Eurographics Symposium on Rendering 2016
E. Eisemann and E. Fiume
(Guest Editors)

Volume 35 (2016), Number 4

Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings

Benedikt Bitterli[1] Fabrice Rousselle[1] Bochang Moon[1] José A. Iglesias-Guitián[1]
David Adler[2] Kenny Mitchell[1,3] Wojciech Jarosz[4] Jan Novák[1]

[1]Disney Research [2]Walt Disney Animation Studios [3]Edinburgh Napier University [4]Dartmouth College

Abstract

We address the problem of denoising Monte Carlo renderings by studying existing approaches and proposing a new algorithm that yields state-of-the-art performance on a wide range of scenes. We analyze existing approaches from a theoretical and empirical point of view, relating the strengths and limitations of their corresponding components with an emphasis on production requirements. The observations of our analysis instruct the design of our new filter that offers high-quality results and stable performance. A key observation of our analysis is that using auxiliary buffers (normal, albedo, etc.) to compute the regression weights greatly improves the robustness of zero-order models, but can be detrimental to first-order models. Consequently, our filter performs a first-order regression leveraging a rich set of auxiliary buffers only when fitting the data, and, unlike recent works, considers the pixel color alone when computing the regression weights. We further improve the quality of our output by using a collaborative denoising scheme. Lastly, we introduce a general mean squared error estimator, which can handle the collaborative nature of our filter and its nonlinear weights, to automatically set the bandwidth of our regression kernel.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing; I.4.3 [Computer Graphics]: Enhancement—Filtering

1. Introduction

In the last few years, rendering in movie production has experienced a dramatic shift [KFF*15], as evidenced by recent developments in production renderers like Renderman, Arnold, Hyperion, and Manuka. Methods based on rasterization and micropolygon rendering—the dominant rendering techniques for decades—are being displaced by variants of path tracing. The potential benefits of this transition are particularly appealing for production, not only due to the increased flexibility and generality of the (quasi) Monte Carlo (MC) integration, but also due to native support for progressive rendering that allows for quick turnaround for rough previews while easily scaling to final quality with more render time.

Unfortunately, MC rendering comes with the drawback that rendered images are noisy, and, moreover, the level of noise decreases only with the square-root of the number of samples, making synthesis of perceptually noise-free images prohibitively costly for movie production. This has made image-based denoising methods a virtual necessity in production, and images are only rendered up until the remaining noise can be eliminated safely (i.e. without introducing artifacts) using less expensive image-based denoising.

Graphics researchers have developed and adapted many denoising techniques from the signal-processing community; see the comprehensive survey by Zwicker and colleagues [ZJL*15]. While these approaches have gained considerable traction in recent years, based on our analysis, none of them seems capable of outperforming its rivals at all fronts.

An ideal, production-ready denoising algorithm should posses the following essential characteristics:

Effectiveness. Produce significant noise reduction without introducing disturbing artifacts or overblurring.

Predictability. Perform consistently well without requiring per-shot parameter tuning or other manual intervention and degrade gracefully when assumptions are not met.

Temporal stability. Produce flicker-free results across frames rendered with varying random samples, camera positions, object positions, or other scene changes.

In addition, the following are required if denoising is to integrate into a production environment:

Ease of adoption. The filter should be easy to integrate into the production pipeline. Generating the filter's inputs should not require renderer changes which are difficult to implement or expensive to run.

Speed. The filter should run significantly faster than the time to render equivalent additional samples. Ideally, filtering would be much faster than a typical render so that it can be performed multiple times during a render for adaptive sampling and for viewing denoised progressive rendering.

Memory. Use significantly less memory than the renderer itself to minimize memory cost of adaptive sampling.

In this paper, we pick the small subset of existing algorithms (Table 1) that represent the state of the art with respect to the aforementioned criteria and compare their performance on a diverse set of test scenes. Based on our findings, we then propose a new practical algorithm that combines individual elements of the algorithms in a novel way, yielding comparable or higher-quality results than attainable before.

Our *Nonlinearly weighted First-Order Regression* (NFOR) leverages the correlation between auxiliary buffers and ground truth images. We additionally incorporate a patch-based weighting kernel into our regression framework to robustly preserve fine details even when features do not provide enough correlation.

2. Previous Work

In the following, we briefly review image-space denoising algorithms that are relevant to our work; a more thorough theoretical and comparative analysis follows in Sections 3 and 4. We focus on *a posteriori* techniques that rely on empirical (and often statistical) analysis of sampled data without requiring prior knowledge of how the samples were generated. A thorough review of the alternative—*a priori* methods—can be found in a recent survey [ZJL*15].

Algorithms for denoising MC renderings often build upon techniques developed for processing natural images, such as the bilateral [TM98, PKTD08], cross bilateral [ED04, PSA*04], and NL-Means [BCM05] filters, or wavelet thresholding [Don95]. Examples include a cross bilateral filter that utilizes a Gaussian blurred image as an edge-stopping function [XP05], non-local means filters which employs a dual-buffer variance estimation [RKZ12], or a histogram-based distance metric [DMB*14], and soft thresholding methods driven by the estimated local variance [ODR09, KS13]. A common adaptation in denoising rendered images is to fully utilize auxiliary buffers as edge-stopping functions such as normals, albedos, depths [McC99, DSHL10, LWC12, SD12, KBS15], virtual flash image [MJL*13], or ambient occlusion map [RMZ13, KBS15].

As we will show in Section 3, all the aforementioned techniques can be formulated as zero-order regressions. The theoretically more powerful first-order regression has also been considered in previous work, ranging from edge-preserving smoothing [HST10], denoising MC renderings using constant [BEM11] or per-pixel regression weights [MCY14], sparse reconstructions [MIGYM15], or combined with a Poisson reconstruction using sampled gradients [MVZ16]. Recently, higher-order regression was also explored [MMMG16]. Our algorithm is based on a first-order regression, but uses the NL-Means filter to shape the regression kernel.

Different approaches have been used for estimating the optimal bandwidth of the regression kernel, all of which strive to minimize the mean squared error (MSE), i.e. the sum of variance and squared bias. While directly quantifying the variance is trivial for filters with constant [RKZ11] or noise-free [MCY14] weights, which also allow reasoning about the bias [MCY14], estimating the MSE of a non-linear (e.g. bilateral or NL-Means) filter is challenging as the weights are both noisy and correlated with the input. Following the work of Van de Ville and Kocher [VdVK09], both Li et al. [LWC12] and Rousselle et al. [RMZ13] estimate the MSE using Stein's Unbiased Risk Estimate [Ste81] (SURE), a powerful technique that can be used to estimate the MSE of non-linear filters. Kalantari et al. [KBS15] proposed an interesting alternative to conventional bandwidth estimation; they used a neural network trained on a diverse set of scenes to set the parameters of cross bilateral and cross NL-Means filters.

3. Theoretical Background

Image-space denoising estimates the value of a pixel p as a weighted average of its neighborhood:

$$\hat{c}_p = \frac{1}{C_p} \sum_{q \in \mathcal{N}_p} c_q \, w(p,q), \tag{1}$$

where c_p and $\hat{c}_p$ are the (noisy) input and filtered color values of a pixel p, $\mathcal{N}_p$ is the (typically square) neighborhood centered on p, $w(p,q)$ is the weight of the contribution of pixel q to the estimate, and $C_p = \sum_{q \in \mathcal{N}_p} w(p,q)$ is a normalization factor. We now describe existing approaches for setting $w(p,q)$ that have been used for denoising MC renderings, in order of increasing complexity.

3.1. Zero-order Models

Bilateral Filtering. The bilateral filter [TM98, Ela02] is a popular edge-preserving smoothing filter that leverages a conventional filtering kernel (typically a Gaussian), and applies it to both the *spatial* and *range* (i.e. color, also referred to as *radiometric*) components of the image:

$$w_{\mathrm{bf}}(p,q) = \exp\left(\frac{-\|p-q\|^2}{2\sigma_s^2}\right) \exp\left(\frac{-\|c_p - c_q\|^2}{2\sigma_r^2}\right), \tag{2}$$

where σ_s^2 and σ_r^2 are parameters to adjust distance across image and color space, respectively. For example, the above weight will be small if p and q are far from each other spatially or have different colors; the latter ensures that edges are well preserved in the filtered image. A key issue with the bilateral filter, however, is that the range-distance estimation can be sensitive to noise. To make it more stable, one can either compute the range distances on a low-pass-filtered version of the image [XP05], or average them over small neighborhoods around pixels, as described next.

Non-Local Means. The non-local means (NL-Means) filter [BCM05] is a generalization of the bilateral filter where the range distance is computed using small patches $\mathcal{P}(\cdot)$ around the pixels:

$$w_{\mathrm{nlm}}(p,q) = \exp\left(\frac{-\|p-q\|^2}{2\sigma_s^2}\right) \exp\left(\frac{-\|\mathcal{P}_p - \mathcal{P}_q\|^2}{k^2 2\sigma_r^2}\right), \tag{3}$$

where k is a user parameter controlling the strength of the filter. The squared patch-based range distance is defined as:

$$\|\mathcal{P}_p - \mathcal{P}_q\|^2 = \max\left(0, \frac{1}{|\mathcal{P}|}\sum_{n\in\mathcal{P}_0} d(p+n, q+n)\right), \quad (4)$$

where $\mathcal{P}_p$ is a square patch of size $|\mathcal{P}|$ (usually 7×7 pixels) centered on p, and $\mathcal{P}_0$ enumerates the offsets to the pixels within a patch. The range distance $d(p,q)$ between two pixels is estimated using the squared difference of their respective colors, i.e. $d(p,q) = \|c_p - c_q\|^2$. It is worth noting that since the colors are noisy, the estimate tends to overestimate the true squared distance; this is referred to as the noise bias. Buades et al. [BCM05] propose to counter the overestimation by subtracting twice the variance of the input colors σ_r^2:

$$d(p,q) = \|c_p - c_q\|^2 - 2\sigma_r^2. \quad (5)$$

When the pixel variance changes spatially, which is often the case in MC rendering, the previous distance estimate can be further improved by considering the per-pixel color variance as proposed by Rousselle at al. [RKZ12]:

$$d(p,q) = \frac{\|c_p - c_q\|^2 - (\mathrm{Var}_p + \mathrm{Var}_{p,q})}{\varepsilon + \frac{\mathrm{Var}_p + \mathrm{Var}_q}{2\sigma_r^2}}, \quad (6)$$

where Var_p is the variance at p, $\mathrm{Var}_{p,q} = \min(\mathrm{Var}_p, \mathrm{Var}_q)$, and ε is a small offset to prevent dividing by zero.

Joint Filtering. If additional per-pixel information is available, the quality of bilateral or NL-Means filtering can be further improved by factoring the information into the weight—the so-called joint filtering:

$$w_{\mathrm{jbf}}(p,q) = w_{\mathrm{bf}}(p,q)\, w_{\mathrm{aux}}(p,q), \quad (7)$$

$$w_{\mathrm{jnlm}}(p,q) = w_{\mathrm{nlm}}(p,q)\, w_{\mathrm{aux}}(p,q), \quad (8)$$

where $w_{\mathrm{aux}}(p,q)$ is a weight derived from the additional information. Given a vector of k auxiliary *feature* buffers, $\mathbf{f} = (f_1, \ldots, f_k)$, we can compute the weight as:

$$w_{\mathrm{aux}}(p,q) = \prod_{i=1}^{k} \exp(-d_{f,i}(p,q)), \quad (9)$$

where:

$$d_{f,i}(p,q) = \frac{\|f_{i,p} - f_{i,q}\|^2}{2\sigma_{i,p}^2}, \quad (10)$$

and σ_i^2 is the bandwidth of feature i. Joint filtering is typical for denoising MC renderings where feature buffers (e.g. normal, albedo, depth) can be obtained inexpensively as a byproduct of rendering the image.

Zero Order Regression. It can be shown that all the aforementioned filters are solutions to a weighted, local, *zero-order* regression, i.e. they model the pixel neighborhood as a constant function. Consider the following general formulation of zero-order regression:

$$\hat{c}_p = \arg\min_{\hat{c}_p} \sum_{q\in\mathcal{N}_p} (c_q - \hat{c}_p)^2\, \tilde{w}_{\mathbf{x}}(p,q) \quad (11)$$

Table 1: *Acronyms for the previous works considered in this analysis, and corresponding regression order.*

Tag	Algorithm	Order	Reference
LBF	Learning-Based filter	0	[KBS15]
NLM	Non-Local Means	0	[RKZ12]
RDFC	Robust Denoising – Features & Color	0	[RMZ13]
RHF	Ray Histogram Fusion	0	[DMB*14]
WLR	Weighted Linear Regression	1	[MCY14]

where $\mathbf{x} = (x_1, \ldots, x_D)$ is a D-dimensional feature vector storing the spatial, radiometric, and auxiliary features; and $\tilde{w}_{\mathbf{x}}$ is the weight calculated from $\mathbf{x}$. Takeda et al. [TFM07, Eq. 45], show that if the feature space consists of pixel coordinates and pixel colors, i.e. $\tilde{w}_{\mathbf{x}}(p,q) = w_{\mathrm{bf}}(p,q)$, then the solution to Equation (11) is the bilateral filter. By extension, the joint bilateral and NL-Means filters are also solutions of weighted, zero-order regressions, but using a higher dimensional feature vector $\mathbf{x}$ to set the weights. In the case of the joint NL-Means filter, the feature vector $\mathbf{x}$ contains the coordinates of pixel p, the colors of the pixels in the patch $\mathcal{P}_p$, and the auxiliary feature buffers.

3.2. First-order Models

A natural extension to zero-order models is to consider a *first-order* regression:

$$[\hat{c}_p, \nabla\hat{c}_p] = \arg\min_{\hat{c}(p),\nabla\hat{c}(p)} \sum_{q\in\mathcal{N}_p} (c_q - \hat{c}_p - \nabla\hat{c}_p\cdot(\mathbf{y}_q - \mathbf{y}_p))^2\, \tilde{w}_{\mathbf{x}}(p,q) \quad (12)$$

where $\mathbf{y}$ is, similarly to $\mathbf{x}$, a 1D vector corresponding to a point in the high dimensional feature space. While we may have $\mathbf{y} = \mathbf{x}$, it is not a requirement, that is, we may choose to use a different set of features to define the regression weights, than to perform the regression itself. We will detail the algorithms which use first-order regression [BEM11, MCY14] in the next section.

4. Comparative Analysis

In the previous section, we showed how various methods for denoising MC renderings can be formulated as either a zero- or first-order regression. Besides the order, the main differentiating characteristic is how these methods compute the regression weights $\tilde{w}_{\mathbf{x}}$. There are three elements involved in this computation: the kernel function (uniform, Gaussian, Epanechnikov, etc.), the pixel features considered (color, normal, albedo, etc.), and the kernel bandwidth along each dimension of the feature space.

In this section we cover techniques used in recent works to increase the robustness of the regression weights computation. For the purpose of our comparative analysis, we consider recent methods, listed in Table 1, that encompass the current state of the art for denoising Monte Carlo renderings. We use a test bench composed of 21 scenes covering a wide range of light transport effects, and provide full resolution results and error metrics (MSE, relative MSE, PSNR, and SSIM) for all methods in our supplemental material, along with a web-based interactive viewer that allows for close inspection of the results.

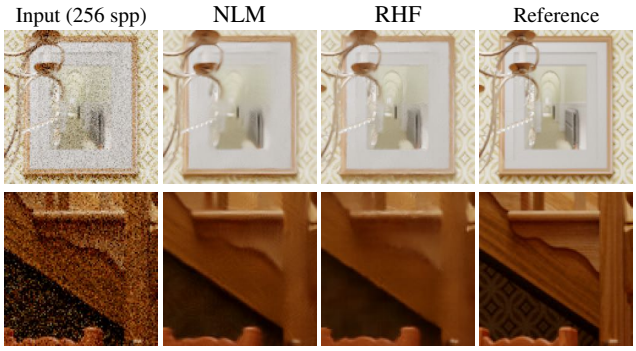

Figure 1: *Results of NLM and RHF on the* STAIRCASE *scene. RHF robustly preserves the details of the illustration in the top row, despite the low signal-to-noise ratio. NLM, however, better preserves the details in the second row, since the histogram is highly skewed in this region, with 95% or more of the samples contained in the first bin.*

We first present the recently proposed histogram-based distance, then the use of auxiliary buffers in joint filtering schemes, and the approaches used to automatically set kernel bandwidths. We will then consider the specific case of first-order regressions, and finally conclude with some observations that will instruct the design of our proposed filter.

4.1. Mean- vs. Histogram-based Distance

Denoising techniques generally define a pixel feature as the mean of all the corresponding samples. For instance, the pixel color is the sum of all radiance samples, weighted by the pixel filter function. RHF builds on the observation that considering only the sample mean and variance implicitly discards a wealth of information, and instead considers the actual sample distributions. Since storing individual samples is prohibitively expensive, the algorithm bins them in per-pixel histograms, and the distance between two pixels is computed as the chi-square distance of their histograms. This amounts to replacing the 3d feature space of the color mean, with the 60d space of the color histogram (using 20 bins for the histogram), with the hope that it is easier to discriminate neighbors in this high-dimensional space.

Discussion. In our experiments, we have found that the histogram-based distance can indeed allow for discriminating pixels, even when the color-mean difference is within the noise range. Also, an outlier sample can greatly affect the sample mean, but can only offset the sample count of a histogram bin by one, making the histogram-based distance more robust to fireflies. In practice, however, these improvements are not consistent; we illustrate this in Figure 1 by comparing the outputs of NLM, which considers the sample mean and variance, to RHF. The reliability of the histogram-based metric largely depends on having a good binning strategy, which is not known a priori, and the metric degenerates if all samples are concentrated in one or two bins.

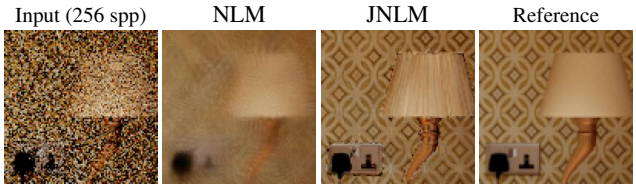

Figure 2: *Comparison of NLM to a Joint NL-Means (JNLM) filter, which also considers auxiliary buffers when computing the regression weights. NLM produces a smooth output, but only captures the lower frequencies of the signal. JNLM better preserves the details, such as the wall texture captured in the albedo buffer, but has banding artifacts on the lamp shade, aligned with the isolines of the normal buffer. The figure shows a crop from the* STAIRCASE *scene.*

4.2. Joint Filtering

With the notable exception of RHF, all recent denoising methods leverage *auxiliary buffers* to define the regression weights. Auxiliary buffers allow for a more robust discrimination since they are generally significantly less noisy than the color buffer. As illustrated in Figure 2 using a Joint NL-Means filter (JNLM), zero-order methods [SD12,LWC12,RMZ13,KBS15] greatly benefit from this improved discrimination.

Discussion. The main drawback of using auxiliary buffers to define the regression weights lies in the ill-defined relation between the feature distance and the filtered output MSE, which makes it difficult to set the feature bandwidths. Consider the case in Figure 2: the tight kernel bandwidth robustly preserves the geometric edges, but leaves distracting banding artifacts on the lamp shade. In the next section, we present common approaches for estimating appropriate bandwidths on a per-pixel basis.

4.3. Bandwidth Estimation

The kernel bandwidth has to balance two conflicting goals: reducing the variance and preventing bias (over-blurring). A larger bandwidth increases the effective size of the neighborhood $\mathcal{N}_p$, which reduces the variance, but tends to increase the bias. The kernel bandwidth must therefore strike a good balance between residual variance and bias, which amounts to minimizing the MSE.

As outlined in the introduction, requiring the user to manually tune the bandwidth is not acceptable in production environments. Furthermore, a global bandwidth is likely to be sub-optimal locally; see the lamp in Figure 2 for an illustration of resulting artifacts. Thus, most methods automatically adjust the bandwidth locally, using one of the following two approaches: 1) a selection-based estimate, where the filter is run using different predefined bandwidths, and the one resulting in the lowest estimated MSE is used on a per-pixel basis [RKZ11,RKZ12,RMZ13,BEEM15]; 2) a direct estimate, using either statistical analysis [SD12], or machine learning [KBS15]. Moon et al. [MCY14] combine both approaches: they factor the feature bandwidth as the product of a shared bandwidth, h, and a per-feature bandwidth, $\mathbf{b}_i$, and use a selection-based estimate for the shared bandwidth, and a direct estimate for the per-feature bandwidth.

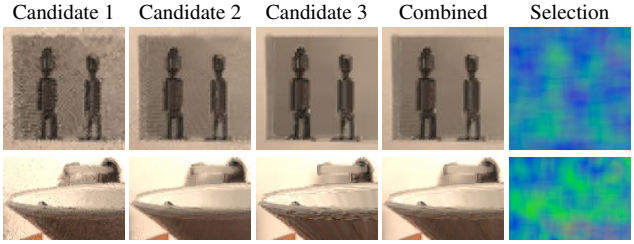

Figure 3: *Bandwidth estimation of RDFC in the* BATHROOM *scene. The first two candidates (red and green in the selection map) better preserve details in the color buffer, in particular shadows, while the third candidate (blue) better preserves the geometric edges captured in the auxiliary buffers. The combined reconstruction offers a reasonable compromise on a per-pixel basis.*

Figure 4: *Comparison of RDFC and WLR on the* DINING ROOM *scene. The Joint NL-Means filter used by RDFC can only combine pixels with similar values, which leaves residual noise on the curved teapot surface, where the normals change rapidly. In contrast, WLR uses a first-order regression that better models the data, leading to smoother results.*

Discussion. While setting the bandwidth automatically on per-pixel level is crucial for good performance, as we illustrate in Figure 3, the high dimensionality of the feature space makes the problem challenging and error-prone, and all existing methods have some limitations. For instance, LBF uses a neural network trained in an offline process to harness the potential of their regression kernel, but their solution does not extend to adaptive sampling. The selection algorithm of Bauszat et al. [BEEM15] can be applied to any filter, but their approach also cannot handle adaptive sampling. The main nuisance of RDFC, which employs SURE [Ste81] to estimate the MSE, is the requirement of knowing the derivative of the filter. WLR directly estimates the output variance and bias, but their approach assumes a filter with noise-free regression weights.

4.4. Zero- vs. First-order Regression

First-order regression uses a linear approximation instead of a constant to represent the pixel neighborhood, solving for both the center pixel value and its first partial derivatives. In practice, this additional degree of freedom is significant, since we have many auxiliary buffers that can be linearly combined, allowing for a far richer model of the pixel neighborhood than a single constant.

Discussion. For zero-order regressions, the kernel bandwidth must sometimes be drastically reduced to enforce the uniformity assumption of the model. A first-order regression is less constrained, since it directly leverages the correlation between the auxiliary buffers and the color buffer, which allows for a better use of the neighborhood data. We illustrate this in Figure 4 by filtering the curved surface of a teapot using RDFC (zero-order) and WLR (first-order). The improved model of first-order regressions can offer a reasonable prediction of the whole pixel neighborhood, a fact that was exploited by Moon et al. [MIGYM15] to perform a sparse reconstruction in order to achieve interactive filtering performance.

4.5. Observations and Proposed Method

From a theoretical point of view, we would have expected WLR to be a clear winner in these tests, since it is the only method to perform a first-order regression. However, filters performing a zero-order regression often produced better results, both subjectively and numerically. We believe there are three key reasons for this result. First, RDFC and LBF prefilter the auxiliary buffers to ensure they are nearly noise-free, whereas WLR relies on a Truncated Singular Value Decomposition to handle noisy features. This truncation is less robust in practice, since it can only remove noisy features; in contrast, prefiltering can extract useful information even when the features are noisy.

Second, WLR uses only the auxiliary features to shape the regression kernel and does not utilize the color buffer, and can provide suboptimal results when the auxiliary features have low correlation with the input. In contrast, the NL-Means kernel used by RDFC provides reasonable results even when no useful auxiliary features are present.

Third, WLR needs to estimate bandwidths for each feature, and does so on a continuous scale, a challenging problem given the dimensionality of the reduced feature space. In contrast, RDFC only considers three bandwidth configurations known to give generally useful results, and solves the simpler problem of picking between these three configurations. While this approach has less potential, its restricted scope makes it more robust.

Solving the first problem is trivial, since WLR can use prefiltered features. We show in Figure 5 that this can greatly improve the filter behavior. For the second problem, we propose to use the weights of a standard NL-Means filter as the regression kernel. The NL-Means kernel provides two advantages: 1) it provides a reasonable base when the auxiliary buffers do not provide any useful information; 2) it has only one bandwidth parameter to set. While the NLM result in Figure 2 suggests that the NL-Means kernel is a poor choice, we show in Figure 6 that, when paired with a first-order model, it can produce results of very high quality. Using the NL-Means kernel also allows us to use a similar selection-based bandwidth estimation as RDFC.

Based on these observations, our proposed method uses:

- feature prefiltering,
- a first order regression,
- NL-Means to compute the regression weights,
- collaborative filtering,
- and a selection-based bandwidth estimation.

Figure 5: *The WLR output for the* MUSEUM *scene, without and with prefiltering of the auxiliary buffers. With prefiltering, WLR can better leverage the feature buffers.*

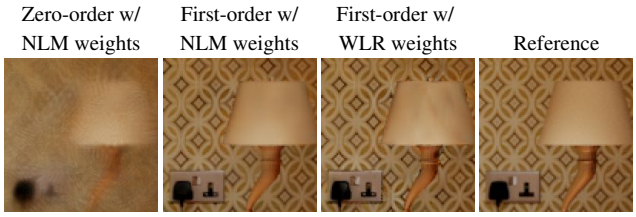

Figure 6: *Using the same NL-Means regression kernel as in Figure 2 with zero- and first-order regressions. Even though this kernel gives a poor result in a zero-order regression, it produces a high-quality result when paired with a first-order regression, comparable to the WLR output, despite using a much simpler weighting scheme.*

5. Algorithm Overview

The aim of our algorithm is to achieve the ideal filter properties outlined in the introduction. To this end, we build upon the conclusions of our comparative analysis: our filter employs a first-order model, enriched by the commonly used set of auxiliary buffers (normal, albedo, depth, and visibility), and a nonlinear regression kernel, namely the NL-Means filter. The first-order model allows us to better exploit any existing correlation between the auxiliary buffers and the pixel color, while the NL-Means kernel offers a robust baseline in the absence of such correlation. Following previous work [RKZ12], we evenly split all samples between two statistically independent image buffers, which we utilize in a cross-filtering scheme to minimize residual filtering artifacts, and to perform a simple but effective MSE estimation, needed to automatically set the bandwidth of our regression kernel.

In the following sections, we describe the main steps of our filter in order: Section 5.1, auxiliary buffers prefiltering; Section 5.2, collaborative first-order regression; Section 5.3, bandwidth selection; Section 5.4, second filtering pass. We also provide the complete pseudocode of our filter in the supplemental material, including all parameter settings we used to produce our results.

5.1. Auxiliary Buffer Prefiltering

Our regression model assumes that the input auxiliary buffers are noise-free, which is generally not the case. In particular, the visibility buffer is often quite noisy; see Figure 7. As proposed by Rousselle et al. [RMZ13], we therefore prefilter all auxiliary buffers using an NL-Means filter. Because we use a first-order model, any residual filtering artifacts in the feature buffers propagate directly to the filtered output, and we have found it beneficial to employ a two-step filtering scheme to minimize residual noise.

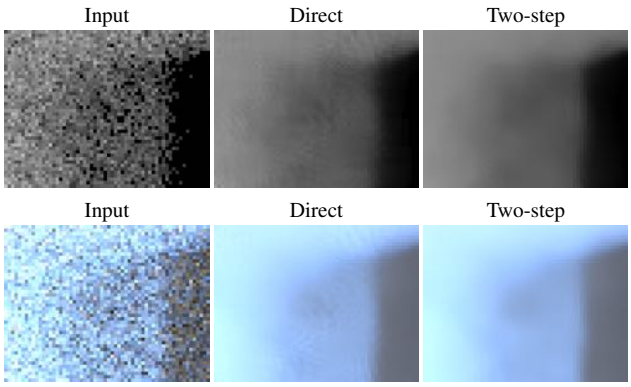

Figure 7: *Prefiltering of the visibility buffer (here a shadow on a wall of the* LIVING ROOM *scene) using a single-step NL-Means filter, and our two-step cross-filtering. Our scheme minimizes the amount of residual noise artifacts, which are otherwise directly propagated to the regression output.*

Our prefiltering scheme leverages the two statistically independent half buffers in a cross-filtering scheme, which was proposed in an earlier work by Rousselle et al. [RKZ12]. We use the NL-Means weights computed from one half buffer to filter the other and vice-versa, which has the advantage of decorrelating the noise of the weights from the noise of the data. The squared difference of the two filtered half buffers gives an estimate of their residual variance, which we use in a second filtering pass to further remove residual noise. In Figure 7, we compare the outputs of single-pass filtering to our two-step cross-filtering, and show the improvement on the resulting regression output.

5.2. Collaborative First-order Regression

Our filter builds upon a first-order model, using the auxiliary buffer data to predict the denoised color output. To this end, we consider the following 10-dimensional feature space: pixel coordinates (2D), normal (3D), albedo (3D), visibility (1D), and depth (1D). We can further enrich this set of features as needed. For instance, LBF uses a secondary albedo buffer that encodes the albedo of the second ray intersection, and when running our filter on the LBF input data, we also include this secondary albedo in our feature set.

Feature Cross-Filtering. Our prefiltering of the auxiliary buffers is effective at addressing the mid to high frequencies of the noise, but not its low frequencies. Since the noise in the feature buffers is generally correlated with the noise in the color buffer, the regression will tend to reconstruct the low frequency noise of the color buffer in the filtered output. To address this issue, we again take advantage of the two half buffers. We filter the two half buffers separately, and use the feature vector of the first buffer when fitting the second buffer and vice-versa. This decorrelates the noise in the features from the noise in the color buffer during the regression.

First-Order Regression. Having established the feature space of our regression model, we now turn to the computation of the regression weights, $\tilde{w}_{\mathbf{x}}(p,q)$, of Equation 12. Since the auxiliary buffers are already accounted for in our regression model, we opted to compute the regression weights only on the pixel color. In this sense, the regression model and weights are complementary: the color-based regression weights can capture elements not encoded in our feature set (e.g. indirect shadows, caustics, etc.), while the model preserves the high-frequency elements captured by our feature set (e.g. texture details, high-frequency normal map, etc.). Using the NL-Means kernel for our regression weights also has the convenient advantage of having only a single bandwidth parameter, k in Equation 3, to set.

In contrast, WLR considers the auxiliary buffers to shape its regression kernel, and must then estimate appropriate bandwidths for different types of features on their reduced feature space. In addition, unlike our method, WLR does not consider the color information when computing the regression weights, and therefore falls back on a simple isotropic kernel when the features are uncorrelated with the color.

Collaborative Filtering. It is worth noting that through the use of a first-order regression, the entire filter window is denoised at once, not only the center pixel. In other words, each pixel in the image is denoised multiple times, once for each filter window it appears in. We utilize this fact in a collaborative filtering scheme, and compute the filtered output as a weighted average of denoised filter windows, each weighted by its regression kernel. This further reduces the variance of the output and tends to give smoother and more homogeneous results. Moon et al. [MIGYM15] previously used collaborative filtering in a sparse reconstruction scheme to decrease the computational cost of denoising. However, they use an unweighted average of filter windows; in contrast, we can leverage the NL-Means regression kernel to robustly average them. Because of their sparse reconstruction and variable window size, filter windows in Moon et al.'s method also overlap much less than in our method, which reconstructs densely and always uses the full filter window size. As such, their collaborative scheme targets performance, whereas ours targets higher quality output.

5.3. Bandwidth Selection

In our experiments, we found using NL-Means regression weights to be surprisingly robust, and that setting its bandwidth parameter to $k = 0.5$ gives reasonable results on a wide range of scenes. In many cases however, a larger bandwidth is preferable. Consequently, we perform a selection-based bandwidth estimation (see Section 4.3) using the parameter set $k = \{0.5, 1.0\}$.

An estimate of each candidate bandwidth MSE is needed to perform the selection. We considered using SURE [Ste81], but this approach requires the filter derivative to be known, which is challenging to do for our collaborative filter. The direct variance and bias estimation used by WLR is also not an option in our case, since its variance estimate assumes the filter weights to be noise-free, which is not the case for our method. Instead we propose a more general scheme leveraging our two half buffers.

Consider the squared difference between the filtered output, F, and the noisy input, C, minus the variance of the input. Its expected value is

$$E[(C-F)^2 - \mathrm{Var}_C] = \mathrm{Bias}_F^2 + \mathrm{Var}_F - 2\,\mathrm{Cov}_{C,F}\,.$$

If the input and the filtered result are uncorrelated, then we can trivially use this result to create an unbiased estimator of the true MSE. This is not the case for our filter, but we can again leverage the two uncorrelated half buffers to sidestep this problem: We compute the squared difference of the filtered first half buffer to the second input half buffer and vice-versa, and subtract the variance of the input. If the two filtered half buffers are uncorrelated, this gives an unbiased estimator of the MSEs of each half buffer with

$$\mathrm{MSE}_{F_1} = (F_1 - C_2)^2 - \mathrm{Var}_{C_2},$$
$$\mathrm{MSE}_{F_2} = (F_2 - C_1)^2 - \mathrm{Var}_{C_1},$$

where $\mathrm{Var}_{C_1} = \mathrm{Var}_{C_2} = 2\,\mathrm{Var}_C$, where Var_C is the sample mean variance of all color samples. After the MSE estimation, we average the two filtered half buffers to obtain $F = (F_1 + F_2)/2$. The expected bias of F remains unaffected, but its variance is halved; consequently, we estimate the MSE of the averaged half buffers,

$$\mathrm{MSE}_F = \frac{\mathrm{MSE}_{F_1} + \mathrm{MSE}_{F_2}}{2} - \mathrm{Var}_F\,.$$

This estimator is unbiased under the assumption that our filtered half buffers are statistically independent. Our cross-filtering scheme introduces some correlation, but in practice, this correlation is sufficiently weak to produce a useful MSE estimate. Similar to Rousselle et al. [RMZ13], we then filter the estimated MSE and use it to generate selection maps for each filter bandwidth. We compare our filtered MSE and selection map to ground truth in Figure 8.

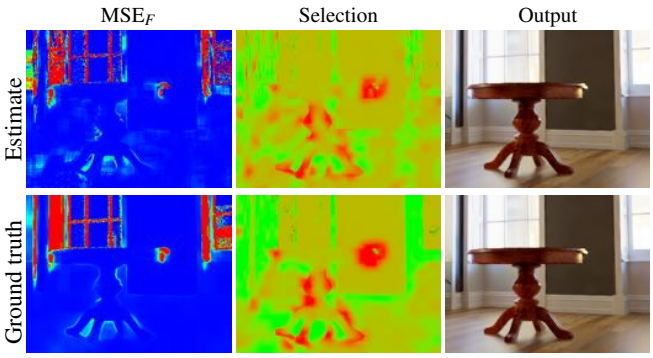

Figure 8: *Comparison of our* MSE *estimate for the first candidate bandwidth ($k = 0.5$) and the ground truth* MSE *in the* HORSE ROOM *scene. The middle and right columns show the corresponding selection maps (red where $k = 0.5$ is used, green where $k = 1.0$ is used) and filtered outputs, respectively.*

5.4. Second Regression Pass

Similarly to Rousselle et al. [RMZ13], we perform a second filtering pass on the color buffer to remove residual noise artifacts. We need the variance of the first pass output to compute our regression weights, which we estimate as the variance across our filtered half buffers. We do not perform cross-filtering for the second pass however, but directly refilter the average of the filtered half buffers.

6. Implementation Details

We implemented our algorithm on the CPU using C++11 and the Eigen 3 library for efficient matrix operations.

Feature Normalization. In order to improve numerical conditioning, we locally normalize the auxiliary features to occupy the same range, similarly to Moon et al. [MCY14]. Specifically, we offset and scale each feature within our filtering window such that it occupies the $[-1,1]$ interval.

Space-Time Filtering. Residual low-frequency noise in the filtered output causes disturbing flickering artifacts in animations. Similarly to WLR, we mitigate these artifacts by exploiting temporal coherence. We expand our regression window to the spatio-temporal domain by including noisy pixels and features from the two previous and the two next frames. We include the frame index as an additional auxiliary vector, and compute the regression weights using a spatio-temporal NL-Means kernel [BCM07, RKZ12]. The rest of our algorithm remains the same. We include a video demonstrating the temporal stability of our spatio-temporal filter in our supplementary material.

7. Results

We use a test bench composed of 21 scenes (see Figure 9) to evaluate our method, and compare its output to four state-of-the-art methods: RDFC [RMZ13], RHF [DMB*14], WLR [MCY14], and LBF [KBS15]. All results were obtained with a fixed set of parameters for all denoisers. Our set of scenes feature volumes, hair and fur, interiors lit by an environment map, as well as diffuse, glossy, and specular reflections. Full resolution images at 16, 64, 256, and 1024 spp can be found in the supplemental material. Additionally, we provide denoised outputs using the standard NL-Means filter, as well as a multiscale NL-Means variant using the hierarchical scheme used by Delbracio et al. [DMB*14] in the RHF algorithm. Lastly, our supplemental material also provides a comparison with the RenderMan denoiser, which is used in production.

Some of these previous methods (RDFC, WLR) are integrated into the PBRT code base, which we did not use for rendering. Consequently, we modified the public implementation of these methods to inject our own data in their pipeline. For RHF, we output the required color histograms from our renderer and use the public implementation to process it. For LBF, we similarly modified the public implementation to inject our own data. We note however that our auxiliary buffers differ from those used by Kalantari et al. [KBS15]: we use depth instead of world positions, and we do not export a secondary albedo, but instead have a buffer that stores the albedo of the first non-specular surface along the path. Since LBF employs a trained neural network to set its filter parameters, it is not clear how it can generalize to our data set. For a fair comparison to LBF, we rendered the SAN MIGUEL scene using the LBF code base, and passed its input data (including the secondary albedo) to the other denoisers. The SAN MIGUEL scene uses low discrepancy sampling, while all the other scenes were rendered using independent samples. For low discrepancy samples, we use the variance across the pairs of half buffers as an estimate of the sample mean variance. To clearly verify denoising performance of all

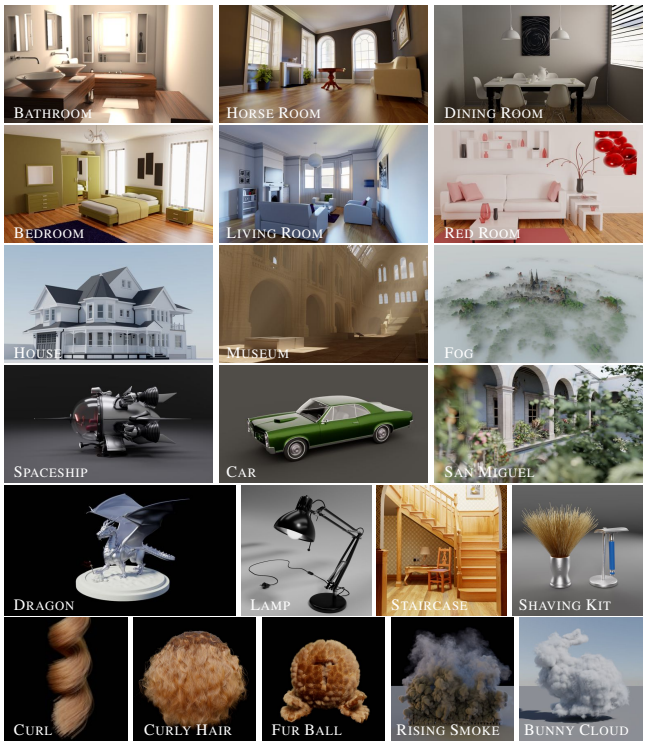

Figure 9: *Our test bench consists of 21 scenes used to collect metrics for NL-Means [BCM05], NL-Means with multiscale, RDFC [RMZ13], RHF [DMB*14], WLR [MCY14], LBF [KBS15], and for our approach. Please refer to the supplementary material for a complete evaluation.*

tested methods, we have used the same input color and auxiliary buffers for all denoisers. Since the WLR method does not perform any prefiltering of the auxiliary buffers, we ran it twice, once with the noisy buffers (WLR), and once with the prefiltered buffers produced by our filter (WLR-PF). Results with both filtered and unfiltered inputs are provided in our supplemental material.

We used four different metrics to evaluate the output of the various denoisers: MSE, relative MSE, SSIM and PSNR, and provide the relative MSE and SSIM heatmaps for all results in our supplemental material. In Figure 10, we show the overall statistics of these denoisers on our test scenes. The thin blue lines correspond to the metric on a given scene, while the dots indicate the average metric value over all scenes. The error bars show the standard deviation of each method to the best performing technique for each scene. Therefore, a large error bar suggests that a method output quality varies greatly from scene to scene. In the case of RHF, the large error bars are due partly to the fact that this method relies on hand-tuning of its parameter, whereas we used a constant setting of $\kappa = 0.7$, which we found to be a good compromise over all scenes. Our filter exhibits both the best average and the smallest error bars across all metrics, indicating that it is consistently on par with the best alternative approach, and often substantially better.

In Figure 11, we show results from all denoisers on a selection of our test scenes rendered at 256 samples per pixel (BATHROOM,

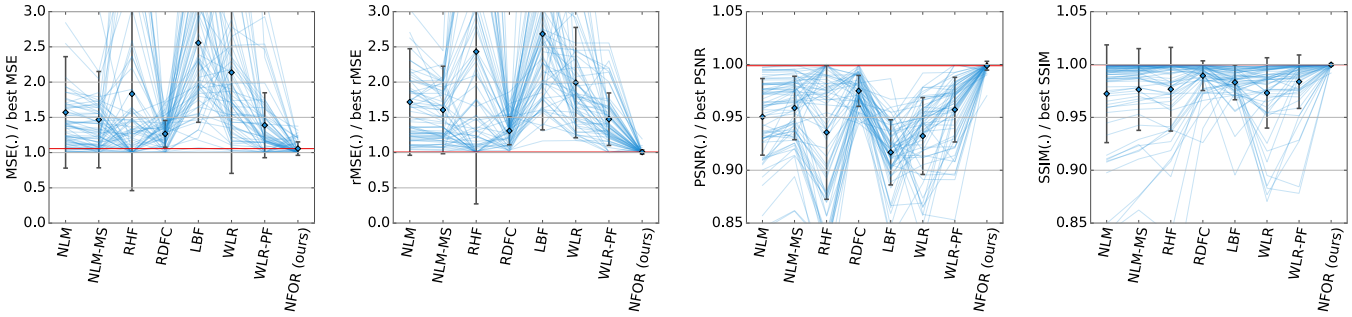

Figure 10: *Means and standard deviations of four metrics (MSE and rMSE; lower means better, PSNR and SSIM; higher means better) computed over 64, 256, and 1024 spp renderings of our test scenes; each blue line corresponds to one scene/spp configuration, all values are expressed relative to the best denosing technique thereof.*

HORSE ROOM, LIVING ROOM, RED ROOM) and 64 samples per pixel (SAN MIGUEL). On these scenes, RHF yields blurry results (although it performs better on hair data, shown in our supplemental material). LBF tends to be quite aggressive in its filtering, resulting in a loss of local details, but robustly preserves geometric and texture edges. RDFC and WLR give overall good results, but still suffer from noticeable artifacts. In contrast, our filter reliably preserves input detail without overblurring, while still effectively removing noise and introducing few to no artifacts.

8. Discussion

In its current state, our filter offers a good match for the characteristics of the ideal production denoiser outlined in the introduction. In particular, our filter offers a significant and stable improvement across a wide range of scenes, as demonstrated in our supplemental material. Similar to previous works, we leverage common auxiliary buffers that most renderers can readily provide, and operate in a post-process, allowing for a straightforward inclusion in the production pipeline. There are however a number of limitations with our filter, which we will now cover.

Computational overhead. Our current implementation is CPU-based and significantly slower than previous works; see timings in Table 2. While we would expect a GPU implementation to drastically reduce these timings, it remains that our filter is only suitable for offline rendering. We note however that rendering times of multiple core hours per frame are common in production, and that our filter can offer substantial improvements in this context. The main cost of our filter is the regression itself, which we perform three times (twice when filtering the half buffers in the first pass and once in the second pass), and the overall cost of our filter could be greatly reduced by a sparse collaborative reconstruction, instead of the dense one we currently use.

Table 2: *Average timings and memory consumption across all tested scenes. For this comparison, we implemented the listed algorithms in our CPU framework using the same set of optimizations; all other results use the implementations provided by the authors.*

Algorithm	Avg. Runtime	Avg. Memory Usage
RDFC	41s	349 MB
RHF	18s	460 MB
WLR	72s	309 MB
WLR-PF	91s	311 MB
NFOR (ours)	223s	3248 MB

Memory overhead. In order to better parallelize the filtering operations, we compute the full set of regression weights for the entire image buffer before fitting the data using our first-order model. Given that we use a window of 19×19 pixels, this represents a significant amount of memory, which could prove problematic for 4k renderings; see Table 2. As with the computational overhead, using a sparse collaborative reconstruction would drastically reduce the memory consumption of our technique.

Adaptive sampling. While our filter does not assume a uniform sampling distribution, we currently do not provide a way of guiding sampling based on the estimated MSE of our filter output.

Residual low-frequency noise. Large-scale temporal flickering due to residual low-frequency noise is a major challenge for image-space denoising techniques. While our spatio-temporal filtering results are on par with the current state of the art, they still leave room for improvement. We do not use motion vector to align frames temporally in our current implementation, which should offer some improvement. We have however experimented with the multiscale reconstruction proposed by Delbracio et al. [DMB*14], but have not been able to get consistently good results with it. In our supplemental material, we provide results using a standard NL-Means filter both without and with a multiscale reconstruction, which show the type of improvement this reconstruction can provide.

9. Conclusion

We have presented a comparative analysis of recent denoising methods, in order to assess their respective strength and weaknesses in the context of high-end production requirements. Based on this analysis, we identified a set of useful techniques that we combined in a novel filter, that consistently produces results on par with the current state of the art, and often provides substantial improvements. In its current implementation, our filter has a significant computational cost that restricts its use to high-quality off-line

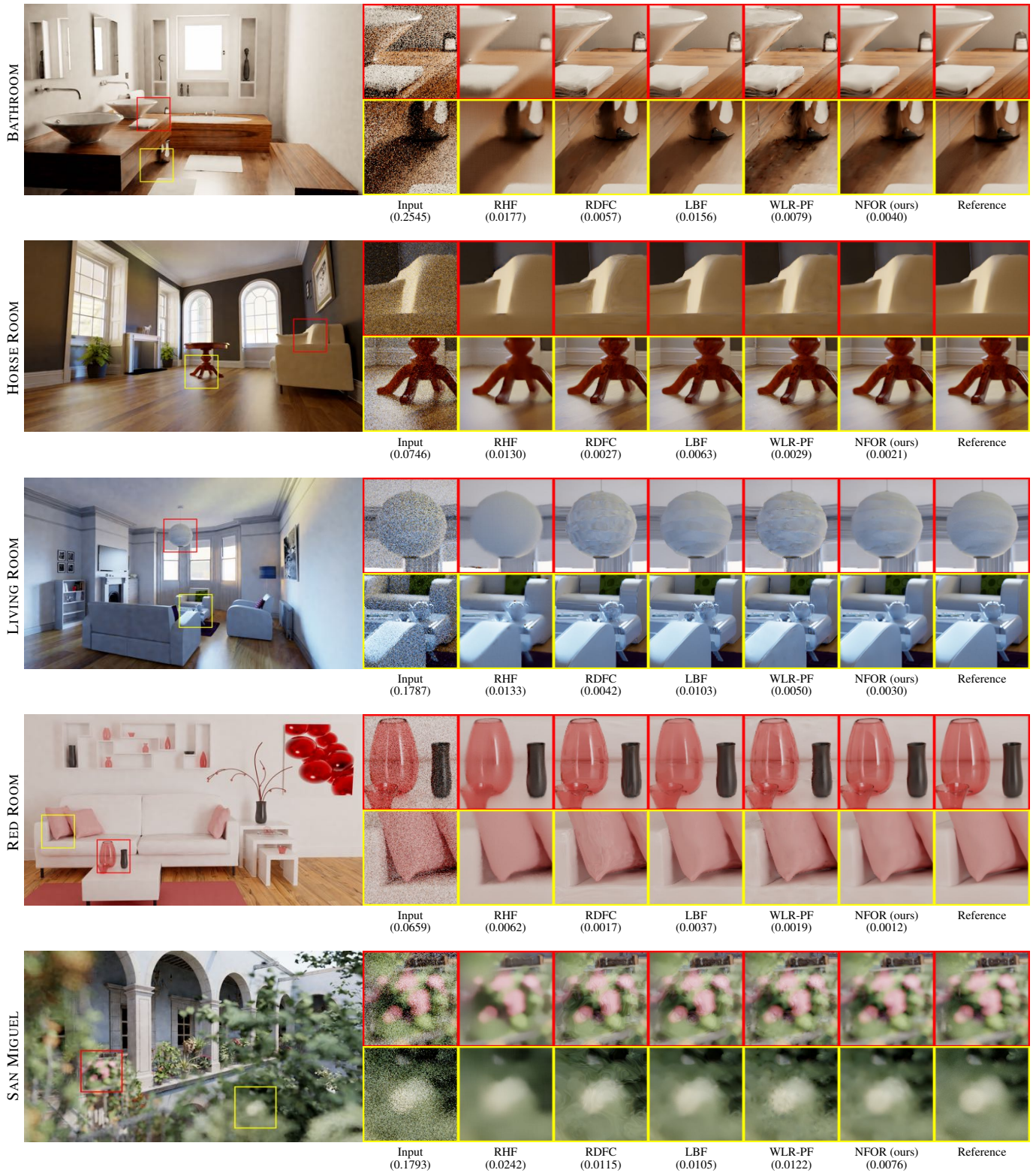

Figure 11: *We show results from our technique (large image) compared to four baseline methods. For each technique, we show two zoom-in locations and the rMSE (in parenthesis) to the reference; please refer to the supplemental material for full-resolution images and comprehensive comparison across all test scenes.*

rendering, and we have performed some promising early experiments using sparse reconstruction, which we intend to pursue further. Lastly, our work has not considered the important question of adaptive sampling, and we expect that a similar study of existing techniques could lead to interesting developments.

Acknowledgements

We thank the following blendswap.com artists: thecali (CAR and SPACESHIP), MrChimp2313 (HOUSE), UP3D (LAMP), SlykDrako (BEDROOM), nacimus (BATHROOM), Delatronic (DRAGON), Wig42 (HORSE ROOM, DINING ROOM, RED ROOM, STAIRCASE), Jay-Artist (LIVING ROOM), and NickWoelk (SHAVING KIT). We also thank Cem Yuksel (CURLY HAIR), Alvaro Luna Bautista and Joel Anderson (MUSEUM), Guillermo M. Leal Llaguno (SAN MIGUEL), Chris Harvey (DINOSAUR), Chris Scoville (SHEEP), and Maurizio Nitti (ROBOT). The BUNNY CLOUD comes from the OpenVDB website, and the geometry in the FOG scene from Minecraft. We thank Susan Harden for shepherding the internal approval process. This project was supported in part by Innovate UK (project #101858).

References

[BCM05] BUADES A., COLL B., MOREL J.-M.: A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation 4*, 2 (2005), 490–530. 2, 3, 8

[BCM07] BUADES A., COLL B., MOREL J.-M.: Nonlocal image and movie denoising. *International Journal of Computer Vision 76*, 2 (2007), 123–139. 8

[BEEM15] BAUSZAT P., EISEMANN M., EISEMANN E., MAGNOR M.: General and robust error estimation and reconstruction for Monte Carlo rendering. *Computer Graphics Forum 34*, 2 (2015), 597–608. 4, 5

[BEM11] BAUSZAT P., EISEMANN M., MAGNOR M.: Guided image filtering for interactive high-quality global illumination. *Comp. Graph. Forum 30*, 4 (2011), 1361–1368. 2, 3

[DMB*14] DELBRACIO M., MUSÉ P., BUADES A., CHAUVIER J., PHELPS N., MOREL J.-M.: Boosting Monte Carlo rendering by ray histogram fusion. *ACM Trans. Graph. 33*, 1 (Feb. 2014), 8:1–8:15. 2, 3, 8, 9

[Don95] DONOHO D. L.: De-noising by soft-thresholding. *Information Theory, IEEE Transactions on 41*, 3 (1995), 613–627. 2

[DSHL10] DAMMERTZ H., SEWTZ D., HANIKA J., LENSCH H. P. A.: Edge-avoiding à-trous wavelet transform for fast global illumination filtering. In *Proc. Conf. on High Performance Graphics* (2010), HPG '10, Eurographics Association, pp. 67–75. 2

[ED04] EISEMANN E., DURAND F.: Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3 (Aug. 2004), 673–678. 2

[Ela02] ELAD M.: On the origin of the bilateral filter and ways to improve it. *IEEE Transactions on Image Processing 11*, 10 (Oct 2002), 1141–1151. 2

[HST10] HE K., SUN J., TANG X.: Guided image filtering. In *Proc. 11th European Conf. on Computer Vision: Part I* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 1–14. 2

[KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering Monte Carlo noise. *ACM Trans. Graph. (Proc. SIGGRAPH) 34*, 4 (July 2015), 122:1–122:12. 2, 3, 4, 8

[KFF*15] KELLER A., FASCIONE L., FAJARDO M., GEORGIEV I., CHRISTENSEN P., HANIKA J., EISENACHER C., NICHOLS G.: The path tracing revolution in the movie industry. In *ACM SIGGRAPH 2015 Courses* (2015), ACM, pp. 24:1–24:7. 1

[KS13] KALANTARI N. K., SEN P.: Removing the noise in monte carlo rendering with general image denoising algorithms. *Comp. Graph. Forum (Proc. Eurographics) 32*, 2pt1 (2013), 93–102. 2

[LWC12] LI T.-M., WU Y.-T., CHUANG Y.-Y.: Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 31*, 6 (Nov. 2012), 194:1–194:9. 2, 4

[McC99] MCCOOL M. D.: Anisotropic diffusion for Monte Carlo noise reduction. *ACM Trans. Graph. 18*, 2 (Apr. 1999), 171–194. 2

[MCY14] MOON B., CARR N., YOON S.-E.: Adaptive rendering based on weighted local regression. *ACM Trans. Graph. 33*, 5 (Sept. 2014), 170:1–170:14. 2, 3, 4, 8

[MIGYM15] MOON B., IGLESIAS-GUITIAN J. A., YOON S.-E., MITCHELL K.: Adaptive rendering with linear predictions. *ACM Trans. Graph. (Proc. SIGGRAPH) 34*, 4 (July 2015), 121:1–121:11. 2, 5, 7

[MJL*13] MOON B., JUN J. Y., LEE J., KIM K., HACHISUKA T., YOON S.-E.: Robust image denoising using a virtual flash image for Monte Carlo ray tracing. *Comp. Graph. Forum 32*, 1 (2013), 139–151. 2

[MMMG16] MOON B., MCDONAGH S., MITCHELL K., GROSS M.: Adaptive polynomial rendering. *To appear in ACM Trans. Graph. (Proc. SIGGRAPH)* (2016). 2

[MVZ16] MANZI M., VICINI D., ZWICKER M.: Regularizing Image Reconstruction for Gradient-Domain Rendering with Feature Patches. *To appear in Comp. Graph. Forum (Proc. Eurographics)* (2016). 2

[ODR09] OVERBECK R. S., DONNER C., RAMAMOORTHI R.: Adaptive wavelet rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 28*, 5 (2009), 140–1. 2

[PKTD08] PARIS S., KORNPROBST P., TUMBLIN J., DURAND F.: Bilateral filtering: Theory and applications. *Foundations and Trends in Computer Graphics and Vision 4*, 1 (2008), 1–73. 2

[PSA*04] PETSCHNIGG G., SZELISKI R., AGRAWALA M., COHEN M., HOPPE H., TOYAMA K.: Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3 (Aug. 2004), 664–672. 2

[RKZ11] ROUSSELLE F., KNAUS C., ZWICKER M.: Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 30*, 6 (Dec. 2011), 159:1–159:12. 2, 4

[RKZ12] ROUSSELLE F., KNAUS C., ZWICKER M.: Adaptive rendering with non-local means filtering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 31*, 6 (Nov. 2012), 195:1–195:11. 2, 3, 4, 6, 8

[RMZ13] ROUSSELLE F., MANZI M., ZWICKER M.: Robust denoising using feature and color information. *Comp. Graph. Forum (Proc. of Pacific Graphics) 32*, 7 (2013), 121–130. 2, 3, 4, 6, 7, 8

[SD12] SEN P., DARABI S.: On filtering the noise from the random parameters in Monte Carlo rendering. *ACM Trans. Graph. 31*, 3 (June 2012), 18:1–18:15. 2, 4

[Ste81] STEIN C. M.: Estimation of the mean of a multivariate normal distribution. *The annals of Statistics* (1981), 1135–1151. 2, 5, 7

[TFM07] TAKEDA H., FARSIU S., MILANFAR P.: Kernel regression for image processing and reconstruction. *IEEE Transactions on Image Processing 16*, 2 (Feb 2007), 349–366. 3

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conf. on* (1998), pp. 839–846. 2

[VdVK09] VAN DE VILLE D., KOCHER M.: Sure-based non-local means. *IEEE Signal Process. Lett. 16*, 11 (2009), 973–976. 2

[XP05] XU R., PATTANAIK S.: A novel Monte Carlo noise reduction operator. *IEEE Comput. Graph. Appl. 25*, 2 (2005), 31–35. 2

[ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHI R., ROUSSELLE F., SEN P., SOLER C., YOON S.-E.: Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Comp. Graph. Forum (Proc. Eurographics) 34*, 2 (May 2015), 667–681. 1, 2