Pacific Graphics 2013
B. Levy, X. Tong, and K. Yin
(Guest Editors)

Volume 32 (2013), Number 7

Robust Denoising using Feature and Color Information

Fabrice Rousselle   Marco Manzi   Matthias Zwicker

University of Bern

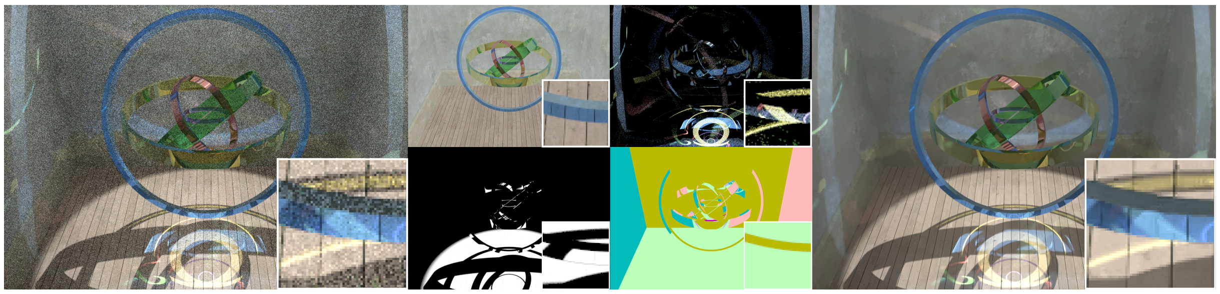

Figure 1: *We propose a method to denoise Monte Carlo renderings using noisy color (left) and feature buffers (middle: texture, caustics, visibility, normals) as an input. We construct a denoising filter by combining color and feature information using a SURE error estimate. Our results (right) improve visually and quantitatively over the previous state-of-the-art.*

Abstract

We propose a method that robustly combines color and feature buffers to denoise Monte Carlo renderings. On one hand, feature buffers, such as per pixel normals, textures, or depth, are effective in determining denoising filters because features are highly correlated with rendered images. Filters based solely on features, however, are prone to blurring image details that are not well represented by the features. On the other hand, color buffers represent all details, but they may be less effective to determine filters because they are contaminated by the noise that is supposed to be removed. We propose to obtain filters using a combination of color and feature buffers in an NL-means and cross-bilateral filtering framework. We determine a robust weighting of colors and features using a SURE-based error estimate. We show significant improvements in subjective and quantitative errors compared to the previous state-of-the-art. We also demonstrate adaptive sampling and space-time filtering for animations.

1. Introduction

Monte Carlo rendering suffers from noise artifacts that can often only be avoided by sampling an excessive number of light paths. This has slowed the adoption of Monte Carlo rendering in applications ranging from movie production to real-time rendering. While a vast variety of variance reduction and sophisticated sampling techniques have been proposed for Monte Carlo rendering, a renewed interest in image space filtering methods has shown that such methods can be surprisingly effective. These methods are appealing because they are relatively easy to implement, mostly orthogonal to other variance reduction techniques, applicable to general light transport effects, computationally efficient, and often competitive with more specialized approaches targeting specific rendering effects. A particularly successful idea is to use feature buffers, such as per pixel normals, textures, or depth, to compute denoising filter weights. Feature buffers are highly correlated with the rendered image, since they represent most edges in the image, but they are usually much less noisy than the color output of the Monte Carlo renderer. Therefore, filters based on feature buffers effectively remove noise while preserving most edges. Unfortunately, they are prone to blurring image details that are not well represented by the features. Feature buffers have been used to determine denoising filters in the context of guided image filtering, which is based on local regression, wavelet thresholding, and cross-bilateral filtering.

In this paper we present a method to robustly combine color and feature buffers to improve denoising performance. We use a filtering framework based on NL-means filter weights for color buffers and bilateral weights for feature buffers. We control the influence of color and feature buffers by adjusting the parameters of the NL-means and bilateral filters. To combine color and feature information, we evaluate three candidate filters using different parameters designed to provide a trade-off between fidelity to image detail and robustness to noise. Then we compute a weighted average of the candidate filters on a per-pixel basis using a SURE-based error estimate to minimize the output error. We deal with noisy features by denoising them first in a separate step using an NL-means filter. This allows us to include novel features, such as a caustics buffer shown in Figure 1, and a direct visibility feature. We demonstrate that our approach leads to significant improvements both in subjective and quantitative errors compared to the previous state-of-the-art. In summary, we make the following contributions:

- We propose to combine color and feature buffers to improve image space denoising of Monte Carlo renderings.
- We implement this idea based on NL-means and cross-bilateral filtering, and a SURE-based error estimate.
- We propose to deal with noisy features by denoising them in a separate step. This allows us to introduce novel features such as caustics and direct visibility.
- We demonstrate significant subjective and numerical improvements in image quality over the previous state-of-the-art.
- We extend our approach to adaptive sampling and space-time filtering for animations.

2. Previous Work

Image space filtering techniques are often inspired by denoising algorithms from the image processing research community, adapted to the specifics of Monte Carlo rendering. Early work by McCool [McC99] adapts anisotropic diffusion to filter noisy Monte Carlo images. Xu et al. [XP05] build on the bilateral filter [TM98]. Overbeck et al. [ODR09] propose an algorithm based on wavelet shrinkage [DJ94]. Rousselle et al. [RKZ11] use adaptive bandwidth selection [Kat99] and later [RKZ12] non-local means filtering [BCM05]. Kalantari et al. [KS13] show that general image denoising algorithms such as BM3D [DFKE07] can be effectively applied to Monte Carlo renderings. These methods also include error estimation techniques, which are used in combination with an adaptive sampling stage.

While the above methods only rely on color samples obtained from Monte Carlo renderers, filtering quality can be greatly improved by exploiting auxiliary per pixel information, which we call *features*, such as per pixel normals, depth, or texture. These features can be used to obtain effective filters because they are highly correlated with the output image, but they are usually much less noisy. This has first been explored in the context of real-time rendering. For example, Bauszat et al. [BEM11] build on the guided image filter [HST10], and Dammertz et al. [DSHL10] employ a fast wavelet transform. A disadvantage of these techniques is that they assume that the features are not corrupted by noise. Shirley et al. exploit the depth buffer to denoise motion and defocus blur [SAC*11].

The current state-of-the-art in off-line rendering combines feature-based filters with error estimation techniques to drive adaptive sampling, and includes approaches to deal with noisy features. Sen and Darabi [SD12] propose a technique based on cross-bilateral filtering [ED04] using an information theoretic approach to deal with noisy features. Li et al. [LWC12] combine a per-pixel SURE-based error estimate [Ste81] with cross-bilateral filtering. Van De Ville and Kocher [VDVK09] propose a SURE-based error estimate for NL-means denoising, although they used it for parameter selection on a per-image rather than per-pixel basis. Moon et al. [MJL*13] apply an NL-means filter guided by a virtual flash image. Our approach is most related to the work by Li et al., with some significant differences. While they optimize over the spatial filter support size, we find a trade-off between driving the filter using color or feature information. We denoise features in a separate step, which allows us to effectively exploit novel features such as direct light source visibility or caustics that tend to be noisy. Finally, we add a second filtering pass based on a variance estimate of the first pass. Together, this yields significant quality improvements.

An alternative to image space methods are techniques that attempt to exploit the higher dimensional structure of light transport effects. Hachisuka et al. [HJW*08] describe a general framework for multi-dimensional adaptive sampling and reconstruction. A number of techniques build on four-dimensional light field representations to obtain impressive results for specific effects such as motion blur [ETH*09], motion blur and depth of field [LAC*11], area lights and complex occluders [EHDR11], or indirect illumination [LALD12], without raising any claim to completeness. The recent work by Mehta et al. [MWR12] on image space filtering of soft shadows explores an interesting middle ground: combining an accurate analysis in higher-dimensional space with the efficiency of image space filtering. Since the scope of pure image space methods is more general compared to these techniques, a direct comparison would be less meaningful.

3. Overview

Filtering based on feature buffers, such as per pixel normal, texture, or depth, has proven extremely effective, in particular for images rendered with very few samples per pixel and high noise levels in the Monte Carlo output [SD12,LWC12]. On the other hand, image details that are not represented in the feature buffers tend to be blurred by such approaches. Hence our main idea is to construct a filter that implements a balance between filtering using color and feature information. In general, our filter computes a weighted average of neighboring pixels. The filtered color values $F(p) = (F_1(p), F_2(p), F_3(p))$ of a pixel p in a color image $u(p) = (u_1(p), u_2(p), u_3(p))$ are

$$F_i(p) = \frac{1}{C(p)} \sum_{q \in N(p)} u_i(q) w(p,q), \tag{1}$$

where $N(p)$ is a $2r+1 \times 2r+1$ square neighborhood centered on p, $w(p,q)$ is the weight of the contribution of neighboring pixel q to p, i is the index of the color channel, and $C(p) = \sum_{q \in N(p)} w(p,q)$ is a normalization factor. The fundamental challenge is to determine suitable weights $w(p,q)$.

In our approach, illustrated in Figure 2, we construct three *candidate filters*, which we call the *FIRST*, *SECOND*, and *THIRD candidate filter*. We design the filters such that the FIRST filter is most sensitive to details in the color buffer, but also most sensitive to its noise; the THIRD filter is least sensitive to noise in the colors, but also least sensitive to its details; and the SECOND filter is in between. Then we compute the final filter as a weighted average of the candidate filters using a SURE-based per-pixel error estimate. We build the candidate filters from two types of weights, called color and feature weights. We obtain the color weights as NL-means weights from the noisy color output of the Monte Carlo renderer as described in Section 4. We compute the feature weights as bilateral weights from the feature buffers, as presented in Section 5. In Section 6 we then describe how we construct the FIRST, SECOND, and THIRD candidate filters from the color and feature weights, and how we compute the candidate filter averaging weights using SURE error estimation. We provide a summary of our algorithm in Section 7, and in Section 8 we present extensions to adaptive sampling and space-time filtering for animations.

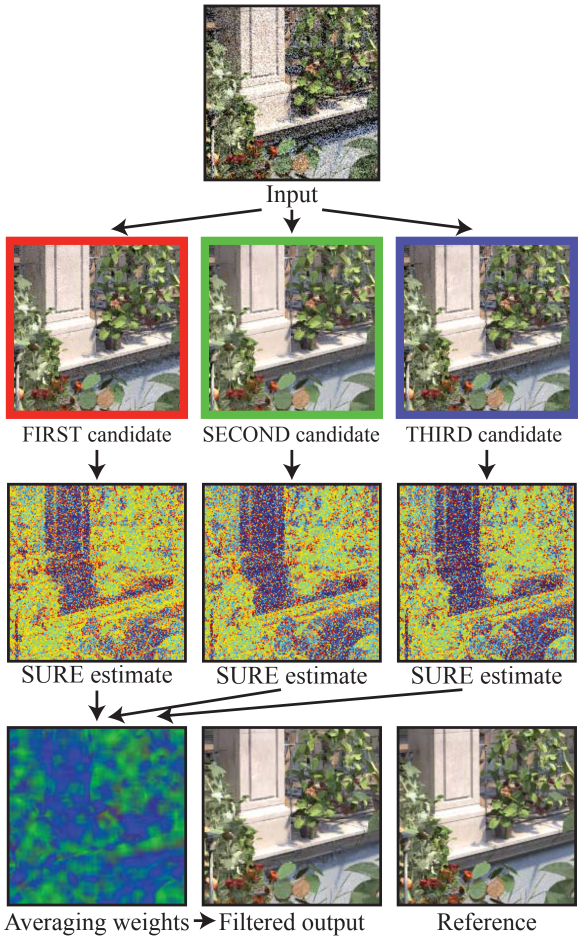

Figure 2: *Our main idea is to filter using a balance of color and feature information. We evaluate three candidate filters designed to provide a trade-off between sensitivity to image detail and robustness to noise. We compute averaging weights for the candidate filters at each pixel using a SURE-based error estimation. The color coding of the weights corresponds to the FIRST, SECOND, and THIRD candidates.*

4. NL-means Weights from Color Buffer

Our color weights w_c are based on NL-means filtering [BCM05], which has proven effective for denoising Monte Carlo renderings because it can easily be generalized to spatially varying variances typical in such data [RKZ12]. We compute the NL-means weights from the noisy color output of the Monte Carlo renderer, and per-pixel variance estimates. We next review NL-means weights computation and then describe our per-pixel variance estimates.

NL-Means Weights. NL-means weights for a pixel p and a neighbor q are determined based on the distance $d_c^2(P(p), P(q))$ between a pair of small patches $P(p)$ and $P(q)$ of size $2f+1 \times 2f+1$ centered at p and q,

$$d_c^2(P(p), P(q)) = \frac{1}{3(2f+1)^2} \sum_{i=1}^{3} \sum_{n \in P(0)} \Delta_i^2(p+n, q+n),$$

where $\Delta_i^2(p+n, q+n)$ is is a per-pixel distance in color channel i and $n \in P(0)$ are the offsets to each pixel within a patch. We follow the approach by Rousselle et al. [RKZ12] and define the per-pixel distance as

$$\Delta_i^2(p,q) = \frac{(u_i(p) - u_i(q))^2 - (\mathrm{Var}_i[p] + \mathrm{Var}_i[q,p])}{\varepsilon + k_c^2(\mathrm{Var}_i[p] + \mathrm{Var}_i[q])}.$$

The term $(u_i(p) - u_i(q))^2$ measures the squared difference between the color values at pixels p and q. Since $u_i(p)$ and $u_i(q)$ are noisy this consistently overestimates the true squared difference. Hence, we subtract a variance cancellation term $(\mathrm{Var}_i[p] + \mathrm{Var}_i[q,p])$ to remove this bias, similar as proposed originally for NL-means [BCM05], where $\mathrm{Var}_i[p]$ is a variance estimate for the sample mean in pixel p, and $\mathrm{Var}_i[q,p] = min(\mathrm{Var}_i[q], \mathrm{Var}_i[p])$. The denominator

$\varepsilon + k_c^2(\text{Var}_i[p] + \text{Var}_i[q])$ is a normalization factor, where ε is a small value to prevent division by zero, and k_c is a user specified factor that controls the sensitivity of the filter to color differences. Larger values of k_c lead to more aggressive filtering. Finally, we obtain our color filter weight $w_c(p,q)$ of the contribution of pixel q to p using an exponential kernel,

$$w_c(p,q) = \exp^{-\max(0, d_c^2(P(p), P(q)))}. \tag{2}$$

Variance Estimation. In the case of random sampling, we could estimate the variances Var_i of pixel means simply by considering the sample variance within each pixel. This approach is not suitable, however, to support low-discrepancy sampling where it will consistently overestimate the variance. Rousselle et al. [RKZ12] address this problem by splitting the noisy color samples into two half-buffers and computing the empirical variance of these half-buffers. While this is an unbiased estimate of the pixel variance, it is also very noisy and leads to poor NL-means filtering performance if used directly. To address this, we observe that the sample variance exhibits the detailed structure of the actual spatially non-uniform variance, but with a systematic bias. Hence we attempt to remove this bias by simply scaling the sample variance to match the magnitude of the two-buffer variance. We smooth both the sample variance and the two-buffer variance with a large, 21×21 box filter and then compute the ratio between the two on a per-pixel basis. We then apply this ratio on the initial unfiltered sample variance. This results in a variance estimate with the lower noise of the sample variance, and the correct magnitude of the two-buffer variance.

5. Cross-bilateral Weights from Feature Buffers

We determine the feature weights w_f using the feature buffers and bilateral weights. An important distinction compared to previous work exploiting feature buffers [LWC12, SD12] is that we deal with noisy features by first prefiltering them separately. We next describe our feature prefiltering technique, and then the computation of the bilateral weights using the prefiltered features.

Feature Prefiltering. Our prefiltering approach exploits the fact that features can be denoised effectively because their dynamic range, and hence their variance, is typically limited. We apply an NL-means filter as described in the previous section including the same method to estimate the input variance, although using individual features instead of color as input. We choose window radius $r = 5$, patch radius $f = 3$, and sensitivity $k_c = 1.0$ for all features.

Output Variance Estimation. To determine the bilateral weights we will also require the residual variance of the prefiltered features, that is, we need the per-pixel variance of the prefiltered output. We obtain the output variance using a two-buffer approach similar to Section 4. We split the feature data into two half-buffers that we both filter using the same NL-means weights determined from the complete data, as described above. Note that given fixed weights, the filter (see Equation 1) is linear, and averaging the filtered half-buffers is equivalent to filtering the full data. By processing the half-buffers, however, we can estimate the residual per-pixel variance as the squared per-pixel difference between the filtered half-buffers. We further reduce noise in this two-buffer variance estimate by smoothing it with a small Gaussian kernel with standard deviation of 0.5 pixels.

Bilateral Weights. We denote feature buffers such as normals, textures, or depth, denoised and normalized to unit range, as f_j. The feature distance $\Phi_j^2(p,q)$ for feature j between pixels p and q is based on the squared feature difference including variance cancellation similar to Section 4,

$$\Phi_j^2(p,q) = \frac{(f_j(p) - f_j(q))^2 - (\text{Var}_j[p] + \text{Var}_j[p,q])}{k_f^2 \max(\tau, \max(\text{Var}_j[p], \|\text{Grad}_j[p]\|^2))},$$

normalized by two factors: First, the user parameter k_f controls the sensitivity of the filter to feature differences. The second factor depends on the residual variance of the prefiltered feature (as described above) denoted by $\text{Var}_j[p]$ and the squared gradient magnitude $\|\text{Grad}_j[p]\|^2$, thresholded to a minimum value τ. This factor normalizes the feature distance relative to residual noise left in the filtered feature and the local feature contrast, measured by its gradient magnitude. Finally, we obtain an overall distance $d_f(p,q)$ by taking the maximum distance over all M features,

$$d_f^2(p,q) = \underset{j \in [1 \ldots M]}{\operatorname{argmax}} \Phi_j^2(p,q),$$

and the final feature weight $w_f(p,q)$ is obtained using an exponential kernel, similar as in Section 4,

$$w_f(p,q) = \exp^{-d_f^2(p,q)}. \tag{3}$$

We illustrate the benefit of our feature prefiltering step in Figure 3 on a simple scene with depth of field. With prefiltering, we effectively remove noise in out-of-focus regions, while preserving detail otherwise. Feature prefiltering allows us to exploit novel types of features that tend to be too noisy to be useful without prefiltering. For the "rings" scene in Figure 1 we define a caustics feature as the per-pixel density of caustic photons. We also introduce a direct illumination visibility feature as the fraction of shadow rays that hit any light source over all direct shadow rays evaluated in a pixel. Figure 4 illustrates how considering the feature gradient in the distance normalization improves filtering performance. Without the gradient term feature weights along edges are too restrictive, preventing effective filtering.

6. Filter Weighting using SURE-based Error Estimate

We use a SURE-based approach [Ste81] to estimate the mean squared error (MSE) of three candidate filters, which we design to provide a trade-off between fidelity to image

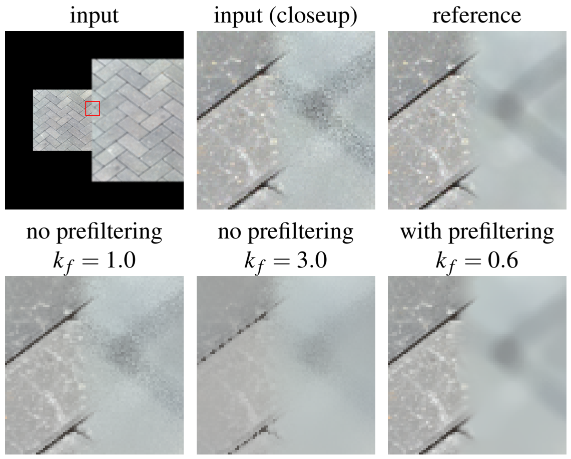
input
input (closeup)
reference
no prefiltering $k_f = 1.0$
no prefiltering $k_f = 3.0$
with prefiltering $k_f = 0.6$

Figure 3: *We show the effectiveness of feature prefiltering on a scene with depth-of-field, which leads to noisy features. We use only the texture feature. The left quad is in-focus and noise free, while the right quad is out-of-focus and noisy (top left). Our approach with prefiltering preserves detail in the in-focus-region while effectively denoising the out-of-focus region (bottom right). Without prefiltering the bilateral feature weights fail to distinguish between noise and texture detail. While smaller k_f values preserve texture detail, they cannot remove the noise (bottom left). Larger k_f values yield smoother results but blur out the details (bottom middle).*

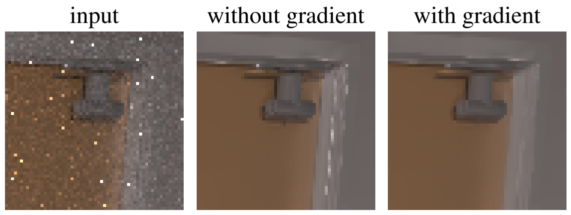

Figure 4: *Filtered output for the "conference" scene without (middle image) and with the feature gradient (right image). Including the gradient term improves filtering along edges.*

detail and robustness to noise. We then leverage the error estimate to compute a weighted average of the candidate filters minimizing the error on a per-pixel basis. We next describe the candidate filters, the SURE-based error estimate, and the computation of the per-pixel filter averaging weights.

Candidate Filters. Our FIRST, SECOND, and THIRD candidate filters (Figure 2) differ in their color sensitivity k_c and patch radius f. The FIRST filter uses $k_c = 0.45$ and a small patch radius of $f = 1$, which makes it sensitive to small image detail but also less robust to noise. The SECOND filter is the same except that it uses a larger patch radius $f = 3$. Hence it is more robust to noise but less effective at filtering intricate image detail at low noise levels. The THIRD filter has $k_c = \infty$, which means that the color information does not influence the filter weights and the patch size f is irrelevant. This filter is most robust towards noise since its weights completely ignore color information. However, it fails to recover image detail that is not represented in the features. All filters use feature sensitivity $k_f = 0.6$ and the same window radius r, which is the only parameter we expose to the user. We determine the final filter weights by taking the minimum of the color and feature weights, that is

$$w(p,q) = \min(w_c(p,q), w_f(p,q)). \tag{4}$$

SURE Error Estimate. We explain the SURE error estimate by extending our notation from Equation 1 for a generic color image filter. Let us interpret the output $F_{u_i}(p)$ of the filter at pixel p as a function of the noisy value u_i of color channel i at p. The SURE error estimator at pixel p is then

$$\text{SURE}(p) = \sum_{i=1}^{3} \|F_{u_i}(p) - u_i\|^2 - \sigma_i^2 + 2\sigma_i^2 \frac{\mathrm{d}F_{u_i}(p)}{\mathrm{d}u_i} \tag{5}$$

where σ_i^2 is the true variance of the pixel mean of color channel i. Since our color weights w_c include discontinuous terms, they are technically not differentiable. But we found that a finite difference approximation of the derivatives still leads to reliable error estimates in practice. Hence we approximate $\mathrm{d}F_{u_i}(p)/\mathrm{d}u_i$ as,

$$\frac{\mathrm{d}F_{u_i}(p)}{\mathrm{d}u_i} \approx \frac{F_{u_i+\delta}(p) - F_{u_i}(p)}{\delta}, \text{ where } \delta = 0.01 \times u_i.$$

Candidate Filter Averaging. While SURE provides an unbiased error estimate, it is very noisy and needs to be filtered for per-pixel error minimization. A straightforward approach would be to spatially smooth the error estimate until its variance is low enough to reliably determine the best candidate filter on a per-pixel basis. Unfortunately, in our typical data the SURE estimate tends to be contaminated by strong outliers. This requires large spatial smoothing filters, introducing bias in the per-pixel error estimates.

Instead, we achieved better results with a two-step strategy that is more robust to outliers in the initial error estimate. In a first step, we smooth the error estimate with a small kernel and obtain three binary selection maps for the candidate filters, containing a 1 in each pixel if the filter had lowest error, and 0 otherwise. Since the FIRST candidate filter is most sensitive to noise it may occur that it preserves noise, but the SURE estimate does not register the error. We avoid such residual noise by selecting the FIRST candidate only if it filters more than the SECOND, that is, the derivative term in the SURE estimate is lower for the FIRST candidate. In the second step we smooth the binary maps with a larger kernel. This approach has the advantage that the binary selection maps suppress outliers, which allows us to use smaller smoothing kernels in the second step. We found that the smoothing step of the initial error estimate is necessary to obtain sufficiently discriminatory binary maps.

7. Algorithm

We summarize our algorithm in Algorithm 1. The workhorse of our filtering pipeline, which we use to compute the three candidate filters and some auxiliary filtering, is the function

$$[out, d\_out\_d\_in] = flt(in, u, var\_u, f[], var\_f[], p).$$

It filters an input *in* by constructing color weights (Equation 2) from a color buffer u with variance var_u, and feature weights (Equation 3) from a set of feature buffers $f[]$ with variances $var\_f[]$, and taking their minimum as in Equation 4. The function also returns the derivative $d\_out\_d\_in$ that is needed for the SURE estimate (Equation 5). The parameter values are specified in the structure p. We further assume that the input *in* and output *out* of the filter consist of two half-buffers to support two-buffer output variance estimation (Section 5, output variance estimation). We implement the SURE error estimate (Equation 5) in a function

$$SURE(u, var\_u, F, dF\_du)$$

that takes a noisy buffer u with its variance var_u, and its filtered version F with derivative dF_du as input. The algorithm also uses an auxiliary function *scale_svar* that takes a buffer (consisting of two half-buffers) and its sample variance as input and implements the sample variance scaling technique described in Section 4. Finally, the function *buffer_var* computes the two-buffer variance of a buffer (consisting of two half-buffers) smoothed with a Gaussian kernel (Section 5, output variance estimation). In addition to the steps described so far, our algorithm includes a final filtering pass that removes residual noise (lines $27-29$), which we detect by computing the two buffer variance of the output. This residual noise is typically located along edges, where the filter is more constrained.

Algorithm 1: DENOISE

```
Input: Noisy color buffer $c$ with sample variance
       $svar\_c$; set of $M$ noisy feature buffers $f[]$ with
       sample variances $svar\_f[]$; window radius $R$
Output: Denoised image $pass2$
1  begin
       /* Sample variance scaling. */
2      $var\_c = scale\_svar(svar\_c, c)$
3      for all features $j = 1 \ldots M$ do
4          $var\_f[j] = scale\_svar(svar\_f[j], f[j])$
       /* Feature prefiltering. */
5      $p = \{k_c = 1, k_f = \infty, f = 3, r = 5\}$
6      for all features $j = 1 \ldots M$ do
7          $flt\_f[j] = flt(f[j], f[j], var\_f[j], nil, nil, p)$
8          $var\_flt\_f[j] = buffer\_var(flt\_f[j])$
       /* Candidate filters. */
9      $p = \{k_c = 0.45, k_f = 0.6, f = 1, r = R, \tau = 10E-3\}$
10     $[r, d\_r] = flt(c, c, var\_c, flt\_f[], var\_flt\_f[], p)$
11     $p = \{k_c = 0.45, k_f = 0.6, f = 3, r = R, \tau = 10E-3\}$
12     $[g, d\_g] = flt(c, c, var\_c, flt\_f[], var\_flt\_f[], p)$
13     $p = \{k_c = \infty, k_f = 0.6, f = 1, r = R, \tau = 10E-4\}$
14     $[b, d\_b] = flt(c, c, var\_c, flt\_f[], var\_flt\_f[], p)$
       /* Filtered SURE error estimates. */
15     $p = \{k_c = 1.0, k_f = \infty, f = 1, r = 1, \tau = 10E-3\}$
16     $e\_r = flt(SURE(c, var\_c, r, d\_r), c, var\_c, nil, nil, p)$
17     $e\_g = flt(SURE(c, var\_c, g, d\_g), c, var\_c, nil, nil, p)$
18     $e\_b = flt(SURE(c, var\_c, b, d\_b), c, var\_c, nil, nil, p)$
       /* Binary selection maps. */
19     $sel\_r = e\_r < e\_g \&\& e\_r < e\_b$
           $\&\& d\_r < d\_g\ ?\ 1 : 0$
20     $sel\_g = e\_g < e\_r \&\& e\_g < e\_b\ ?\ 1 : 0$
21     $sel\_b = e\_b < e\_r \&\& e\_g < e\_b\ ?\ 1 : 0$
       /* Filter selection maps. */
22     $p = \{k_c = 1, k_f = \infty, f = 1, r = 5, \tau = 10E-3\}$
23     $sel\_r = flt(sel\_r, c, var\_c, nil, nil, p)$
24     $sel\_g = flt(sel\_g, c, var\_c, nil, nil, p)$
25     $sel\_b = flt(sel\_b, c, var\_c, nil, nil, p)$
       /* Candidate filter averaging. */
26     $pass1 = r * sel\_r + g * sel\_g + b * sel\_b$
       /* Second pass filtering. */
27     $p = \{k_c = 0.45, k_f = \infty, f = 1, r = R, \tau = 10E-4\}$
28     $var\_pass1 = buffer\_var(pass1)$
29     $pass2 = flt(pass1, pass1, var\_pass1, nil, nil, p)$
```

8. Extensions

Adaptive Sampling. We follow the adaptive sampling strategy proposed by Rousselle et al. [RKZ12] which we summarize here. We distribute samples over multiple iterations, each having an equal share of the sample budget. The first iteration performs uniform sampling, and the subsequent ones perform adaptive sampling. In the adaptive iterations, the sampling density is proportional to the estimated relative MSE returned by SURE, scaled by a weight W. The weight W represents the error reduction potential of a single sample, and accounts for the number of samples used in the filter, as well as the filter support,

$$W(p) = \frac{\sum_{q \in N(p)} w(p,q)}{1 + n_p},$$

where p is a pixel, $N(p)$ the filter window around p, $w(p,q)$ the weight of a neighbor q within the window, and n_p the number of samples already contributing to the filtered value of p. The resulting weighted error is quite noisy, so we filter it aggressively with our flt function and parameters $r = 10$, $k_c = 1.0$, and $k_f = \infty$. Lastly we clamp the number of samples allocated to each pixel to a fixed value to prevent spending too many samples on outliers.

Space-time Filtering for Animations. Filtering animations on a per-frame basis suffers from disturbing flickering artifacts due to low-frequency residual noise. We can greatly mitigate these problems by space-time filtering. We implement space-time filtering in our framework by extending our filtering window from a spatial to a spatio-temporal window across several frames, as proposed by Buades et al. [BCM08]. Otherwise our algorithm remains the same as before. We provide animations for the "conference" and "sanmiguel" scenes in our supplemental material, demonstrating both spatial and space-time filtering. While there is hardly any noticeable difference on still frames, the space-time filtered output exhibits significantly less flickering, especially in the "conference" scene.

9. Results

We integrated our method as an extension of the PBRT rendering framework [PH10] and implemented the filtering operations themselves in CUDA for GPU acceleration. The complexity of each filtering step is proportional to image resolution, window radius f, and patch radius r. For an image resolution of 1024×1024 pixels and user specified window radius $r = 10$, the complete filtering pipeline summarized in Algorithm 1 takes 5.4 seconds on an NVidia GeForce GTX TITAN GPU, and 8.0 seconds on a Geforce GTX 580 GPU. All timings reported below were measured on a workstation with dual 6-core Intel Xeon processor at 2.3 GHz, 12 rendering threads, and a Geforce GTX TITAN GPU.

The "rings" scene in Figure 1 is rendered using PBRT's photon mapper using 32 samples per pixel, 450k caustic photons, 1000k indirect photons, and 4 final gather rays per sample. It took 869s to render, and 19.5s to filter with a window radius $r = 20$ (the filtering cost is higher for this scene, since we have the additional caustics buffer and a larger window radius). It uses our novel caustics feature, which stores the density of caustics photons. We use ray differentials to compute pixel-sized radiance estimation radii to avoid blurring of caustics. Figure 5 further highlights the contribution of our novel features. We show closeups of the "sanmiguel" scene filtered with and without the visibility feature, and the "rings" scene with and without the caustics feature. Including the novel features clearly improves our results.

Figure 6 contains log-log convergence plots for the "sibenik", "conference", and "sanmiguel" scenes. We show results for our complete filter using uniform sampling (OUR in cyan) and adaptive sampling (dotted cyan), our candidate filters (red, green, and blue), and the work by Li et al. [LWC12] (SBF in magenta). The plots indicate that our SURE-based weighted averaging of the candidate filters consistently improves the error of the individual candidate filters. It is also interesting that both our FIRST and SECOND filters generally outperform SBF, which underlines the usefulness of including color weights in the filter.

In Figure 7 (last page) we compare rendering with low-discrepancy sampling and no filtering (LD), the approach by Li et al. [LWC12] (SBF) using the implementation provided by the authors, and our technique (OUR) at roughly equal render time. We also include a reference image (REFERENCE). We use uniform sampling in this comparison since

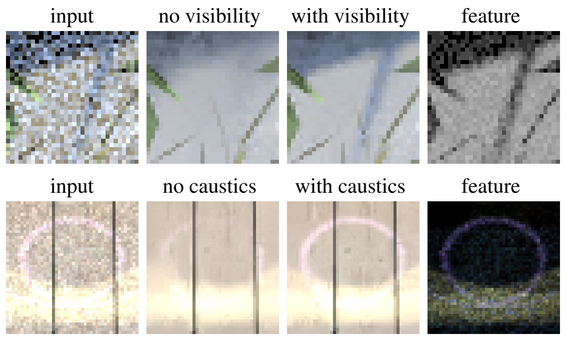

Figure 5: *Closeups of "sanmiguel" filtered with and without the visibility feature, and "rings" with and without the caustics feature. The features shown on the right (before prefiltering) help preserving important details.*

we found that SBF often gave worse results with adaptive sampling. Our feature buffers include texture, depth, normal, and visibility. We attribute our improvements over previous work to three main factors: the consideration of the color buffer to construct the final filter, the use of the visibility feature, and the improved handling of noisy features. The "conference" scene highlights the benefits of the visibility feature. We preserve the shadows, while SBF, which does not use visibility features, is not able to do so. For this scene we used a larger window radius of $r = 20$ to remove spike noise. In the comparison with SBF we obtain a much lower MSE. In the "sanmiguel" scene, our filter yields a smoother result while preserving the foliage details as well as the small direct shadow details, which SBF blurs because of the absence of the visibility feature. In the "sibenik" scene we obtain results largely free of artifacts in out-of-focus regions, where typically all features are noisy. In contrast, SBF suffers from residual noise in these areas. In the "teapot-metal" scene, our filter can better preserve the chaotic structure of the floor, as well as the glossy highlights that are not captured in the feature buffers. The "dragonfog" scene includes participating media to demonstrate the flexibility of our approach in dealing with a variety of rendering effects, and was also filtered with a window radius $r = 20$. We also provide comparisons to the methods of Dammertz et al. [DSHL10], Bauszat et al. [BEM11], Kalantari and Sen [KS13], and Rousselle et al. [RKZ12] in the supplemental material.

Figure 8 illustrates our adaptive rendering scheme on the "sibenik" scene with an average of 16 samples per pixels. We also compare to SBF using the original authors' implementation. Adaptive sampling improves our MSE by roughly 18% compared to the result presented in Figure 7 with uniform sampling. The convergence plots of Figure 6 also indicate the MSE obtained with adaptive sampling with dotted lines, showing a consistent improvement over uniform sampling.

In Figure 9 we use the mean of the BRDF samples as a

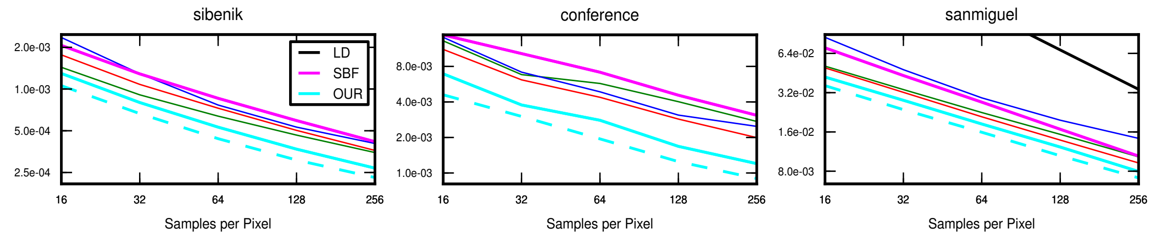

Figure 6: *Convergence plots for some of the scenes of Figure 7 (last page) for our method and the SURE-based approach by Li et al. [LWC12]. For our method, we show the errors of the FIRST, SECOND, and THIRD candidate filters and the final output. We indicate results from adaptive sampling with dotted lines.*

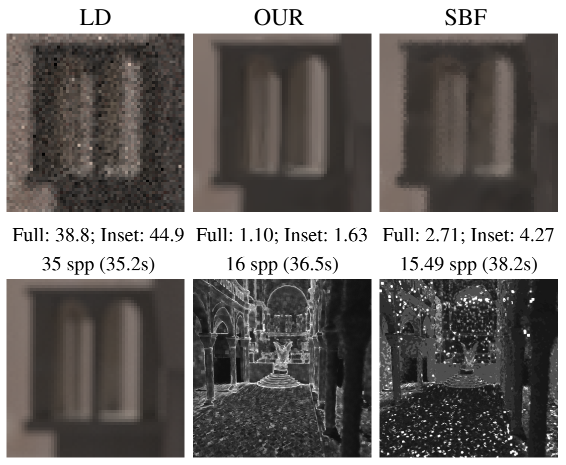

Figure 8: *Adaptive rendering of the "sibenik" scene at an average of* 16 *spp using both our method and SBF, including* MSE *values (multiplied by* 10*E*3*) and sampling density maps. The bottom left image is the reference.*

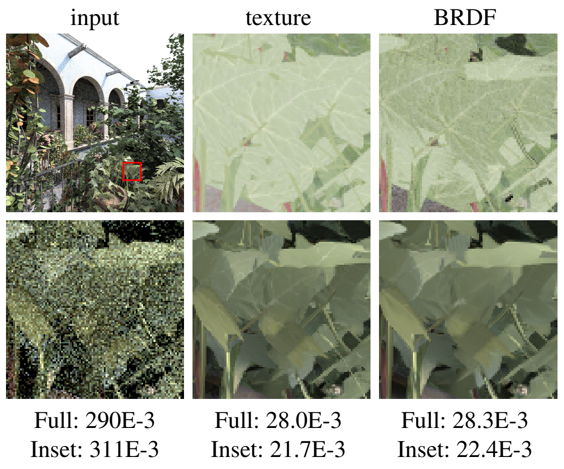

Figure 9: *Using the mean of BRDF samples instead of textures as a feature, with the "sanmiguel" scene at 32 samples per pixel. The feature buffers are given in the top row, with the corresponding filtered output and* MSE *values below. Noise in the BRDF samples may lead to loss of some fine texture details at low sampling rates.*

feature buffer instead of a texture. The BRDF value is readily available in a raytracer independent of rendered materials, whereas a texture may not be well-defined or easily extracted for many materials. While the mean of the BRDF samples is typically noisier than a texture, we can still handle this case with our feature prefiltering approach, although finer details may not be preserved as well.

A limitation of our approach is that at very low sampling rates, typically less than 10 spp, the color weights tend to become less useful and SURE-error estimation less reliable because of excessive variance in the color buffer. In such situations it is possible that our final filter using candidate filter averaging fails to improve the global MSE over the best candidate filter. At such low sampling rates it also becomes challenging to prefilter noisy features such as the visibility. In these cases the filter parameters need to be adjusted to obtain reasonable results. It is important to note, however, that all images in the paper and supplemental materials were rendered using fixed parameters as in Algorithm 1.

10. Conclusions

We have presented a method for denoising Monte Carlo renderings by constructing filters using a combination of color and feature information. We construct three candidate filters based on NL-means weights for color buffers and cross-bilateral weights for feature buffers. We determine robust averaging weights of the three candidate filters on a per-pixel basis using SURE error estimation. We also introduce a novel approach to dealing with noisy features using a pre-filtering step, and we apply it to new caustics and visibility features. Together, candidate filter weighting including color information, feature prefiltering, and the novel caustics and visibility features provide significant improvements in terms of both visual and numerical quality over the previous state-of-the-art. While the computational cost of our filter is insignificant in the context of off-line rendering, the technique is not suitable for interactive rendering. In the future, we will explore extensions of our approach targeted at real-time applications. It would also be interesting to further improve methods targeted at extremely low sampling rates.

Acknowledgements

This project was funded in part by the Swiss National Science Foundation under grant no. 200021_143886. San Miguel model by Guillermo M. Leal Llaguno of Evolución Visual, dragon from the Stanford 3D Scanning Repository, Sibenik cathedral by Marko Dabrovic and Mihovil Odak, gargoyle from INRIA via the AIM@SHAPE repository, and conference scene by Anat Grynberg and Greg Ward.

References

[BCM05] Buades A., Coll B., Morel J.: A review of image denoising algorithms, with a new one. *SIAM Journal on Multiscale Modeling and Simulation 4*, 2 (2005), 490–530. 2, 3

[BCM08] Buades A., Coll B., Morel J.: Nonlocal image and movie denoising. *International Journal of Computer Vision 76*, 2 (2008), 123–139. 7

[BEM11] Bauszat P., Eisemann M., Magnor M.: Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering (EGSR)) 30*, 4 (June 2011), 1361–1368. 2, 7

[DFKE07] Dabov K., Foi A., Katkovnik V., Egiazarian K.: Image denoising by sparse 3-d transform-domain collaborative filtering. *Image Processing, IEEE Transactions on 16*, 8 (Aug. 2007), 2080 –2095. 2

[DJ94] Donoho D. L., Johnstone I. M.: Ideal spatial adaptation by wavelet shrinkage. *Biometrika 81*, 3 (1994), pp. 425–455. 2

[DSHL10] Dammertz H., Sewtz D., Hanika J., Lensch H. P. A.: Edge-avoiding à-trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics* (2010), Eurographics Association, pp. 67–75. 2, 7

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

[EHDR11] Egan K., Hecht F., Durand F., Ramamoorthi R.: Frequency analysis and sheared filtering for shadow light fields of complex occluders. *ACM Trans. Graph. 30*, 2 (Apr. 2011), 9:1–9:13. 2

[ETH*09] Egan K., Tseng Y.-T., Holzschuch N., Durand F., Ramamoorthi R.: Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. 28* (July 2009), 93:1–93:13. 2

[HJW*08] Hachisuka T., Jarosz W., Weistroffer R. P., Dale K., Humphreys G., Zwicker M., Jensen H. W.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27* (August 2008), 33:1–33:10. 2

[HST10] He K., Sun J., Tang X.: Guided image filtering. In *Proceedings of the 11th European conference on Computer vision: Part I* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 1–14. 2

[Kat99] Katkovnik V.: A new method for varying adaptive bandwidth selection. *Signal Processing, IEEE Transactions on 47*, 9 (1999), 2567–2571. 2

[KS13] Kalantari N. K., Sen P.: Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum (Proceedings of Eurographics 2013) 32*, 2 (2013). 2, 7

[LAC*11] Lehtinen J., Aila T., Chen J., Laine S., Durand F.: Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30* (August 2011), 55:1–55:12. 2

[LALD12] Lehtinen J., Aila T., Laine S., Durand F.: Reconstructing the indirect light field for global illumination. *ACM Trans. Graph. 31*, 4 (July 2012), 51:1–51:10. 2

[LWC12] Li T.-M., Wu Y.-T., Chuang Y.-Y.: Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 194:1–194:9. 2, 4, 7, 8, 10

[McC99] McCool M. D.: Anisotropic diffusion for monte carlo noise reduction. *ACM Trans. Graph. 18*, 2 (April 1999), 171–194. 2

[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. *Computer Graphics Forum 32*, 1 (2013), 139–151. 2

[MWR12] Mehta S. U., Wang B., Ramamoorthi R.: Axis-aligned filtering for interactive sampled soft shadows. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 163:1–163:10. 2

[ODR09] Overbeck R. S., Donner C., Ramamoorthi R.: Adaptive wavelet rendering. *ACM Trans. Graph. 28* (December 2009), 140:1–140:12. 2

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. 7

[RKZ11] Rousselle F., Knaus C., Zwicker M.: Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 159:1–159:12. 2

[RKZ12] Rousselle F., Knaus C., Zwicker M.: Adaptive rendering with non-local means filtering. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 195:1–195:11. 2, 3, 4, 6, 7

[SAC*11] Shirley P., Aila T., Cohen J., Enderton E., Laine S., Luebke D., McGuire M.: A local image reconstruction algorithm for stochastic rendering. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 9–14 PAGE@5. 2

[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 9*, 6 (1981), pp. 1135–1151. 2, 4

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

[VDVK09] Van De Ville D., Kocher M.: Sure-based non-local means. *Signal Processing Letters, IEEE 16*, 11 (2009), 973–976. 2

[XP05] Xu R., Pattanaik S.: A novel monte carlo noise reduction operator. *Computer Graphics and Applications, IEEE 25*, 2 (2005), 31–35. 2

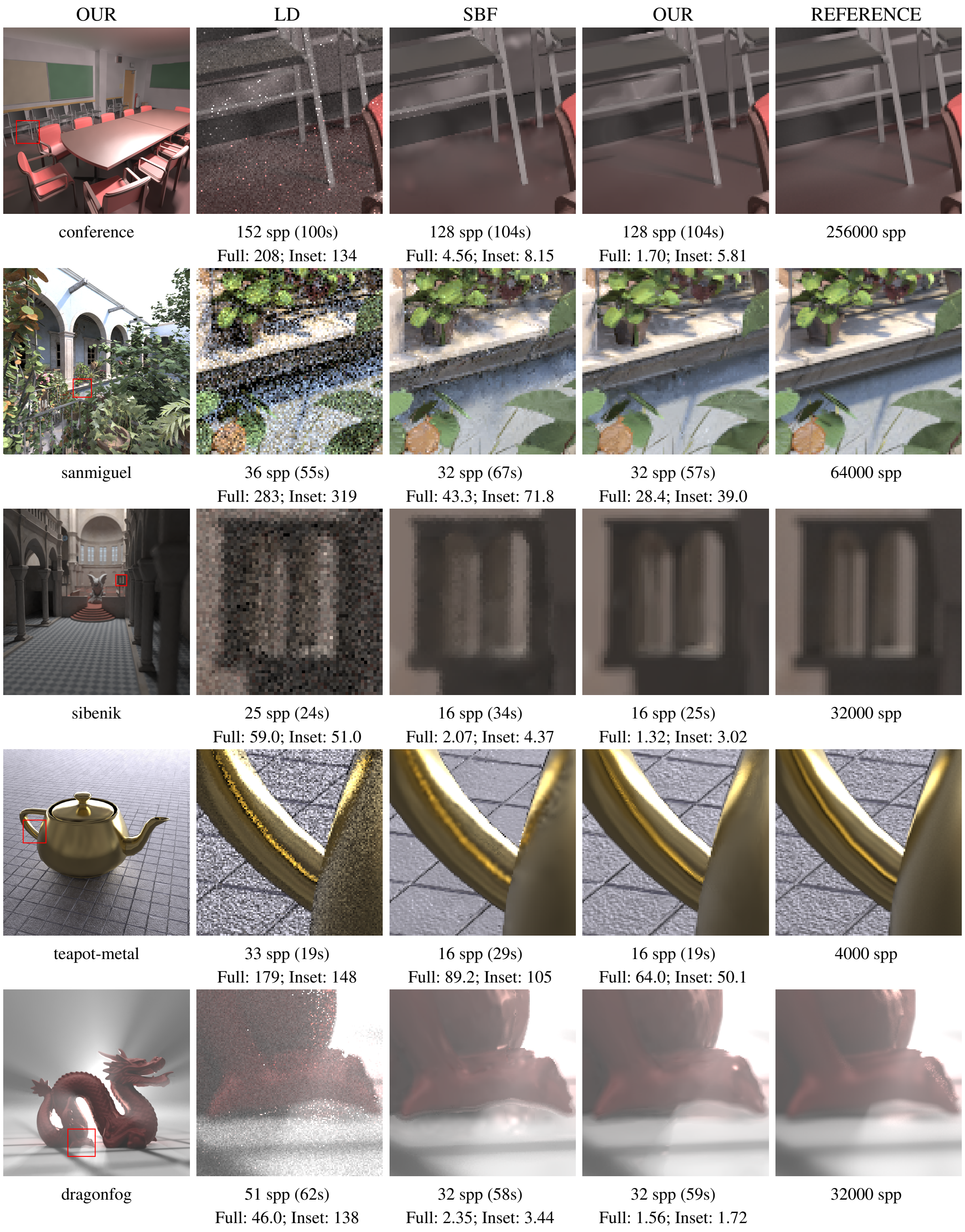

Figure 7: *Comparison of our approach (OUR) to path tracing with low-discrepancy (LD) and the SURE-based approach by Li et al. [LWC12] (SBF). All images are rendered at a resolution of* 1024 × 1024 *using PBRT at roughly equal render time for all methods. At the bottom of the images we give the number of samples per pixel (spp), rendering time in seconds, and* MSE *values multiplied by* 10*E*3.*We provide consistently better* MSE *values, which we attribute to three main factors: the consideration of the color buffer to construct the final filter, the use of visibility features, and the improved handling of noisy features.*