik 1964, 1970] and it dominates the original JS estimator $\delta(X, Y)$.

Our motivation. While the original paper uses the zero vector of size p in place of Y, it is unnecessary to use the simplest one. For example, one can assign biased estimates to Y. In this case, the estimator can be interpreted as a combined estimator that fuses a pair of unbiased and biased estimates [Green and Strawderman 1991]. While the existing literature in statistics focuses on theoretical perspectives of such combinations by the JS estimator, in practice, the biased estimates should be determined in an application-specific manner. These theoretical studies guide us to combine unbiased (but noisy) X and biased (but smooth) rendering estimates Y.

An intuitive alternative for such a combination is to estimate the errors of both inputs (X and Y) and blends those using the estimated errors per pixel. It can be theoretically ideal when the errors are accurate since combined estimates can dominate both inputs. However, such an oracle estimator does not exist in practice, and thus this MSE-based combination can be sensitive to the accuracy of the MSE estimation. On the other hand, the JS estimator enables combined estimates to dominate an unbiased input X for any Θ, irrespective of the errors of a biased input Y (e.g., even for the zero vector in the original JS paper), and it allows our framework to perform a more robust combination than the MSE-based approach, as shown in Fig. 2.

4 OUR NEURAL JAMES-STEIN COMBINER

This section proposes a novel combination framework (Fig. 3) that integrates unbiased and biased rendering estimates, where we aim at producing robust combined estimates whose errors become equal to or smaller than the unbiased estimates. Specifically, in Sec. 4.1 we introduce a localized formulation of the JS estimator that combines unbiased and biased image blocks, followed by a theoretical analysis of its MSE error in Sec. 4.2. Then, we present an optimization of the localized combiner to generate enhanced estimates in Sec. 4.3. The implementation details for our framework are presented in Sec. 4.4.

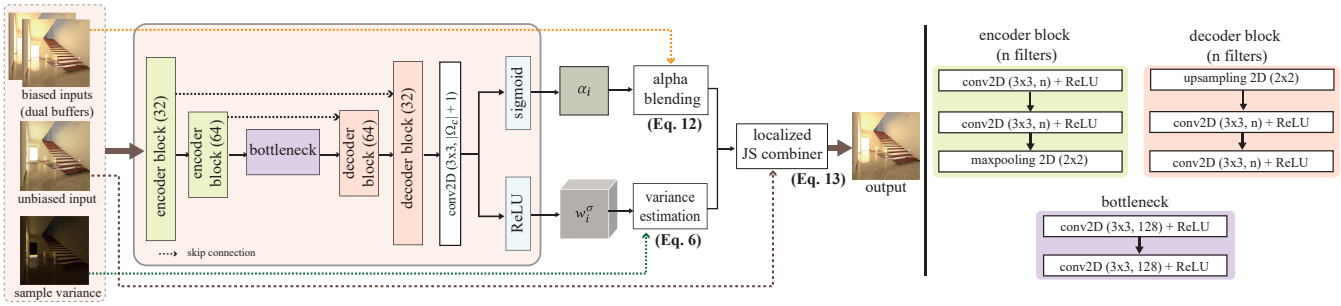

Fig. 3. An overview of our combination framework. We feed an unbiased image X, the variance of the X, and dual-buffered biased images $\hat{Y}^A$ and $\hat{Y}^B$ to the U-Net with the skip connections. The network infers the parameters per pixel i, i.e., α_i for constructing a single biased image Y^* from the $\hat{Y}^A$ and $\hat{Y}^B$ and w_i^σ for estimating the unknown variance of the X. Then, the James-Stein combiner generates the final output by fusing the X and Y^* locally guided by the estimated variance.

Table 1. Notations used throughout this paper.

Notation	Description
X, Y, Θ	unbiased, biased, and ground truth images
X_c, Y_c, Θ_c	image blocks centered at pixel c in X, Y, and Θ
x_c, y_c, θ_c	colors of pixel c in X, Y, and Θ
$\sigma_c^2 I$, $\xi_c^2 I$	variances of X_c and Y_c (I: $p \times p$ identity matrix)
Λ_c	bias of Y_c
$\delta(X_c, Y_c)$	James-Stein combiner for X_c and Y_c

4.1 Localized James-Stein Combiner

Suppose that we have a pair of unbiased and biased rendering estimates (X and Y) as input (e.g., a path-traced image and its denoised output). Given the input image pair (X, Y), our objective is to robustly estimate the ground truth image Θ by fusing the images via the JS estimator. A naïve approach for achieving this goal is to compute the original estimator (Eq. 2) using the two input images, but this could severely violate the homogeneous variance assumption (i.e., $X \sim N(\theta, \sigma^2 I)$ in Sec. 3) since the rendering estimates typically have heterogeneous variances.

To take advantage of the JS estimator for rendering images while alleviating the violation of the naïve technique, we conduct a localized combination using two image blocks per pixel, one for unbiased estimates, the other for the biased ones. Let us denote vectorized image blocks at pixel c by X_c and Y_c, each of which includes the pixel colors of X and Y within a window Ω_c centered at pixel c, respectively. We set the Ω_c to a small window (e.g., 15×15 pixels).

Let us define statistical models for the two image blocks as

$$X_c \sim N(\Theta_c, \sigma_c^2 I), \tag{3}$$

$$Y_c \sim N(\Theta_c + \Lambda_c, \xi_c^2 I), \tag{4}$$

where Θ_c and Λ_c are the vectorized ground truth and bias values. $\sigma_c^2 I$ and $\xi_c^2 I$ are the variances for X_c and Y_c, respectively. Also, we use lowercase symbols (e.g., x_c, y_c, and θ_c) for the c-th pixel values in X, Y, and Θ respectively to distinguish those from vectorized image blocks (e.g., X_c, Y_c and Θ_c). We shall treat the pixel values as 1D scalars for brevity since we process each color channel independently. Hence, the size of the image blocks (X_c and Y_c) is $p = |\Omega_c|$. Table 1 summarizes the notation used throughout this paper.

Given the localized statistical assumptions, we combine the pair of image blocks (X_c, Y_c) via the JS estimator (Eq. 1):

$$\delta(X_c, Y_c) = Y_c + \left(1 - \frac{(p-2)\sigma_c^2}{\|X_c - Y_c\|^2}\right)(X_c - Y_c), \tag{5}$$

which produces the estimates $\hat{\Theta}_c$ for the unknown image block Θ_c (not the entire image Θ). For the unknown variance σ_c^2, we locally average the sample variances of X_c for the estimate $\hat{\sigma}_c^2$:

$$\hat{\sigma}_c^2 = \frac{1}{\sum_{i \in \Omega_c} w_i^\sigma} \sum_{i \in \Omega_c} w_i^\sigma s_i^2, \tag{6}$$

where w_i^σ is a positive weight allocated to the i-th sample variance s_i^2. We omit the center pixel index c in $w_{c,i}^\sigma$ for brevity. We train the neural network (Fig. 3) by supervised learning so that it can infer the per-pixel weights properly (details will be given in Sec. 4.4).

The normalized positive weight ($w_i^\sigma / \sum_{i \in \Omega_c} w_i^\sigma$ in Eq. 6) makes the estimated variance to be in-between the maximum and minimum values of the input variances s_i^2 that are inversely proportional to the sample size. It makes that the estimated variance also decreases as the sample size increases. Hence the estimates of the localized JS combiner (Eq. 5) go to the unbiased block X_c that converges to the ground truth Θ_c in the limit. It allows the combined estimates to become consistent, regardless of both an estimation error in the variance and the homogeneous variance assumption.

As the estimator in Eq. 5 gives us the combined estimates $\hat{\Theta}_c$ per pixel c, a pixel i has multiple estimates, each of which is predicted from the blockwise estimation at its neighbor pixel c, i.e., $\{\delta(X_c, Y_c) | c \in \Omega_i\}$. We average the multiple estimates to produce our pixel estimate $\hat{\theta}_i$ for the i-th pixel color:

$$\begin{aligned}\hat{\theta}_i &= \frac{1}{|\Omega_i|} \sum_{c \in \Omega_i} \delta_i(X_c, Y_c) \\ &= \frac{1}{|\Omega_i|} \sum_{c \in \Omega_i} \left(y_i + \left(1 - \frac{(p-2)\sigma_c^2}{\|X_c - Y_c\|^2}\right)(x_i - y_i)\right),\end{aligned} \tag{7}$$

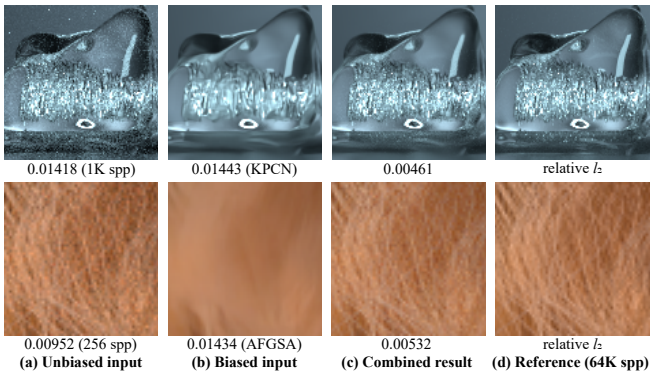

Fig. 4. Results of the James-Stein combination for unbiased and biased images. The learning-based denoisers (KPCN and AFGSA) do not improve the path-traced results due to their denoising bias (e.g., over-blurred edges). Our localized combiner restores the lost details while making the combined estimates dominate the unbiased estimates. The insets are from the scenes Glass-of-water (top) and Curly-hair (bottom).

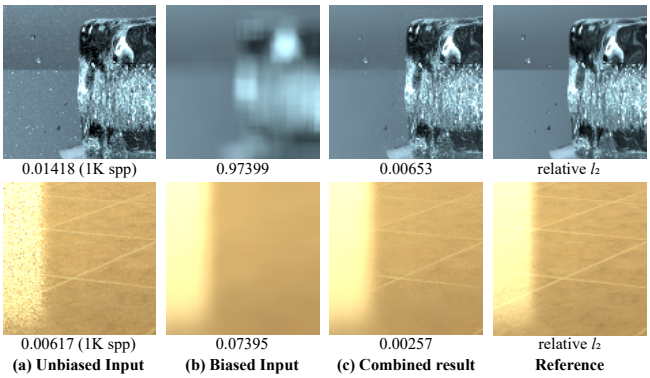

Fig. 5. James-Stein combined results using a severely biased input. We apply a box filter of size 15×15 to the unbiased input (a) per pixel and generate overly-blurred input images (b) for two scenes: Glass-of-water (top) and Staircase (bottom). The combined results (c) show lower errors than the unbiased input for this extreme case, thanks to the dominance property of the JS combiner.

where $\delta_i(X_c, Y_c)$ produces the estimate only for pixel i (i.e., an element in the per-block estimates $\hat{\Theta}_c$ obtained by $\delta(X_c, Y_c)$).

Fig. 4 shows the example results of the localized combiner applied to the results of image denoisers. As can be noticed in the figure, our technique reduces the excessive denoising bias introduced by state-of-the-art learning-based denoisers while producing improved numerical accuracy and visual fidelity.

4.2 Theoretical Discussion on the James-Stein Combiner

To evaluate the blockwise James-Stein combiner (Eq. 5), let us consider the mean squared error (MSE) of the combiner, $E||\delta(X_c, Y_c) - \Theta_c||^2$. By assuming X_c and Y_c are independent to each other, the error and its upper bound can be derived into compact forms [Green and Strawderman 1991] (our derivations are included in Appendix A):

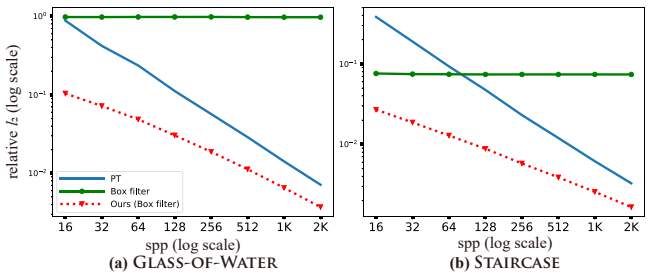

Fig. 6. Numerical convergences of the JS combiner when it takes severely biased images as biased input. The biased inputs were generated using a box filter with a large filtering window (15×15). Note that the biased inputs show almost constant errors over the tested ranges (16 to 2K spp) due to the fixed filtering bandwidth. However, the JS combiner using the severely biased inputs makes the combined results have lower errors than its unbiased input (PT).

$$E\|\delta(X_c, Y_c) - \Theta_c\|^2 = p\sigma_c^2 - \frac{(p-2)^2\sigma_c^4}{\|X_c - Y_c\|^2} \tag{8}$$

$$\leq p\sigma_c^2 - \frac{\sigma_c^4(p-2)^2}{\Lambda_c^T\Lambda_c + p\xi_c^2 + p\sigma_c^2}. \tag{9}$$

As the second term in Eq. 8 is positive, i.e., $\frac{(p-2)^2\sigma_c^4}{\|X_c - Y_c\|^2} \geq 0$ for $p \geq 3$, the MSE of $\delta(X_c, Y_c)$ is equal to or less than the error of the unbiased input X_c (i.e., $E||X_c - \Theta_c||^2 = p\sigma_c^2$). This dominance property can be advantageous, particularly for learning-based denoisers that typically have lower convergence rates than their input (i.e., unbiased MC rendering) since one could employ learning-based denoisers for various rendering scenarios without concerning the worst case, i.e., $E||Y_c - \Theta_c||^2 > E||X_c - \Theta_c||^2$. Note that the JS estimator (Eq. 5) does not take the errors (Λ_c and ξ_c^2) of the biased Y as input, and the dominance property holds for an arbitrarily chosen biased input even when its errors are much higher than the unbiased X_c.

To illustrate such behavior, in Figs. 5 and 6, we show example results where the biased inputs of the JS combiner are generated by applying a 15×15 box filter to the unbiased inputs. We have chosen the large window size to produce severely biased images by removing all high-frequency details in the unbiased images. The figures show that the JS combiner can still generate combined outputs with lower errors than its unbiased inputs, even for the severely biased estimates. It indicates that the JS combiner allows us to conduct a robust combination of the input pair.

It is also worth mentioning that the JS theory does not guarantee the dominance of the combined estimates over the biased input Y_c, unlike the unbiased X_c. Nevertheless, the combined results can be improved when taking more accurate biased estimates as input, as the error is affected by the MSE of the biased input estimates, i.e., $E||Y_c - \Theta_c||^2 = \Lambda_c^T\Lambda_c + p\xi_c^2$ in Eq. 9. As a result, we empirically found that the errors of the combined estimates tend to be smaller than those of the biased input, even when the biased input is much more accurate than the unbiased one.

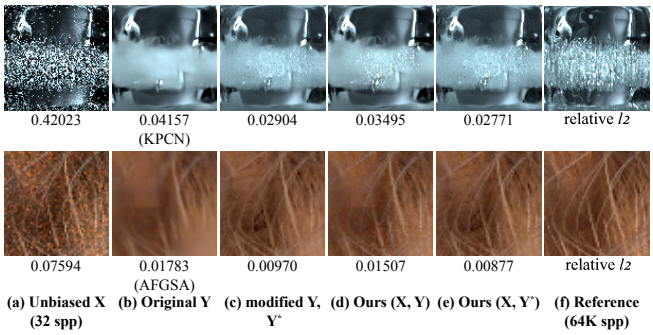

Fig. 7. We visualize the results of our JS combiner that uses either the original biased input Y (b) or the modified one Y^* (c). The combiner with the original Y helps the denoisers (KPCN and AFGSA) to reduce their excessive bias in the local areas from the scenes, Glass-of-water (top) and Curly-hair (bottom). This combination can be more effective when the combiner exploits the optimized Y^* with lower errors than the original Y.

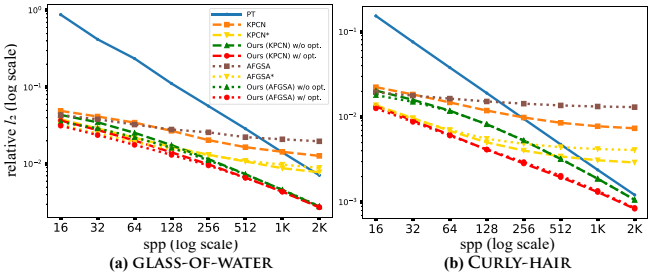

Fig. 8. Numerical accuracy of our localized JS combiner that uses either an original biased image Y (KPCN and AFGSA) or modified one Y^* (KPCN* and AFGSA*). The JS combiner with the original Y significantly improves the convergence of the input denoisers (KPCN and AFGSA) while enabling the denoiser to have lower errors than the unbiased PT. Our practical optimization (KPCN* and AFGSA*) further decreases the errors of our post-correction, mainly when a linear regression, which uses both the original biased input and G-buffers as features, can produce an enhanced biased input (e.g., the Curly-Hair scene).

4.3 Optimization for the Localized James-Stein Combiner

We present a practical optimization for the localized JS combiner (Eq. 5) so that its error can be further reduced even when its biased input Y has much lower errors than the unbiased input X, which often occurs when small sample counts are used. Unfortunately, we cannot directly modify the input generation process since we treat such a process as a black box.

As a practical alternative, we construct a variant Y^* of the biased input Y using the regression-based approach [Bitterli et al. 2016; Moon et al. 2014]. We approximate the unbiased input X with a linear function on a feature space $\mathbf{f}$, which includes the biased input Y and rendering-specific features (albedo, normal, depth and visibility), so that the Y^* can contain high-frequency information in both the original Y and the auxiliary buffers.

Specifically, we generate the two unbiased images, X^A and X^B, independent of each other. For example, the X^A contains the sample means of the first half samples, and the latter X^B has the means of the other samples. We also generate two auxiliary G-buffers in the same way. We then denoise each half-buffer using the given denoiser and obtain two denoised images, Y^A and Y^B. Let us denote the two feature sets $\mathbf{f}^A$ and $\mathbf{f}^B$, each of which contains a half-buffered denoised image and G-buffers.

Once the crossed half-buffer pairs, (X^A and $\mathbf{f}^B$) and (X^B and $\mathbf{f}^A$), are prepared, we compute a linear model using each pair by solving a weighted least squares minimization:

$$\hat{\beta}_c^A = \underset{\beta_c}{\operatorname{argmin}} \sum_{i \in \Omega_c} \left\| x_i^A - \beta_c \begin{bmatrix} 1 \\ \mathbf{f}_i^B - \mathbf{f}_c^B \end{bmatrix}^T \right\|^2 exp\left(\frac{-\left\| y_i^B - y_c^B \right\|^2}{2\kappa + \epsilon} \right), \tag{10}$$

which infers the coefficients of a linear function, $\hat{\beta}_c^A$. Note that the optimization (Eq. 10) uses only a pair (i.e., X^A and $\mathbf{f}^B$), and thus this should be performed again using the other pair (X^B and $\mathbf{f}^A$). We set the window size $|\Omega_c|$ to 51×51. For brevity, we shall discuss the first optimization (i.e., with X^A and $\mathbf{f}^B$), since the second one can be done analogously. We solve the least-squares optimization using the well-known closed-form solution, i.e., the normal equation.

In Eq. 10, $\mathbf{f}_i^B$ and $\mathbf{f}_c^B$ are per-pixel feature vectors at pixel i and c. κ is a bandwidth term that controls the weight $exp(\cdot)$ assigned to the pixel i and $\epsilon = 0.01$. One can estimate the per-pixel bias and variance of the linear function and optimize the bandwidth at each pixel, like classical regression-based denoisers [Bitterli et al. 2016; Moon et al. 2014]. Unfortunately, this additional optimization can be very costly since it often requires conducting such a regression multiple times while varying the bandwidth to select the best one per pixel. Thus, we take a much simpler option, a globally fixed bandwidth κ, instead of a locally varying one. Specifically, we assign the average variability of the biased input Y to the κ and use it for all pixels, i.e., $\kappa = \left\| Y^A - Y^B \right\|^2 / 2N$ where N is the number of pixels. Note that the dominance property of the JS estimator is tolerant to this heuristically chosen bandwidth since the property holds irrespective of the errors of the biased input (see Eq. 9).

As the linear model approximates a local region Ω_c (not just the pixel c), a pixel i can be linearly predicted by multiple linear models in its neighboring pixels $c \in \Omega_i$. We combine the multiple values using the weight (i.e., $\exp(\cdot)$ in Eq. 10) for the output pixel i:

$$\hat{y}_i^A = \frac{1}{W_i} \sum_{c \in \Omega_i} \left(\hat{\beta}_c^A \begin{bmatrix} 1 \\ \mathbf{f}_i^B - \mathbf{f}_c^B \end{bmatrix}^T \right) exp\left(\frac{-\left\| y_i^B - y_c^B \right\|^2}{2\kappa + \epsilon} \right), \tag{11}$$

where W_i is the normalization term, i.e., $W_i = \sum_{c \in \Omega_i} exp(\cdot)$. We also employ the same regression but with a different pair (X^B and $\mathbf{f}^A$), so that we can achieve the $\hat{y}_i^B$ per pixel i. Our final task is to combine the two regression outputs at pixel i, and this can be simply achieved by a per-pixel alpha blending:

$$y_i^* = \alpha_i \hat{y}_i^A + (1 - \alpha_i) \hat{y}_i^B, \tag{12}$$

where the blending factor α_i ($0 \le \alpha_i \le 1$) is inferred by our neural network per pixel i.

This process provides us a modified bias input Y^* through the two regression outputs ($\hat{Y}^A$ and $\hat{Y}^B$), and we perform our localized combination using the unbiased X and Y^* instead of the original Y. Additionally, we exploit the positive part JS (Eq. 2) that dominates

the original JS estimator, and thus the equation (Eq. 7) for our pixel estimates is updated into the final form:

$$\hat{\theta}_i = \frac{1}{|\Omega_i|} \sum_{c \in \Omega_i} \left(y_i^* + \left(1 - \frac{(p-2)\sigma_c^2}{\|X_c - Y_c^*\|^2}\right)^+ (x_i - y_i^*) \right), \tag{13}$$

where $(\cdot)^+ = max(0, \cdot)$.

Figures 7 and 8 show the qualitative results and the numerical convergences of our localized combiner that uses either the original Y and modified Y^*. As observed, our proposed optimization can further reduce the errors of the combined estimates, mainly for small to moderate sample counts. As the sample size increases, the errors of the combined estimates largely depend on the JS combiner alone. Note that the dominance property of our framework is achieved by the JS combiner, irrespective of the practical optimization.

Our framework employs a deep neural network (Fig. 3) that estimates the unknown variance (in Eq. 6) and the alpha-blending factor (in Eq. 12). One may consider a simple alternative to the use of a neural network. For example, one could use a Gaussian filter instead of the variance estimation process and assign a fixed alpha (e.g., $\alpha_i = 0.5$) for the blending. We observed that this classical alternative could produce lower errors than unbiased inputs. However, its combined results were sensitive to its parameter (e.g., the bandwidth of the Gaussian filter) and not robust as our current learning-based approach, primarily when fireflies exist in the unbiased inputs, as analyzed in our supplementary report.

4.4 Implementation of the Neural James-Stein Combiner

To generate our final output, i.e., the combined pixel estimates $\hat{\theta}_i$ (Eq. 13), we should provide the estimated variance $\hat{\sigma}_c^2$ and the inferred biased image Y^*. To this end, we should determine the parameters (w_i^σ in Eq. 6 and α_i in Eq. 12) for these estimation processes.

Inspired by existing learning-based denoisers [Bako et al. 2017; Kalantari et al. 2015], we exploit a deep neural network, i.e., a simple U-Net [Ronneberger et al. 2015] with the skip connections (see Fig. 3), which determines the parameters per pixel. Specifically, we feed an unbiased input X, the variance of the X, and dual-buffered regression outputs ($\hat{Y}^A$ and $\hat{Y}^B$) to the U-Net. The last convolutional layer of the network uses $|\Omega_c| + 1$ filters of size 3×3 so that it can produce the w_i^σ and α_i per pixel. We use ReLU and a sigmoid activation to enforce the constraints to the w_i^σ and α_i, i.e., $w_i^\sigma > 0$ and $0 \le \alpha_i \le 1$.

We then estimate the variances of the unbiased input X using w_i^σ (Eq. 6) and the modified image Y^* using the dual-buffered regression outputs $\hat{Y}^A$ and $\hat{Y}^B$ using α_i. Finally, the pixel estimates $\hat{\theta}_i$ are inferred via the localized JS combiner (Eq. 13) that fuses its two inputs X and Y^* guided by the estimated variance.

To let our network adjust the parameters for our localized combiner appropriately, we train the network by supervised learning. Specifically, we have used the relative l_2 error [Rousselle et al. 2011] with a signed log transformation [Kettunen et al. 2019] for the loss:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \frac{\|\tau(\hat{\theta}_i) - \tau(\theta_i)\|^2}{avg(\tau(\theta_i))^2 + \epsilon}, \tag{14}$$

where $\tau(\theta_i) = sign(\theta_i) log(1 + |\theta_i|)$ and $\epsilon = 0.01$. Also, N and $avg(\tau(\theta_i))$ are the total pixel count and the average intensity of the color $\tau(\theta_i)$, respectively. Given such a configuration, the number of our network parameters is roughly 0.54M, which is much smaller than those of recent learning-based denoisers (e.g., 5.84M and 9.28M for [Bako et al. 2017] and [Yu et al. 2021], respectively).

Fig. 9. Example images in our training dataset. We have randomized the public scenes (Bathroom, Bedroom, Car2, Classroom, Dining-room, Spaceship, and Wooden staircase) by transforming the camera views and materials, and generated the training images using the randomized scenes.

5 RESULTS AND DISCUSSIONS

This section verifies that our JS combiner can improve different types of unbiased and biased rendering pairs. Specifically, we apply the combiner to state-of-the-art image denoisers, i.e., the kernel-predicting neural network (KPCN) [Bako et al. 2017] and auxiliary feature guided self-attention (AFGSA) [Yu et al. 2021] so that their path tracing input and denoising output can be integrated into an improved result.

We compare our combination approach with the recent post-denoisers, deep combiner (DC) [Back et al. 2020], ensemble denoising (ED) [Zheng et al. 2021], and progressive denoising (PD) [Firmino et al. 2022]. We also apply our method to a classical but consistent denoiser, nonlinearly weighted first-order regression (NFOR) [Bitterli et al. 2016]. Lastly, we fuse the two rendering images through our JS combiner, each of which is generated by different light transport algorithms, i.e., bidirectional path tracing (BDPT) [Lafortune and Willems 1993] and stochastic progressive photon mapping (SPPM) [Hachisuka and Jensen 2009].

We validate the JS combiner with the various image pairs rendered using seven test scenes, Dragon, Veach-Ajar, Staircase, Curly-Hair, Glass-of-Water, Pool, and Bookshelf (as shown in Fig. 10 and 17). We use the Tungsten renderer for the Veach-Ajar, Curly-Hair, and Glass-of-Water. We employ the PBRT [Pharr et al. 2016] for the Dragon and Staircase and Mitsuba [Jakob 2010] for the Pool and Bookshelf.

All timings are measured on an AMD Ryzen Threadripper 2990WX CPU and an Nvidia RTX 3090 GPU, and as a numerical metric, we use the relative l_2 error [Rousselle et al. 2011].

Training details of learning-based methods. We have used the same training dataset for all the supervised learning methods, including ours, for a fair comparison. We have generated 1284 image sets (90% for training and 10% for validation) using scenes in a public repository [Bitterli 2016] (see some examples in Fig. 9). Specifically, we have used the path tracing implementation of the Tungsten renderer for generating the pairs of noisy and reference images. We have followed the sample counts (i.e., 32 and 32K samples per pixel (spp)) used in [Yu et al. 2021] when rendering the image pairs. Given the dataset, we have trained KPCN and AFGSA using the public implementations (including parameter settings) released by the authors.

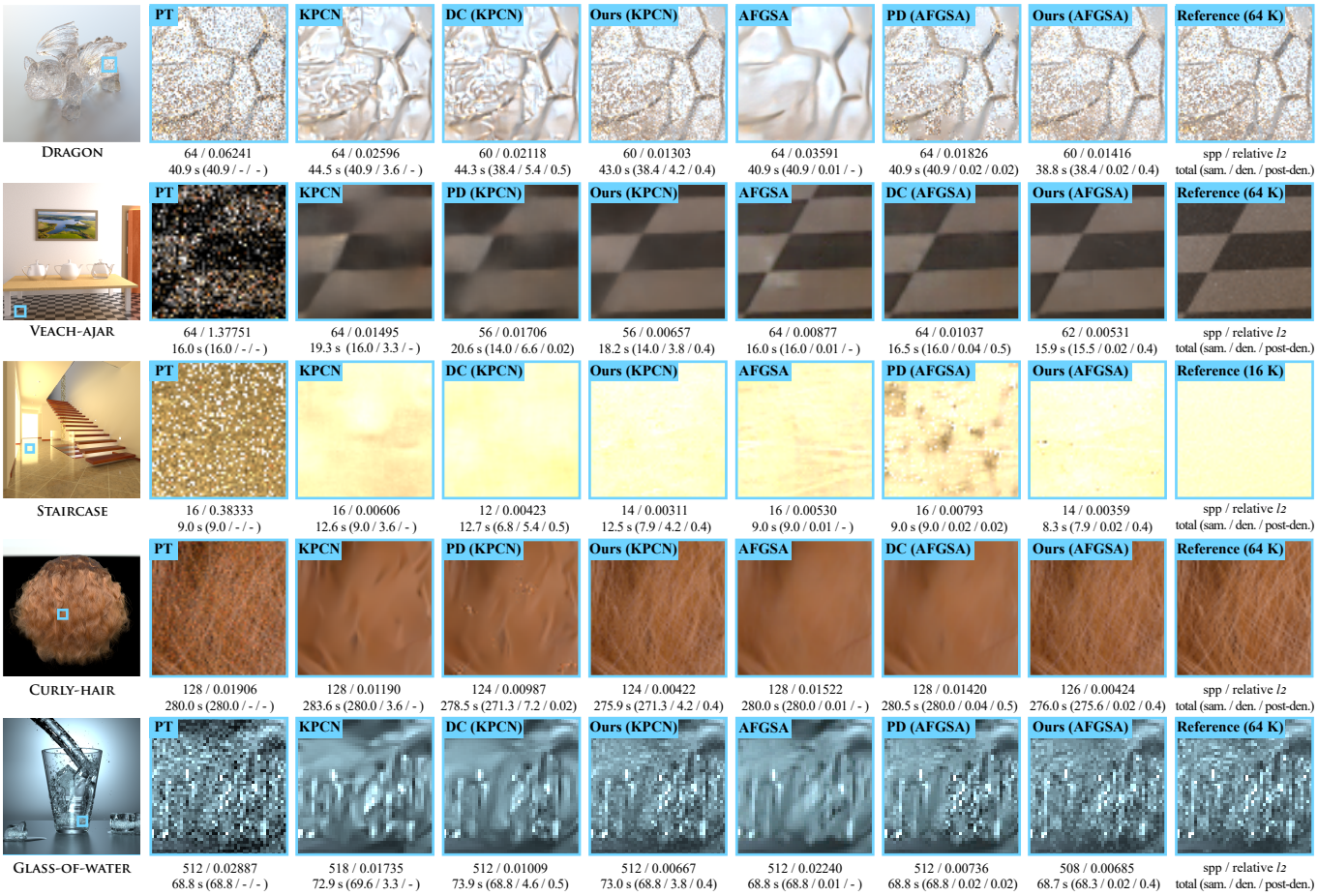

Fig. 10. Equal-time comparisons between our technique and the post-denoisers (DC and PD). We report total rendering times and their breakdowns (i.e., sampling, denoising, and post-denoising times). We apply these combination methods, including ours, to the recent learning-based denoisers, KPCN and AFGSA. The existing methods do not consistently reduce the errors of their input denoisers (e.g., DC for the VEACH-AJAR and PD for the VEACH-AJAR and STAIRCASE). On the other hand, our method consistently improves the visual quality and numerical accuracy of the input denoisers. The supplemental includes the full-resolution images.

We have allocated enough training times for the methods and then selected the best epoch using the validation set.

We have generated the biased inputs for learning-based post-denoisers (PD and ours) using the trained KPCN and AFGSA. Then, we have trained ours and the network of PD using the path tracing input and denoising output pairs. For the PD, we have used the public code provided by the authors. For ours, the image patch and batch sizes have been set to 128×128 and 4 respectively, and we have trained our network for 50 epochs using the Adam optimizer [Kingma and Ba 2014] with a learning rate of 0.0001, and it has taken approximately eight hours for training. We have implemented our framework using Tensorflow [Abadi et al. 2015].

Comparisons with post-denoisers. We compare our technique with DC and PD in Figs. 10 and 11. DC improves the input denoisers (KPCN and AFGSA) for relatively small sample counts but does not prevent the denoisers from being worse than the unbiased PT. PD makes the learning-based denoisers have lower errors than the PT, but it deteriorates their input denoisers for the VEACH-AJAR and STAIRCASE scenes even up to large sample counts (e.g., up to 2K spp for the VEACH-AJAR and 1K for the STAIRCASE in Fig. 11). On the other hand, our method consistently improves the state-of-the-art denoisers by forcing those to dominate the unbiased PT.

We also compare our technique with ED in Figs. 12 and 13. While ED was originally designed for combining multiple denoising estimates that can be arbitrary, an effective input configuration can be the combination of a consistent denoiser (NFOR) and a learning-based one (KPCN or AFGSA). Also, one may consider a combination of the unbiased input and a learning-based denoiser, i.e., the same input configuration as our technique. The figures show that the ED with the unbiased and biased input pair can produce much higher errors than its biased input due to the fundamental challenge in the error analysis. When the ED takes a consistent denoiser (NFOR) in place of the unbiased input, their combination can be robust and inherit the consistency of the NFOR. However, the combined results

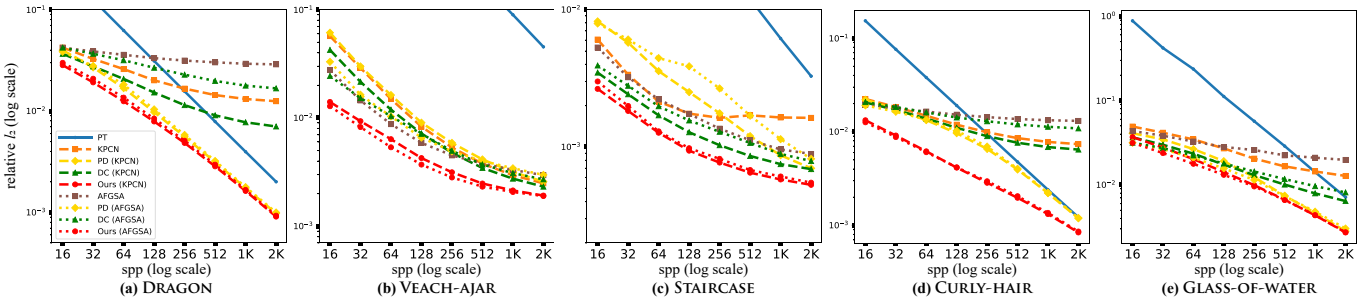

Fig. 11. Numerical convergences of denoisers (KPCN and AFGSA) and post-denoisers (PD, DC, and ours). The KPCN and AFGSA effectively reduce the errors of the unbiased input (PT) for the STAIRCASE and VEACH-AJAR but deteriorate the PT for the other scenes due to their slow convergence. PD improves the convergence rates of KPCN and AFGSA but degenerates the input denoisers for the STAIRCASE and VEACH-AJAR scenes. It indicates that an MSE-based technique can be theoretically appealing, but in practice, a robust correction can be challenging due to the difficulty of an accurate MSE estimation. DC does not require the error analysis on biased inputs, but it is noticeable that DC does not improve the convergence rates of the biased inputs. On the other hand, our technique consistently improves the input denoisers.

do not always have lower errors than their inputs, even in this case. Also, additional usage of a classical method introduces a noticeable computational overhead to the post-denoiser since such a classical method (e.g., NFOR) typically requires an expensive optimization.

It indicates that a robust post-correction guided by MSE-based optimization can be challenging in practice since their post-correction mainly relies on an accurate error analysis of input denoisers. On the other hand, our JS combiner does not require the precise estimation of per-pixel MSEs of an input denoiser. As shown in Eq. 13, it only needs the estimated variance of the unbiased input (e.g., PT). This relaxation allows us to improve an input denoiser consistently while preventing the denoiser from being worse than the unbiased input.

In addition to the same-time comparisons, our supplementary report includes equal-sample comparisons with the tested post-denoisers.

Our technique with ensemble denoiser. As our technique can take an arbitrary pair of unbiased and biased rendering images as input, one can feed an unbiased input and the post-denoising results of ED to our method. To verify this compatibility with ED, we use a learning-based denoiser (AFGSA) and a consistent denoiser (NFOR) as their input denoisers. Fig. 14 compares the numerical accuracy of ED with and without our additional post-correction. The ED without our method improves one of their input (AFGSA) but fails to enhance another input (NFOR) consistently. It also produces slightly higher errors than the unbiased PT for the Curly-Hair scene with 2K spp. On the other hand, our method improves the accuracy of the ED and allows the ED to produce lower errors than its unbiased and biased inputs.

Joint training of KPCN with our technique. We treat a learning-based denoiser as a black box like the other post-denoisers. An attractive alternative is to directly integrate the localized JS combiner into a denoising neural network and train this combined one. We insert the localized JS combiner, which takes an unmodified biased input Y, into the KPCN networks for diffuse and specular components respectively, so that the JS combiner takes the denoising output generated by their output kernels. Note that the original networks do not internally produce the dual-buffered denoising outputs, and thus we do not utilize the practical optimization (Sec. 4.3). For the estimated variance (Eq. 6), we exploit eight convolutional layers where the last one produces the weights w_i^σ. We do not change the other parts of the existing networks. This integration into an denoising neural network is straightforward, but the KPCN jointly trained with the JS combiner produces much-improved numerical results than without our method, as shown in Fig. 15. In particular, its convergence rates improve while dominating the unbiased counterpart.

Comparisons with a consistent denoiser. One may consider feeding a consistent denoiser to our technique as a biased input. NFOR has a strong convergence due to its sophisticated bandwidth adaptation, but it does not necessarily indicate that it can produce more accurate denoising estimates than its unbiased input for all image local regions. Our method can help a denoiser avoid its worst-case scenarios per local region (i.e., image blocks), and thus even consistent methods can benefit from the dominance property of the JS combiner. Note that our method has not been trained with this biased input, and thus this test with the unseen input can be technically challenging. Nonetheless, our method improves the consistent denoiser even for large sample counts (e.g., 2K), as shown in Fig. 16.

Comparisons with BDPT and SPPM. We also demonstrate that one can combine unbiased and biased methods with fast convergence rates. A typical example is the pair of BDPT and SPPM, each of which produces the correct rendering result in the limit. Note that our training dataset did not include BDPT and SPPM results. As a result, combining the two inputs can be technically challenging since SPPM estimates can have different statistical properties (e.g., high-frequency noise on glossy surfaces) compared to the trained denoising inputs (KPCN and AFGSA). We have used the BDPT and SPPM implementation (and built-in parameters for the light transport algorithms) in the Mitsuba renderer.

In Fig. 17 and Fig. 18, we compare the JS combiner to our two input methods (BDPT and SPPM) visually and numerically, given equal-time budgets. For example, the reported rendering times for our technique are the sum of the input generation and inference

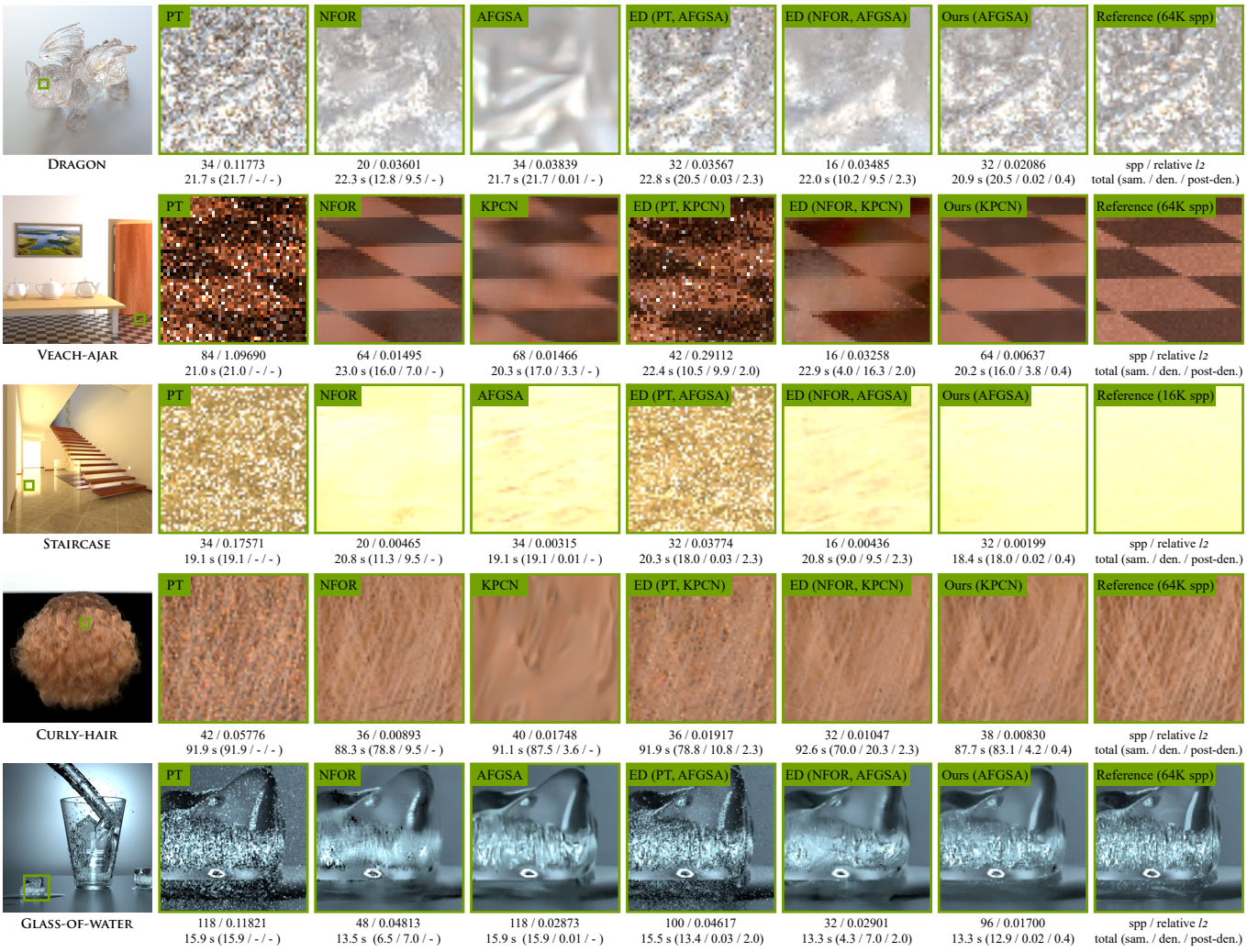

Fig. 12. Equal-time comparisons between our technique and ED. We use a consistent denoiser (NFOR) and a learning-based one (KPCN or AFGSA) as the input of ED, i.e., ED (NFOR, KPCN) and ED (NFOR, AFGSA). We also test ED with the same input configuration as ours (i.e., PT and a learning-based technique). When ED combines an unbiased input (PT) and learning-based denoising results (KPCN or AFGSA) like our technique, their combined results exhibit high-frequency noise propagated from the unbiased one (PT) due to the difficulty in estimating the per-pixel MSE of their inputs. In this case, the ED fails to reduce the errors of its biased input for the Veach-Ajar, Staircase and Glass-of-water scenes. Feeding consistent denoising results (NFOR) instead of the unbiased PT to ED produces visually enhanced output but still shows higher errors than one of its input denoisers for the Veach-Ajar, Staircase, Curly-Hair and Glass-of-Water scenes. On the other hand, our method robustly improves our input denoiser (KPCN and AFGSA) given the same-time budgets. We include the full-resolution images in the supplemental material.

time. To generate our two inputs, we allocate roughly equal times for BDPT and SPPM, respectively. As a result, the BDPT and SPPM techniques, which are compared in the figures, use approximately twice more rendering times than our input BDPT and SPPM.

As shown in Fig. 17, BDPT produces more accurate results than SPPM for glossy surfaces since SPPM relies on the classical distributed ray tracing [Cook et al. 1984] when a ray hits glossy surfaces [Hachisuka and Jensen 2009]. On the other hand, SPPM generates smoother results than BDPT for caustics and diffuse surfaces. Our technique locally blends the two input estimates with distinct strengths and produces visually improved results. It can also be observed that the combiner consistently enhances the numerical accuracy of the input methods in Fig. 18.

One may also consider an improved input in the place of SPPM. For example, CPPM [Lin et al. 2020] (i.e., an optimized SPPM) that adjusts the bandwidths for its photon density estimation can be used. Also, VCM [Georgiev et al. 2012], which fuses BDPT and SPPM, can be an option for our biased input, and in this case, our framework takes BDPT and VCM estimates as an input pair, similar to the demonstrated combination with the post-denoisers. We leave these additional tests to future work since our combined estimates can be improved when taking more accurate input (Sec. 4.2).

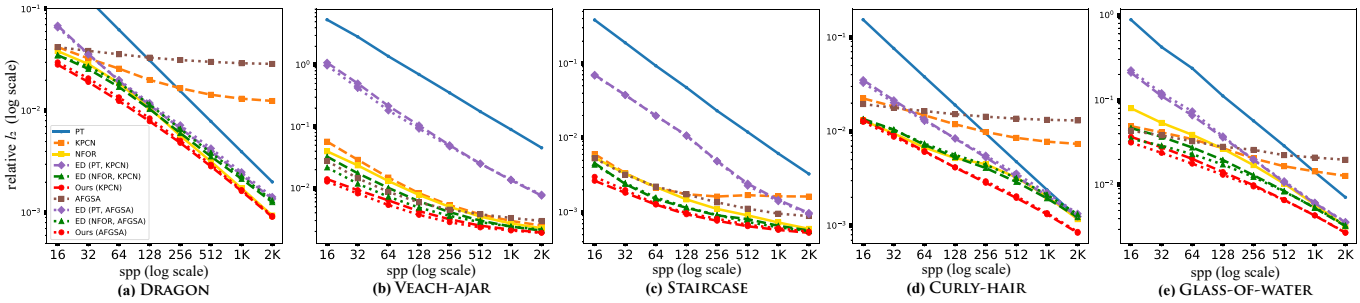

Fig. 13. The ED, which takes a pair of unbiased and biased images, i.e., ED (PT, KPCN) and ED (PT, AFGSA), shows much higher errors than the other biased results, including ours. This MSE-based method can be more robust when it takes only biased inputs, i.e., ED (NFOR, KPCN) and ED (NFOR, AFGSA), but produces higher errors than its input NFOR for the DRAGON (from 256 to 2K spp) and CURLY-HAIR (from 16 to 128 spp and 2K spp) scenes. Our technique, however, robustly improves our input denoisers and shows the best errors over the tested ranges.

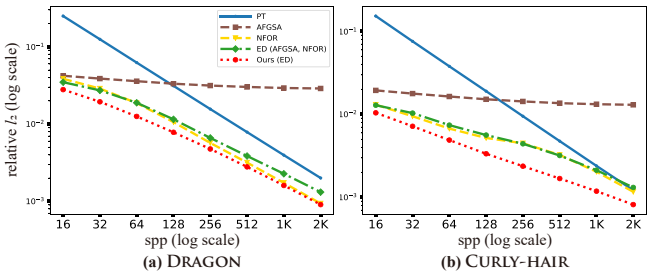

Fig. 14. Numerical convergences of ED with and without our technique. We employ AFGSA and NFOR for the two denoising inputs of ED, and our method takes an unbiased image (PT) and the results of ED as input. Our technique improves the accuracy of our biased input (ED) consistently.

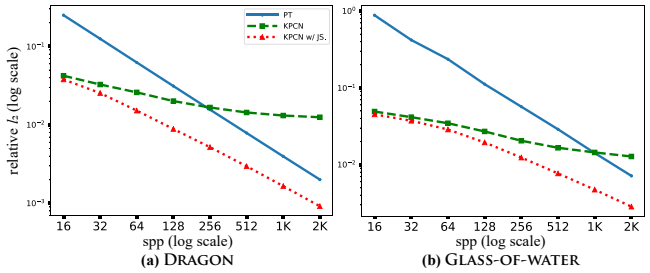

Fig. 15. Numerical accuracy of KPCN jointly trained with our technique. We have integrated our localized JS combiner into the existing KPCN network and trained the combined network, and this joint training improves the numerical convergences of the learning-based denoiser.

Our computational overhead. To produce the variant Y^*, we require the dual-buffered biased images (Y^A and Y^B in Sec. 4.3). Specifically, for NFOR, we use its dual-buffered output images, and thus it does not require an additional process for the method. For KPCN, we generate the two biased images by sharing its output kernels inferred from its original input buffers, and thus additional overhead is negligible. For AFGSA, we run the method twice, and its denoising time increases from 0.01 s to 0.02 s for 1024×1024 images.

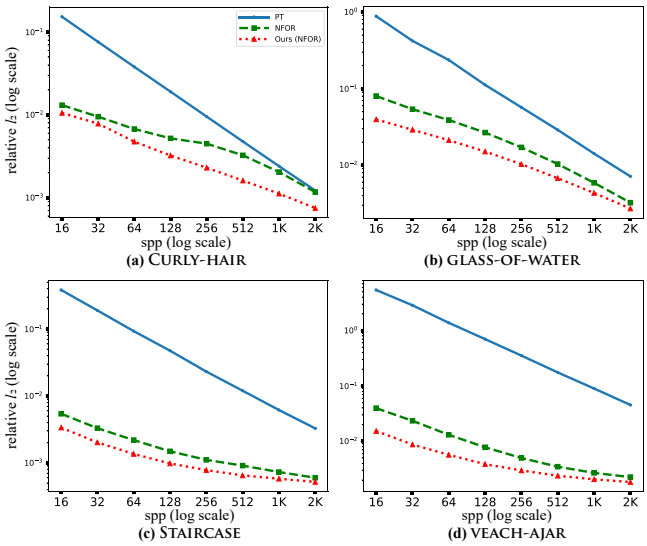

Fig. 16. Numerical comparisons of our technique and a consistent denoiser, NFOR. Our technique takes the biased estimates of the classical denoising technique as a biased input and improves its numerical accuracy by combining it with its unbiased input (PT).

Our own runtime overhead, which excludes the sampling and dual-buffer generation times, only depends on the image resolutions of the inputs. Specifically, generating the two regression outputs $\hat{Y}^A$ and $\hat{Y}^B$ (Sec. 4.3) takes 0.40 s, and the other parts for the variance estimation and JS combination take 0.02 s for 1024×1024 images. As a result, the total overhead for the inference is 0.42 s for 1024×1024 images. The equal-time comparisons in Figs. 10 and 12 verify that our computational overhead can be acceptable for the tested offline rendering scenarios.

Analysis of the independence between unbiased and biased input. We have shown that the error of combined estimates $\hat{\Theta}_c$ is equal to or smaller than the unbiased input X_c, under the independence assumption between the unbiased and biased input in Sec. 4.2. Strictly

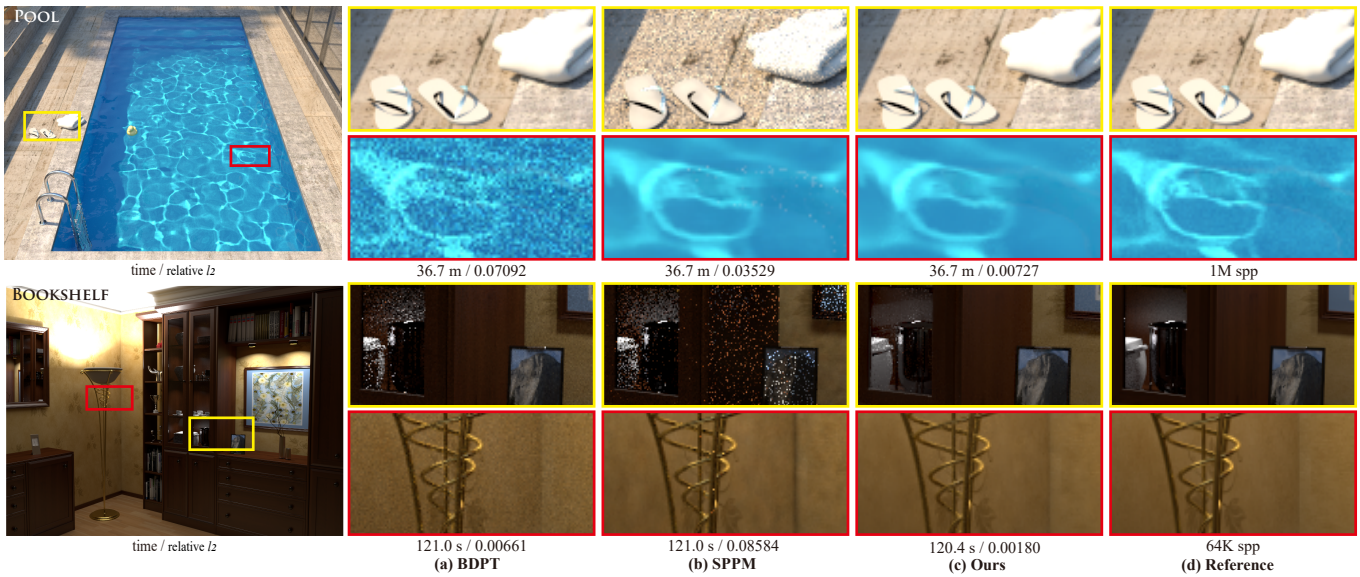

Fig. 17. Equal-time comparisons of our technique with BDPT and SPPM. BDPT (a) produces less noisy results on glossy surfaces (the zoomed areas in yellow-colored boxes) than SPPM (b). For example, the results in the first row are on a glossy surface with both diffuse and specular reflections, and those in the third-row show rendering estimates on a shiny bookshelf with glossy reflections. On the other hand, SPPM generates smoother rendering results for the caustics and diffuse surfaces. Our technique (c) locally fuses their rendering estimates and produces visually more pleasing results. The reference images (d) are rendered using BDPT with large sample counts.

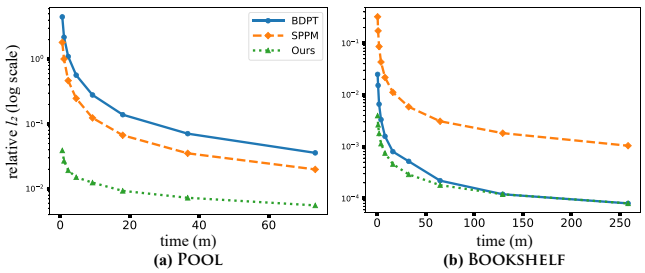

Fig. 18. Numerical convergence plots for the Pool and Bookshelf scenes. Our technique consistently produces more accurate results than the BDPT and SPPM by locally combining their output images.

speaking, the assumption does not hold for denoising (also post-denoising) applications since denoised results correlate to the denoising input (i.e., PT images).

As an alternative, one can alleviate such dependency by feeding a crossed-pair $(X^A, \hat{Y}^B)$ or $(X^B, \hat{Y}^A)$ to the JS combiner instead of using the full-buffer input (X, Y^*), and then blend the two combined images using the alpha blending scheme (Eq. 12). Table 2 compares the JS combiner that uses either an entire buffer or dual buffers. Their numerical quality is overall similar, but the combination using the full buffer generates slightly more accurate results. The JS combiner is a biased estimator, and thus averaging its combination results from each half-buffer does not reduce its bias by half. Consequently, this empirical study leads us to employ our practical option that uses the full-buffered pair (X, Y^*) for the combination.

Limitations and future work. The error analysis for the JS combiner (Sec. 4.2) shows that its MSE is equal to or smaller than the error of an unbiased input for an image block, i.e., $E\|\delta(X_c, Y_c) - \Theta_c\|^2 \leq E\|X_c - \Theta_c\|^2$. However, it does not guarantee that individual pixels in the combined image are more accurate than those in the unbiased one. As a result, the per-pixel MSEs for some pixel colors in the combined image (e.g., $E\|\hat{\theta}_i - \theta_i\|^2$) can be higher than the errors of unbiased pixel colors (e.g., $E\|x_i - \theta_i\|^2$). Furthermore, the MSEs only explain the expected behavior of the actual l_2 errors (e.g., $\|\hat{\theta}_i - \theta_i\|^2$), and thus the oracle per-pixel errors in the combined image can be higher than those of the unbiased input. In addition to these technical limitations inherited from the JS theories, we assume that the unbiased estimates have a homogeneous variance in a small window Ω_c and estimate the variance per pixel. Strictly speaking, it is an approximation since the variance can vary even within the local area.

In Fig. 19, we visualize the relative l_2 errors per pixel, i.e., the oracle errors computed using the references, for our two inputs and our output. Our output estimates have higher errors than the unbiased input at some pixels (e.g., visualized by reddish colors in Fig. 19 (d)). Nonetheless, our per-pixel errors are mostly smaller than the unbiased input (and the biased input), and it allows our final result to have higher numerical accuracy, i.e., a smaller average of the per-pixel errors for all pixels, than the unbiased input.

One can feed temporally processed rendering images to our framework, and in this case, we can correct the input images independently frame by frame. However, it can be more desirable to exploit the animated sequences both spatially and temporally. Also, it can be necessary to train our neural network using various unbiased

Table 2. Numerical accuracy of the JS combiner that uses either full-buffered or dual-buffered input pair.

scene	spp	PT	KPCN	JS (full)	JS (dual)
CURLY-HAIR	128	0.01906	0.01190	**0.00414**	0.00432
	512	0.00477	0.00855	**0.00197**	0.00216
	2K	0.00122	0.00737	**0.00084**	0.00102
scene	spp	PT	AFGSA	JS (full)	JS (dual)
GLASS	128	0.11176	0.02817	**0.01316**	0.01340
OF	512	0.02887	0.02240	**0.00661**	0.00697
WATER	2K	0.00715	0.01978	**0.00275**	0.00275

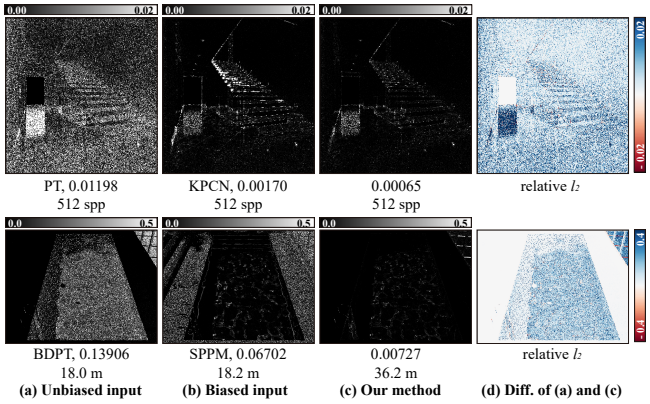

Fig. 19. Analysis of our limitation for the STAIRCASE and POOL scenes (top and bottom rows, respectively). We visualize the relative l_2 errors per pixel for our input ((a) and (b)) and output images (c). Also, the signed differences in the errors between the unbiased input and our estimates are shown in (d). For example, the pixels with bluish colors indicate that our method has lower errors than the unbiased, and our technique shows higher errors on the pixels with reddish colors. While the average errors of our method (0.00065 and 0.00727 in (c)) are smaller than the two inputs (0.01198 and 0.13906 in (a) and 0.00170 and 0.06702 in (b)), our per-pixel errors for some pixel colors can be higher than the unbiased input, as visualized in (d).

and biased rendering estimates (e.g., BDPT and SPPM images) to achieve higher combination quality. Lastly, we would like to optimize the presented framework to support real-time rendering scenarios (e.g., [Bitterli et al. 2020; Chaitanya et al. 2017]) using lightweight neural networks (e.g., [Thomas et al. 2020]). We leave these interesting investigations as future work.

6 CONCLUSION

This paper introduces the JS estimator, an established theory in statistics, to the rendering field. We specialize the general estimator into a localized estimator that optimally combines unbiased and biased image blocks per pixel through a deep neural network. A noteworthy property of this combination is that the error of the locally merged estimates can be equal to or smaller than the unbiased input, even with a severely biased image Y. We take advantage of this technical property of the JS combiner for assisting an input denoiser to have smaller errors than its unbiased input. It enables a non-consistent denoiser (i.e., learning-based methods) to be consistent while improving its error reduction rates more robustly than MSE-based post-denoisers. We believe that our technical approach can inspire additional rendering applications of the JS estimator where unbiased and biased rendering pairs are available.

ACKNOWLEDGMENTS

We appreciate the anonymous reviewers for the constructive comments. We also thank the following authors and artists for each scene: Mareck, SlykDragko, Wig42, NovaZeeke and thecali (training scenes in Fig. 9), aXel (GLASS-OF-WATER), Cem Yuksel (CURLY-HAIR), NewSee20135 (STAIRCASE), Ondřej Karlík (POOL), Tiziano Portenier (BOOKSHELF for the Mitsuba porting), and Christian Schüller (DRAGON). 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). Jose A. Iglesias-Guitian was supported by a 2021 Leonardo Grant for Researchers and Cultural Creators, BBVA Foundation. He also acknowledges the UDC-Inditex InTalent programme, the Spanish Ministry of Science and Innovation (AEI/PID2020-115734RB-C22 and AEI/RYC2018-025385-I), Xunta de Galicia (ED431F 2021/11) and EU-FEDER Galicia (ED431G 2019/01).

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, Dandelion 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. https://www.tensorflow.org/ Software available from tensorflow.org.

Jonghee Back, Binh-Son Hua, Toshiya Hachisuka, and Bochang Moon. 2020. Deep combiner for independent and correlated pixel estimates. *ACM Trans. Graph.* 39, 6 (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 (2017), 14 pages.

Alvin J Baranchik. 1964. *Multiple regression and estimation of the mean of a multivariate normal distribution.* Technical Report. STANFORD UNIV CALIF.

Alvin J Baranchik. 1970. A family of minimax estimators of the mean of a multivariate normal distribution. *The Annals of Mathematical Statistics* (1970), 642–645.

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. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 107–117.

Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. 2020. Spatiotemporal Reservoir Resampling for Real-Time Ray Tracing with Dynamic Direct Lighting. *ACM Trans. Graph.* 39, 4 (2020), 17 pages.

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 (2017), 12 pages.

Rudrasis Chakraborty, Yifei Xing, Minxuan Duan, and Stella X Yu. 2020. C-SURE: Shrinkage Estimator and Prototype Classifier for Complex-Valued Deep Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops.* 80–81.

Robert L Cook, Thomas Porter, and Loren Carpenter. 1984. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques.* 137–145.

Bradley Efron and Carl Morris. 1977. Stein's paradox in statistics. *Scientific American* 236, 5 (1977), 119–127.

Arthur Firmino, Jeppe Revall Frisvad, and Henrik Wann Jensen. 2022. Progressive Denoising of Monte Carlo Rendered Images. *Computer Graphics Forum* (2022). https://doi.org/10.1111/cgf.14454
Iliyan Georgiev, Jaroslav Křivánek, Tomas Davidovic, and Philipp Slusallek. 2012. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.* 31, 6 (2012), 10 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 (2019), 12 pages.
Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:1406.2661 [stat.ML]
Edwin J Green and William E Strawderman. 1991. A James-Stein type estimator for combining unbiased and possibly biased estimators. *J. Amer. Statist. Assoc.* 86, 416 (1991), 1001–1006.
Toshiya Hachisuka and Henrik Wann Jensen. 2009. Stochastic Progressive Photon Mapping. *ACM Trans. Graph.* 28, 5 (2009), 8 pages.
Jean Hausser and Korbinian Strimmer. 2009. Entropy inference and the James-Stein estimator, with application to nonlinear gene association networks. *Journal of Machine Learning Research* 10, 7 (2009).
Yuchi Huo and Sung-eui Yoon. 2021. A survey on deep learning-based Monte Carlo denoising. *Computational Visual Media* 7, 2 (2021), 169–185.
Wenzel Jakob. 2010. Mitsuba renderer. http://www.mitsuba-renderer.org.
James T Kajiya. 1986. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 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 (2015), 12 pages.
Markus Kettunen, Erik Härkönen, and Jaakko Lehtinen. 2019. Deep Convolutional Reconstruction for Gradient-Domain Rendering. *ACM Trans. Graph.* 38, 4 (2019), 12 pages.
Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
Eric P. Lafortune and Yves D. Willems. 1993. Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*. Alvor, Portugal, 145–153.
Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012. SURE-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph.* 31, 6 (2012), 9 pages.
Zehui Lin, Sheng Li, Xinlu Zeng, Congyi Zhang, Jinzhu Jia, Guoping Wang, and Dinesh Manocha. 2020. CPPM: chi-squared progressive photon mapping. *ACM Trans. Graph.* 39, 6 (2020), 12 pages.
Jonathan H Manton, Vikram Krishnamurthy, and H Vincent Poor. 1998. James-Stein state filtering algorithms. *IEEE Transactions on Signal Processing* 46, 9 (1998), 2431–2447.
Bochang Moon, Nathan Carr, and Sung-Eui Yoon. 2014. Adaptive rendering based on weighted local regression. *ACM Trans. Graph.* 33, 5 (2014), 14 pages.
Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. 2016. Adaptive Polynomial Rendering. *ACM Trans. Graph.* 35, 4 (2016), 10 pages.
Jacob Munkberg and Jon Hasselgren. 2020. Neural denoising with layer embeddings. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 1–12.
Ryan S Overbeck, Craig Donner, and Ravi Ramamoorthi. 2009. Adaptive wavelet rendering. *ACM Trans. Graph.* 28, 5 (2009), 12 pages.
Matt Pharr. 2018. Guest Editor's Introduction: Special Issue on Production Rendering. *ACM Trans. Graph.* 37, 3 (2018), 4 pages.
Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically based rendering: From theory to implementation*. Morgan Kaufmann.
Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.
Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2011. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph.* 30, 6 (2011), 12 pages.
Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. 2012. Adaptive Rendering with Non-Local Means Filtering. *ACM Trans. Graph.* 31, 6 (2012), 11 pages.
Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. 2013. Robust denoising using feature and color information. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 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 (2012), 15 pages.
Charles Stein and Willard James. 1961. Estimation with quadratic loss. In *Proc. 4th Berkeley Symp. Mathematical Statistics Probability*, Vol. 1. 361–379.
Charles M Stein. 1981. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics* (1981), 1135–1151.
Manu Mathew Thomas, Karthik Vaidyanathan, Gabor Liktor, and Angus G. Forbes. 2020. A Reduced-Precision Network for Image Reconstruction. *ACM Trans. Graph.* 39, 6 (2020), 12 pages.
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 (2018), 15 pages.
Yue Wu, Brian Tracey, Premkumar Natarajan, and Joseph P. Noonan. 2013. James–Stein Type Center Pixel Weights for Non-Local Means Image Denoising. *IEEE Signal Processing Letters* 20, 4 (2013), 411–414.
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 (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 (2021), 13 pages.
Shaokun Zheng, Fengshi Zheng, Kun Xu, and Ling-Qi Yan. 2021. Ensemble denoising for Monte Carlo renderings. *ACM Trans. Graph.* 40, 6 (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. In *Computer graphics forum*, Vol. 34. Wiley Online Library, 667–681.

A THE MSE OF THE LOCALIZED JS COMBINER

We derive the MSE of our JS combiner $\delta(X_c, Y_c)$ as the following:

$$
\begin{aligned}
&E\|\delta(X_c, Y_c) - \Theta_c\|^2 \\
&= E\|\delta(X_c, Y_c) - X_c + X_c - \Theta_c\|^2 \\
&= E\|\delta(X_c, Y_c) - X_c\|^2 + p\sigma_c^2 + 2E\Big[(\delta(X_c, Y_c) - X_c)^T (X_c - \Theta_c)\Big] \\
&= E\|\delta(X_c, Y_c) - X_c\|^2 + p\sigma_c^2 + 2E\Big[(\delta(X_c, Y_c) - \Theta_c + \Theta_c - X_c)^T (X_c - \Theta_c)\Big] \\
&= E\left[\frac{(p-2)^2\sigma_c^4}{\|X_c - Y_c\|^2}\right] - p\sigma_c^2 + 2E\Big[\delta^T(X_c, Y_c)(X_c - \Theta_c)\Big] \\
&= E\left[\frac{(p-2)^2\sigma_c^4}{\|X_c - Y_c\|^2}\right] - p\sigma_c^2 + 2\sigma_c^2 E\left[\sum_{i\in\Omega_c} \frac{\partial\delta_i(X_c, Y_c)}{\partial x_i}\right],
\end{aligned}
$$

where the equality between the last two lines is due to the Stein's lemma [Stein 1981]. Also, the last term can be further reduced to:

$$
\begin{aligned}
E\left[\sum_{i\in\Omega_c} \frac{\partial\delta_i(X_c, Y_c)}{\partial x_i}\right] &= p - \frac{p(p-2)\sigma_c^2}{\|X_c - Y_c\|^2} + \frac{2(p-2)\sigma_c^2}{\|X_c - Y_c\|^2} \\
&= p - E\left[\frac{(p-2)^2\sigma_c^4}{\|X_c - Y_c\|^2}\right].
\end{aligned}
$$

Therefore the MSE of the combiner is compactly represented as

$$
E\|\delta(X_c, Y_c) - \Theta_c\|^2 = p\sigma_c^2 - E\left[\frac{(p-2)^2\sigma_c^4}{\|X_c - Y_c\|^2}\right].
$$

We can compute an upper bound of the error using the Jensen's inequality [Green and Strawderman 1991]:

$$
\begin{aligned}
E\|\delta(X_c, Y_c) - \Theta_c\|^2 &\le p\sigma_c^2 - \frac{(p-2)^2\sigma_c^4}{E\|X_c - Y_c\|^2} \\
&= p\sigma_c^2 - \frac{\sigma_c^4(p-2)^2}{\Lambda_c^T\Lambda_c + p\xi_c^2 + p\sigma_c^2}.
\end{aligned}
$$