P-RPF: Pixel-based Random Parameter Filtering for Monte Carlo Rendering

Hyosub Park, Bochang Moon, Soomin Kim, Sung-Eui Yoon

Abstract

In this paper we propose *Pixel-based Random Parameter Filtering* (P-RPF) for efficiently denoising images generated from complex illuminations with a high sample count. We design various operations of our method to have time complexity that is independent from the number of samples per pixel. We compute feature weights by measuring the functional relationships between MC inputs and output in a sample basis. To accelerate this sample-basis process we propose to use an upsampling method for feature weights. We have applied our method to a wide variety of models with different rendering effects. Our method runs significantly faster than the original RPF, while maintaining visually pleasing and numerically similar results. Furthermore the performance gap between our method and RPF increases as we have more samples per pixel. As a result, our method shows more visually pleasing and numerically better results of RPF in an equal-time comparison.

Keywords: Random parameter filtering, Monte Carlo rendering

1. Introduction

Monte Carlo (MC) rendering such as path tracing [1] is one of the most general rendering techniques for producing physically-correct rendering results. It calculates color (i.e. radiance) of a pixel by generating and tracing random samples, ray paths, within the integration domain. Ray paths can have complex interactions with the scene being rendered, and are computed by considering various factors such as surface reflection functions, area light sampling, lens sampling, time sampling, and so on. Overall MC rendering is an effective method to solve a multidimensional integration function taking geometry and random parameters as inputs.

The very characteristic of MC rendering produces noise, when insufficient samples are used to estimate the true value. While the scene function is complex and integration domain is a high-dimensional space, we have only limited computation resource to sample these complex functions. Many attempts have been made to remove this noise in images generated by MC rendering.

A recent research focus is on designing effective image-space reconstruction methods, since image-space techniques are easy to implement, can be naturally integrated with existing rendering systems, and are highly efficient thanks to its image-space nature. Most image-space denoising techniques achieve high-quality results by considering various geometric features (e.g., depth, normal, and texture) within well-known filters [2, 3, 4, 5] such as joint bilateral filter.

Recently Random Parameter Filtering (RPF) [2] demonstrated impressive denoising results even with a small number of samples per pixel. A key characteristic that sets it apart from prior work is that it measures the functional relationship of colors and geometric features over any random parameters and then adjusts filtering factors of these features during joint bilateral filtering. This property of RPF enables exceptional results, since varying filtering factors can effectively deemphasize geometric features that are even noisy.

Its shortcoming, however, is the lack of scalability. It runs at a reasonable speed for eight samples per pixel, but it becomes drastically slower as the number of samples per pixel increases. This is because the time complexity of RPF algorithm is dependent on the number of samples per pixel. For scenes with complex illumination it may be impossible to capture most important light paths with low samples per pixel (Fig. 1). In these scenes a high number of samples even with reconstruction methods is required, and the current RPF technique may lost its competitive edge because of the low scalability.

Contributions. In this paper we propose *pixel-based random parameter filtering* (P-RPF) for efficiently denoising various rendering effects generated by MC rendering. Our method consists of three main steps: 1) initialization for pixel-based computation, 2) computing

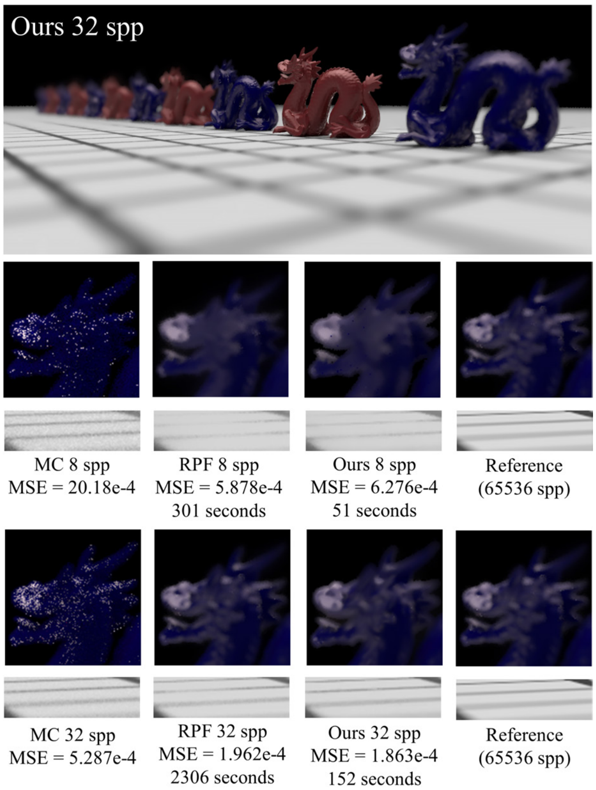


Figure 1: Filtering results of the dof-dragons scene using RPF and our method with 8 and 32 samples per pixel (spp). All the methods with 8 spp lack the information to preserve edges on the out-of-focused dragon's head and in-focused texture on the floor. Our method with 32 spp achieves visually pleasing results, while it runs even faster than RPF with 8 spp. In an equal-time comparison, our method with 32 spp shows three times lower MSE over RPF with 8 spp.

feature weights considering feature types with different filtering factors, and 3) performing joint bilateral filtering with the computed feature weights. Our final filtering operation is performed in a pixel-based approach. We further accelerate the component of computing feature weights, the main computational bottleneck, by using an upsampling technique, whose time complexity is also independent from the sample count. We have applied our method into a set of benchmarks that have different rendering effects. Overall we are able to achieve more than one order of magnitude improvement over the original RPF when we use 32 samples per pixel (spp), and the performance improvement goes higher, as the input image is created by more spp. Furthermore, we numerically verify that our method achieves similar denoising results compared to RPF given the same spp, while our method runs much faster. Specifically, the reconstruction error of our method in terms of the Mean Squared Error (MSE) is only within 10% to that of the original RPF. These results demonstrates the scalability as well as denoising quality of our method. Finally, given equal-time comparisons, our method shows visually better and numerically lower MSE results over RPF, because of its highly efficient and effective denoising process.

2. Related Work

In this section we review prior techniques directly related to our work.

2.1. MC Noise Filtering

Reducing noise in images generated by MC rendering has been actively studied in the field of rendering. To realize this goal many techniques have been proposed for improving the reconstruction and sampling processes of MC rendering, mainly in two approaches: reducing the source of MC noise and filtering MC noise.

One of the well-known examples for reducing the source of MC noise is multidimensional adaptive sampling and reconstruction method [6]. In addition, advanced reconstruction techniques based on a frequency-domain analysis have been designed for specific rendering effects such as depth-of-fields and motion blur [7, 8, 9].

As an early example of filtering MC noise, Rushmeier and Ward [10] proposed an energy preserving nonlinear filter that redistributes the color values of noisy pixels into their neighboring pixels. Jensen and Christensen [11] denoised images by separating light paths that are reflected diffusely two times and by then filtering them using the median filter. Xu and Pattanaik [12] pointed out that the direct application of bilateral filtering [13] cannot remove spike noise generated by MC rendering. To address this problem they used cross bilateral filtering with an edge stopping function that is a smoothed input image by Gaussian filtering. DeCoro et al. [14] introduced an outlier removal technique based on density estimation, which can be used as a preprocessing for various filtering methods. This outlier removal technique can be also used with our method, as a preprocessing tool to remove outliers.

In MC rendering the filtering process is often guided by the additional information (e.g. G-buffer) to perform edge-preserving filtering. McCool [15] introduced an anisotropic diffusion filter guided by additional information (e.g. depth, normal, and texture) easily obtained by MC rendering. Dammertz et al [16] used the Á-trous wavelet transform, and applied cross bilateral filtering using depth, normal, and texture to transformed low-resolution images. Since this method uses low-resolution images, an iterative filtering performance is achieved. Bauszat et al. [4] proposed a filtering process guided by geometric information, which filters out noise in indirect illumination generated by interactive path tracing. Recently, Moon et al. [3] proposed a virtual flash image constructed by considering a nearly noise-free part of light paths, and the image is used as an edge stopping function in non-local means.

2.2. Random Parameter Filtering (RPF)

Sen et al. [2] proposed RPF, which selectively uses different filtering factors on features used in joint bilateral filtering. The main idea of RPF is that MC noise occurs due to point sampling the scene function with various random parameters such as pixel position, lens position, and time. If dependence of geometric features and colors on random parameters can be evaluated, one can determine appropriate weights of those features in joint bilateral filtering. RPF accounts for possible corruptions of scene information due to distribution effects such as motion blur or depth-of-field. They can hence filter not only noise due to variance in light paths, but also noise due to difference in geometry. RPF computes different feature weights by measuring the mutual dependence between pixels, colors, features and random parameters.

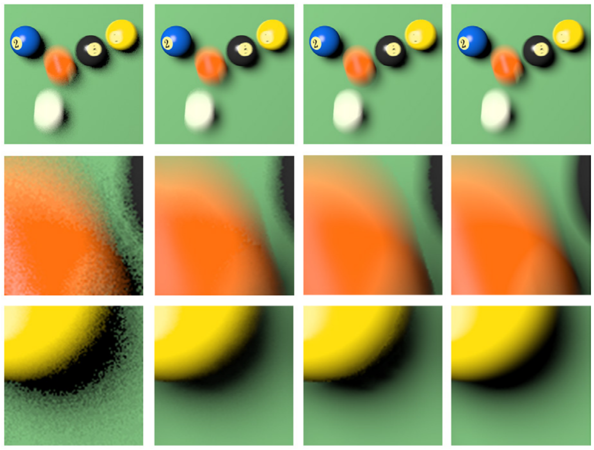
While RPF achieves impressive denoising results with a small number of samples per pixel, RPF requires a high computation cost. This is mainly because filtering each pixel requires thousands of neighboring samples and relies on sample-by-sample analysis. On the other hand, we perform various operations of our method in a pixel basis, while maintaining high denoising quality.

2.3. Bilateral Upsampling

Image upsampling has been well studied as one of the basic image operations [17]. In the field of rendering, upsampling has been mainly used for accelerating the computation of smoothly changing indirect illumination. Sloan et al. [18] used bilateral upsampling [13] to interpolate indirect shading using geometry information as an edge-stopping function. Ritschel et al. [5] also used bilateral upsampling of indirect illumination for interactively generating preview images. In our work, we apply joint-bilateral filter based upsampling to accelerate computing feature weights that are smoothly changing in large regions of images.

3. Overview of Our Approach

In this section we explain our motivations, followed by giving the overview of our approach.



Input (8 spp) MSE: 5.104e-4	RPF (107s) MSE: 2.204e-4	Ours (21s) MSE: 1.804e-4	Reference (8192 spp)

Figure 2: Comparisons of our method and RPF on the pool scene rendered by 8 samples per pixel (spp). Both our method and RPF handle motion blur (second row) and soft shadow with edges (third row), while our method runs five times faster and shows even a lower MSE over RPF with 8 spp.

3.1. Motivations

RPF is a reconstruction technique that considers different importance of feature types for images generated by MC rendering. RPF consists of three stages: selecting and preprocessing of neighboring samples, computing feature weights for joint bilateral filtering, and performing filtering.

RPF filters an input image four times in order to reduce variance as much as possible with different filtering window sizes, starting with 55 and decreasing into 35, 17, and 7 at each filtering step. This multi-pass approach of RPF effectively denoises global low-frequency noise first and then gradually removes more localized noise, thereby cleaning up noise while preserving details.

In addition, RPF provides a high quality filtering result even with a small number of ray samples (e.g. 8). Nonetheless, when input images are corrupted by severe noise, filtering results with a small number of ray samples can be unsatisfactory. For example, depth-of-field effects in Fig. 1 make over-blurred results on detailed geometry, when 8 spp is used. As the number of ray samples (e.g. 32 spp) increases, the detailed geometry can be preserved. It indicates that a relatively large number of ray samples can be required for achieving a high quality filtering result, when noise levels of input images are very high.

Unfortunately, the computation time of RPF is highly dependent on the number of ray samples. Fig. 3 shows performance curves of RPF for processing different mod-

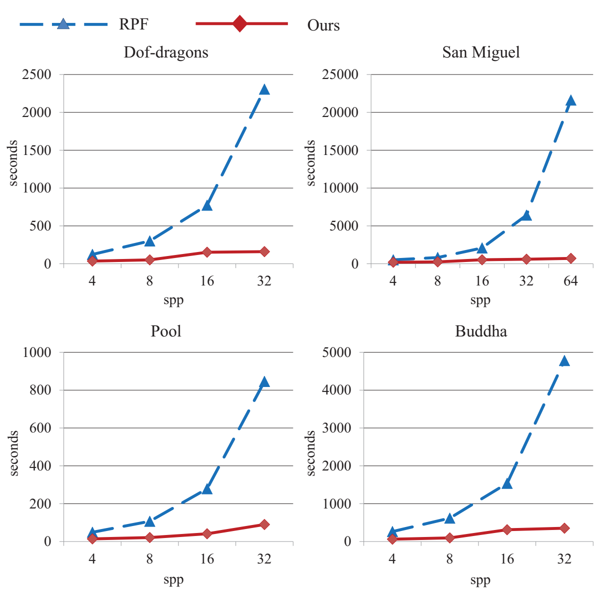


Figure 3: Timing results of RPF and our method. Our method is efficient even with high samples per pixel (spp). Note that the running time of our method with 32 spp is less than or equal to that of RPF with 8 spp.

els. As the number of samples per pixel increases, RPF becomes prohibitively slow, losing its key advantage of providing quality preview images within a short computation time.

3.2. Overall Algorithm

We introduce pixel-based random parameter filtering, which operates on pixels rather than samples, thereby efficiently producing high-quality denoised results. Its key advantage is that we perform various operations of our method in a pixel basis. Feature weight computation, which cannot be done pixel-based, is accelerated by using bilateral upsampling; we sparsely evaluate feature weights over the image, and estimate feature weights for the rest of the image using joint bilateral interpolation.

We first conduct various initialization including computing neighboring pixels and samples (Sec. 4.1), and feature normalization (Sec. 4.2) for a robust denoising process. We then prepare feature weights by directly measuring or interpolating from nearby pixels (Sec. 4.4). Based on those feature weights we finally perform joint bilateral filtering (Sec. 4.3). For the sake of clarity we provide a pseudocode of our pixel-based random parameter filtering in Algorithm 1, and summarize various notations (Table I) that we use throughout the paper.

Algorithm 1 Pixel-based Random Parameter Filtering

Input: Input image $\boldsymbol{I}$
Output: Final image
  for pixel i in image $\boldsymbol{I}$ **do**
    Precompute μ_i and σ_i
  end for
  Divide $\boldsymbol{I}$ into two sets $\boldsymbol{I_s}$ and $\boldsymbol{I_i}$ (Sec. 4)
  for iteration step $t = 0, 1, 2, 3$ **do**
    for each pixel in $\boldsymbol{I_s}$ **do**
      Construct neighboring pixels and samples (Sec. 4.1)
      Compute feature weights (Sec. 4.3)
      Perform filtering (Sec. 4.3)
    end for
    for each pixel in $\boldsymbol{I_i}$ **do**
      Construct neighboring pixels and samples (Sec. 4.1)
      Interpolate feature weights (Sec. 4.4)
      if interpolation is failed **then**
        Compute feature weights
      end if
      Perform filtering (Sec. 4.3)
    end for
  end for
  return final image

4. Our Method

We explain our reconstruction method in this section. Before going into the main filtering loop we first calculate the mean and standard deviations, μ_i and σ_i, of samples within a pixel i of an input image, $\boldsymbol{I}$. They are used for accelerating the computation of the mean and standard deviation of neighboring samples, which will be used for normalization of samples in Sec. 4.2.

We also decompose pixels of the image $\boldsymbol{I}$ into two disjoint sets, $\boldsymbol{I_s}$ and $\boldsymbol{I_i}$. $\boldsymbol{I_s}$ is a sparse set of pixels where we evaluate feature weights, while $\boldsymbol{I_i}$ is a set containing the rest of pixels, whose feature weights are interpolated by their nearby neighbors from $\boldsymbol{I_s}$. In our implementation pixels for $\boldsymbol{I_s}$ are uniformly distributed over the image such that they form a sub-sampled grid from the input image (Fig. 4).

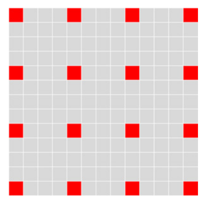


Figure 4: $\boldsymbol{I_s}$ (red) and $\boldsymbol{I_i}$ (grey).

4.1. Neighboring Pixels and Samples

We derive various information including feature weights from each pixel and perform reconstruction. When we have only a few samples in each pixel, information derived from these small sets of samples can be brittle and contain noise. In order to address this problem, given a pixel i, we define a set of neighboring pixels, P_i, and then derive such information robustly from the neighboring pixels P_i.

Table I. Notations used in this paper

s	Number of samples per pixel (spp).
w	Size of filtering window.
p_i	2×1 vector containing floating point pixel position (x, y) of i-th pixel.
c_i	3×1 vector containing the mean color of i-th pixel.
f_i	15×1 vector containing mean geometric features of i-th pixel.
v_i	27×1 feature vector containing all the feature info. including p_i, c_i, and f_i of i-th sample. For its full description, see Sec. 5. We use $v_{i,k}$ to denote the k-th dimension of the vector v_i.
μ_i	Mean vector of samples within i-th pixel.
σ_i	Standard deviation vector of samples within i-th pixel
$\bar{x}$	Denotes a normalized vector for x.(e.g. $\bar{p}_i$ and $\bar{c}_i$)
$\hat{\mu}_i$	Mean vector of neighboring samples of i-th pixel
$\hat{\sigma}_i$	Standard deviation vector of neighboring samples of i-th pixel
z_k	Tolerance parameter for selecting a neighbor when considering k-th dimension of the feature vector.

To construct neighboring pixels P_i given a pixel i we iterate all the pixels within its filtering windows and consider geometric features, stored in f_i. When the mean of pixel j is within $z_k \cdot \sigma_{i,k}$ from the mean of the current pixel i, we include the pixel j to the neighboring pixels P_i; $\sigma_{i,k}$ indicates k-th dimension of the standard deviation vector σ_i. z_k represents the relative tolerance for difference of f_i and f_j at the k-th dimension.

Once we define P_i we construct neighboring samples, S_i, of the pixel i by simply adding all the samples in every pixel $j \in P_i$. We will use neighboring samples S_i to derive mutual information between various variables for random parameter filtering.

Since we compute neighboring samples S_i indirectly from neighboring pixels P_i, some samples in S_i may not be in the range of $z_k \cdot \sigma_{i,k}$ from the mean of pixel i. Nonetheless those samples take only a minor portion on S_i (e.g., 5% to 10% on average). This is mainly because the various statistics derived from samples follow those derived from pixels well. Instead we could compute S_i by additionally checking whether each sample of a pixel from P_i is within $z_k \cdot \sigma_{i,k}$ from the mean of pixel i, and this sample-based alternative was adopted in the original RPF [2]. Fig. 5 shows feature weights and their

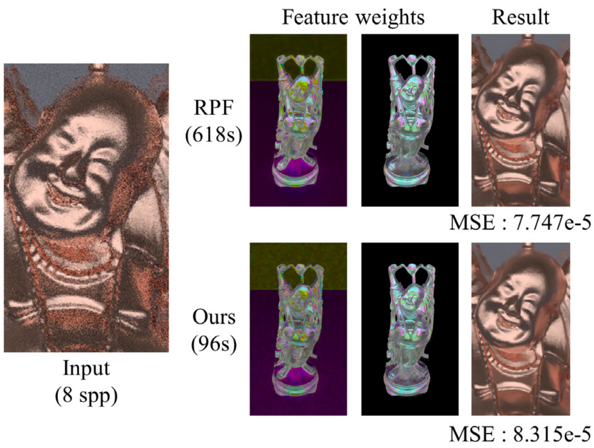


Figure 5: Comparison of feature weights computed by the original RPF and our method. The left ones of the feature weights are $\beta_{i,k}$ derived from world space coordinates, and the right ones are derived from normals. Feature weights computed in a pixel-basis are similar to that computed in a sample-basis.

corresponding denoising results based on our pixel-based definition of neighboring pixels/samples and that of the original RPF. As can be seen, the differences between our pixel-based and sample-based approaches in terms of computed feature weights and denoised results are subtle.

4.2. Feature Normalization

We normalize features after we construct neighboring samples S_i for a pixel i. This step is required, because features taken into account for filtering have different scales. For example, texture values are within the range [0,1], while world-space coordinate can be arbitrarily big. For normalizing features associated with the pixel i, we perform the statistical standardization, which subtracts the mean, $\hat{\mu}_i$, of neighboring samples S_i, and divide the resulting value by their standard deviation, $\hat{\sigma}_i$. We perform this process for each feature of every sample in S_i.

To perform feature normalization we need to compute $\hat{\mu}_i$ and $\hat{\sigma}_i$ for S_i given a pixel i. Instead of computing them based on samples of S_i, we can efficiently compute them based on pre-computed μ_j and σ_j of neighboring pixels $j \in P_i$ of the pixel i. Specifically, $\hat{\mu}_i$ can be computed as the following:

$$\hat{\mu}_i = \frac{\sum_{j \in P_i} \mu_j}{|\ P_i\ |}. \qquad (1)$$

We can also compute $\hat{\sigma_i}$ as the following:

$$\hat{\sigma_i} = \sqrt{\frac{\sum_{j \in S_i} (v_j - \hat{\mu_i})^2}{| S_i |}} = \sqrt{\frac{\sum_{j \in P_i} \sum_{k \in n_j} (v_k - \hat{\mu_i})^2}{| S_i |}}, \quad (2)$$

where n_j is a set containing indices of samples at pixel j. $\sum_{k \in n_j} (v_k - \hat{\mu_i})^2$ in the above equation can be reformulated as $s(\mu_j - \hat{\mu_i})^2 + \sum_{k \in n_j} (v_k - \mu_j)^2$, and $\sigma_j = \sqrt{\frac{\sum_{k \in n_j} (v_k - \mu_j)^2}{s}}$. If we plug these two equations into Eq. 2, we reach the following equation:

$$\therefore \hat{\sigma_i} = \sqrt{\frac{\sum_{j \in P_i} s(\sigma_j^2 + (\mu_j - \hat{\mu_i})^2)}{| S_i |}}. \quad (3)$$

As a result, we can efficiently calculate $\hat{\sigma_i}$ and $\hat{\mu_i}$ from σ_i and μ_i derived from each pixel i.

4.3. Joint Bilateral Filtering with Feature Weights

We use joint bilateral filtering to smooth out colors of pixels. The joint bilateral filter uses a filtering weight, w_{ij}, that measures a contribution of a pixel j within a filtering window to a pixel i, as the following:

$$\begin{aligned} w_{ij} = {} & \exp\Big(-\sum_{k=1}^{2} \frac{1}{2\sigma_{i,p_k}^2} (\bar{p}_{i,k} - \bar{p}_{j,k})^2\Big) \\ & \times \exp\Big(-\sum_{k=1}^{3} \frac{\alpha_{i,k}}{2\sigma_{i,c_k}^2} (\bar{c}_{i,k} - \bar{c}_{j,k})^2\Big) \\ & \times \exp\Big(-\sum_{k=1}^{|f|} \frac{\beta_{i,k}}{2\sigma_{i,f_k}^2} (\bar{f}_{i,k} - \bar{f}_{j,k})^2\Big), \end{aligned} \quad (4)$$

where $\bar{p}$, $\bar{c}$, and $\bar{f}$ are normalized values of pixel, color and geometric features. Also, σ_{i,p_k}, σ_{i,c_k}, and σ_{i,f_k} represent k-th elements corresponding to position p_i, color c_i, and geometric features f_i, respectively, within the standard deviation vector σ_i. $\alpha_{i,k}$ and $\beta_{i,k}$ are two different feature weights per pixel i, and denote the importance of k-th color and importance of k-th feature, respectively. In the same manner used in the original RPF [2], we define these two feature weights $\alpha_{i,k}$ and $\beta_{i,k}$ as follows:

$$\begin{aligned} \alpha_{i,k} &= \max(1 - 2(1 + 0.1t) W_{c,k}^r, 0), \\ \beta_{i,k} &= W_c^{f,k} \cdot \max(1 - (1 + 0.1t) W_{f,k}^r, 0), \end{aligned}$$

where $W_{f,k}^r$, $W_c^{f,k}$, and $W_{c,k}^r$ represent dependence of k-th geometric feature on random parameters, dependence of color on k-th geometric feature, and dependence of k-th color on random parameters, respectively. These dependence relationships are estimated by measuring mutual information between different variables. The mutual information is obtained by constructing histograms of each variable and joint histograms of related variables [2].

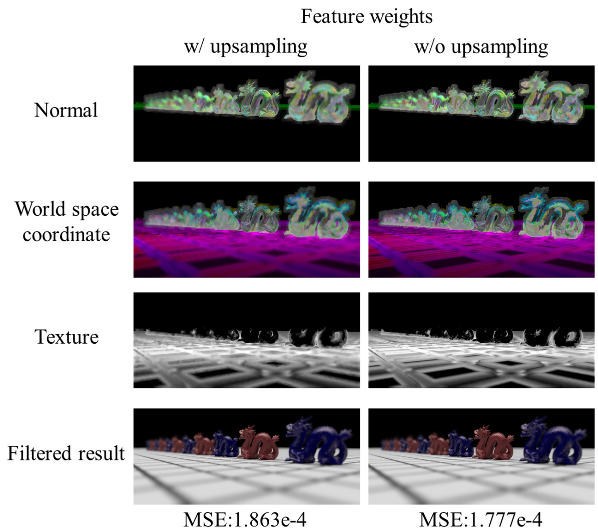


Figure 6: Comparisons between feature weights derived from normal, etc. w/ and w/o using our upsampling method By using our upsampling we achieve 8 to 10 times performance improvement in terms of computing feature weights and about 3 times improvement in terms of total computation time, without a significant quality drop on the final reconstruction results.

Note that these histograms are computed based on samples of geometric features, colors, etc. that are available at pixel i. As a result, computing feature weights can be a major computational bottleneck of our approach. To address this computational problem, we compute feature weights on a sparse set $\boldsymbol{I_s}$ of pixels and interpolate feature weights of other pixels $\boldsymbol{I_i}$ based on those computed for the sparse set. This process is explained in the next section.

4.4. Upsampling Feature Weights

Feature weights $\alpha_{i,k}$ and $\beta_{i,k}$ at each pixel i are highly likely to have correlations with geometric features $f_{i,k}$, colors $c_{i,k}$, and positions $p_{i,k}$, since those feature weights are derived from them. Exploiting this observation, we approximate feature weights of a pixel i by interpolating feature weights of its nearby pixels, while considering the difference in terms of features, colors, etc.

As shown in Algorithm 1, we first compute feature weights for pixels in $\boldsymbol{I_s}$. These are used for interpolating feature weights for pixels in $\boldsymbol{I_i}$. For each pixel in $\boldsymbol{I_i}$, t-nearest pixels in $\boldsymbol{I_s}$ are selected for interpolation. We have found that setting t to 16 strikes a good balance in terms of the performance and quality.

We perform interpolation by using the joint bilateral filter. In this framework, pixels that are more similar in terms of color and geometry have higher interpolation weights. Specifically, given a pixel i of $\boldsymbol{I_i}$, we define interpolation weights, iw_{ij} from nearest pixels j in $\boldsymbol{I_s}$ as the following:

$$
\begin{aligned}
iw_{ij} = & \exp(-\sum_{k=1}^{2} \frac{1}{2\sigma_{i,p_k}^2}(p_{i,k} - p_{j,k})^2) \\
& \times \exp(-\sum_{k=1}^{3} \frac{1}{2\hat{\sigma}_{i,c_k}^2}(c_{i,k} - c_{j,k})^2) \\
& \times \exp(-\sum_{k=1}^{15} \frac{1}{2(z_k \sigma_{i,f_k})^2}(f_{i,k} - f_{j,k})^2).
\end{aligned}
$$

Note that we use unnormalized values of $p_{i,k}$, $c_{i,k}$, and $f_{i,k}$ for computing interpolation weights, since i and j can be located far away and computation based on the normalized values that are standardized within each pixel can be invalid in this context.

Let $\alpha_{j,k}$ to be a feature weight value directly computed for a pixel j in $\boldsymbol{I_s}$. Using computed interpolation weights, the feature weight $\alpha_{i,k}$ at a pixel i in $\boldsymbol{I_i}$ is computed as follows:

$$
\alpha_{i,k} = \frac{\sum(iw_{ij} \times \alpha_{j,k})}{\sum iw_{ij}}.
$$

$\beta_{i,k}$ is defined also in a similar manner.

In the case where $\sum iw_{ij} \simeq 0$, $\alpha_{i,k}$ results in unacceptable values. This indicates that joint bilateral interpolation cannot approximate the feature weight of the pixel well. In this case, its feature weight should be directly computed. Specifically, when $\sum iw_{ij} \leq 0.1$, we directly compute its feature weight. Once we directly compute or estimate feature weights $\alpha_{i,k}$ and $\beta_{i,k}$ per pixel based on interpolation, we perform joint bilateral filtering (Eq. 4) with them.

Fig. 6 shows feature weights (and their corresponding reconstruction results) w/ and w/o upsampling feature weights. On average our joint bilateral interpolation works successfully for 85% to 94% of total pixels, which gives 8 to 10 times speedup in terms of computing feature weights. MSE of feature weights computed w/ and w/o upsampling is in the range from 0.001 to 0.002. The quality degradation on reconstructed images due to upsampling in terms of MSE is minor, less than 0.00001.

5. Results and Comparisons

We have implemented our method and the original RPF method [2] on top of PBRT2 [19]. To faithfully implement the original RPF, we followed detailed comments of its technical report [20]. We have tested our method and compared methods on a machine with two Intel quad-cores of Xeon X5690 3.47 GHz.

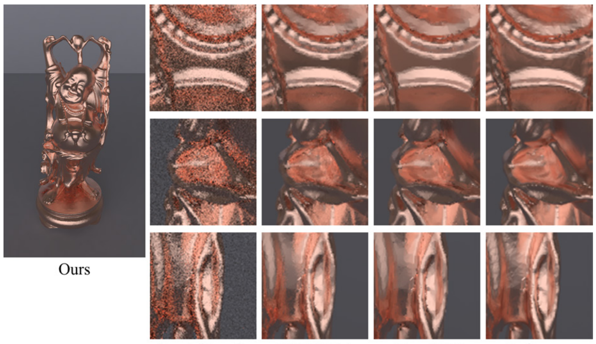


Figure 7: Comparisons under an equal sample count, i.e. 8 spp. Our method achieves visually similar filtering results over RPF, while running five times faster.

Each sample v that we process is a 27 dimensional vector containing 2D pixel coordinate, 3D color, geometric features, and random parameters. Geometric features include world-space coordinate, shading normal, and texture values for the first intersection of primary rays, and world-space coordinate and shading normal of the second intersection. Random parameters used for sampling include the area light information, lens positions, and time at the first and second intersections. For upsampling, we directly compute feature weights for every 5 by 5 pixels, and attempt to estimate for other pixels based on joint bilateral interpolation. z_k values used for defining neighboring pixels are set to 3 for all the feature types except for the world space coordinate. We set z_k to 30 for world space coordinates, since its range is much bigger than other feature types, by following the guideline of RPF [2].

Benchmarks.. We have tested our algorithm on various scenes with different rendering effects. The buddha model (Fig. 7) has a highly glossy material with a 720 X 1280 image resolution; we pick the default image resolution of scenes chosen by PBRT2 system and show it in a parenthesis for other models. Fig. 2 shows a pool scene (512 X 512) with the motion blur effect. Fig. 8 and Fig. 1 show the San Miguel (1024 X 1024) and dof-dragons (1000 X 424) scenes rendered with the depth-of-field effect, respectively. All the scenes are rendered with path tracing except for the pool scene, which is rendered by direct lighting.

5.1. *Qualitative Comparisons*

The San Miguel scene (Fig. 8) is geometrically complex and shows numerically high MC errors, when path tracing is used. The scene becomes an even more challenging benchmark with the depth-of-field effect. This is evident from the fact that the reference image generated with 16 k samples per pixel (spp) still contains a large amount of noise. Overall both our algorithm and RPF with 8 spp show over-blurring results on edge regions (the fourth zoomed region from the top of Fig. 8). These over-blurring results indicate that 8 spp is not enough to capture most of the details of the scene. Reconstructed results from 64 spp preserve the boundary of shadow (the third zoomed region from the top in Fig. 8) and subtle details caused by the distribution effect (4th zoomed region). In this case with 64 spp, RPF takes more than 6 hours to process 64 spp, which is unacceptable for a preview creation purpose. Our method, however, takes 707 seconds, even faster than 8 spp reconstruction of RPF and 30 times faster than RPF with 64 spp. In an equal-time comparison, our method achieves 46% lower MSE over RPF, because of its higher scalability.

The dof-dragons scene (Fig. 1) is another case tested with the depth-of-field effect. There is a noticeable difference between reconstruction results with 8 spp 32 spp on this scene. The BRDF of the dragon model is complex that 8 spp cannot capture a sufficient amount of information for a proper reconstruction. This results in over-blurring, which does not preserve subtle details caused by the depth-of-field effect. Reconstruction results from 32 spp are more visually pleasing and numerically better, while 32 spp still produces a very noisy input. In an equal-time comparison, our method with 32 spp produces visually pleasing and numerically better results, three times lower MSE, over RPF with 8 spp, which is even two times slower than our method with 32 spp.

Fig. 7 shows the results of RPF and our method with 8 spp on the buddha scene. Noise caused by the area light and glossy material is well removed, while keeping geometric details of the buddha model. This is a scene where RPF was effective even with 8 spp, where our approach achieved similar results, while taking only one fifth of running time of RPF.

Fig. 2 compares the performance of RPF and our approach on the pool scene, where motion blur due to movement of pool balls is present. Both methods work well for motion-blurred regions (the second row) and rather static regions exhibiting sharp edges (the third row) with soft shadow due to area lights. Nonetheless our algorithm filters the scene more than 5 times faster than RPF, while achieving a similar level of MSE.

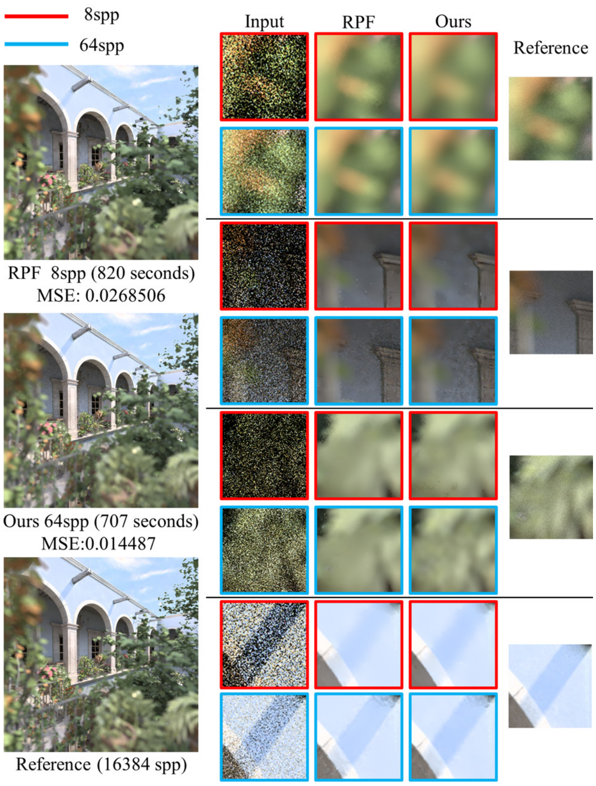


Figure 8: Equal-time comparisons on the San Miguel scene with the depth-of-field effect. Our method with 64 spp provides visually pleasing and numerically lower results over RPF with 8 spp, which is 16% slower than our method with 64 spp.

5.2. *Quantitative Results*

Fig. 3 shows the timing result of our method compared with the original RPF. Our method shows a much faster performance than the original RPF, while the gap between our approach and RPF increases as high spp is used for generating input images. This result comes mainly from performing various operations in a pixel-basis, not in a sample-basis. On average the total computation time of our method with 32 spp is less than that of RPF with 8 spp. In most cases our algorithm is 4 to 6 times faster than RPF, when spp is less than or equal to 16. When spp is higher than 16, our algorithm is faster by a factor of more than one order of magnitude than RPF.

We also measure a breakdown of components of our method. In the case of 32 spp, computation time for 1) constructing neighboring pixels and samples, 2) computing or estimating feature weights, and 3) filtering takes in a ratio of 6:3:1. Constructing neighboring samples

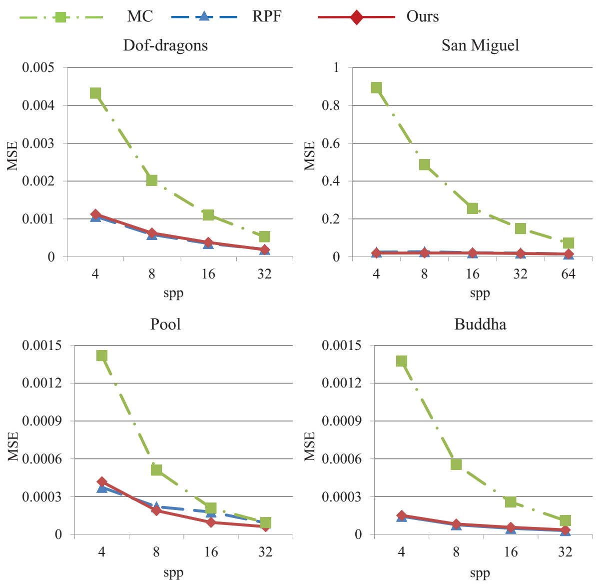


Figure 9: MSE results of different methods. MSE of our method is similar to that of RPF.

and pixels, the simplest part of our algorithm, is the main computational bottleneck, since it includes the normalization process, which is done sample-by-sample. The filtering process takes only a minor portion of computation time, because it is done purely on a pixel basis. As a result, filtering takes a less portion among the total computation, as the number of samples per pixel increases. In the case of RPF, on the other hand, the ratio of computation time of similar operations is 2:4:4 on average. This is mainly because RPF computes feature weights and perform filtering in a sample basis.

Fig. 9 shows error analysis of our method compared with RPF and MC rendering. As spp increases, errors of both RPF and our method consistently decrease. In general, MSE of our method is similar to that of RPF. This demonstrates that our method effectively denoises MC input images like RPF.

6. Conclusion

We have introduced pixel-based random parameter filtering that processes and filters samples on the pixel basis, instead of the sample basis. Our approach accelerates the feature computation stage, which cannot be operated on the pixel basis, by using the upsampling approach. We have compared our method over the original RPF across a diverse set of models and demonstrated that our method effectively denoises input images like RPF. Furthermore, given equal-time comparisons, our method shows visually pleasing and numerically lower MSE results over RPF, because of its higher efficiency.

6.1. Limitations and Future Work

Our method also has limitations. Since our work is based on RPF, it inherits drawbacks of RPF. Notably, our method still has the *dueling filter* problem, where we need to use large filter bandwidths to smooth out noise while keeping sharp edges. As a failure case of our method, our method leaves noise out in the second row of zoomed images on the right of Fig. 8, while blurring edges (the first row) generated by 32 spp. In addition, by upsampling feature weights, our method tends to generate more visually blurry results over RPF, especially when feature weights of pixels contain high-frequency information.

There are many interesting avenues for future research, in addition to addressing the limitation of our approach. Our pixel-based reconstruction method can be naturally combined with various adaptive sampling methods. Furthermore, a low computational overhead of our method makes our approach more suitable to be integrated with an adaptive sampling process. To allocate more samples to where our reconstruction fails, we would like to design a new adaptive scheme tailored to our reconstruction method. In order to guide more samples on high error regions, we would like to employ an error estimation process for our reconstruction method. This is a very challenging problem, but should enable an effective adaptive rendering as well as addressing the dueling filter problem by considering the error during our reconstruction method, as conducted in recent adaptive rendering approaches [21, 22, 23]. In addition, we would like to apply the error estimation process such that it can be used to automatically select currently manually chosen filtering parameters of our reconstruction method. Specifically, Stein's unbiased risk estimator [21] can be utilized to estimate optimal parameters so that MSE introduced by our filtering is minimized.

References

[1] J. T. Kajiya, The rendering equation, in: D. C. Evans, R. J. Athay (Eds.), Computer Graphics (SIGGRAPH '86 Proceedings), Vol. 20, 1986, pp. 143–150.

[2] P. Sen, S. Darabi, On filtering the noise from the random parameters in monte carlo rendering, ACM Trans. Graph. 31 (3) (2012) 18:1–18:15.

[3] B. Moon, J. Y. Jun, J. Lee, K. Kim, T. Hachisuka, S.-E. Yoon, Robust image denoising using a virtual flash image for monte carlo ray tracing, Computer Graphics Forum.

[4] P. Bauszat, M. Eisemann, M. Magnor, Guided image filtering for interactive high-quality global illumination, Computer Graphics Forum 30 (4) (2011) 1361–1368.

[5] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, C. Dachsbacher, Micro-rendering for scalable, parallel final gathering, in: ACM SIGGRAPH Asia, 2009, pp. 1–8.

[6] T. Hachisuka, W. Jarosz, R. P. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, H. W. Jensen, Multidimensional adaptive sampling and reconstruction for ray tracing, in: ACM SIGGRAPH, 2008, pp. 33:1–33:10.

[7] C. Soler, K. Subr, F. Durand, N. Holzschuch, F. Sillion, Fourier depth of field, ACM Trans. Graph. 28 (2) (2009) 18:1–18:12.

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

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

[10] H. E. Rushmeier, G. J. Ward, Energy preserving non-linear filters, in: ACM SIGGRAPH, 1994, pp. 131–138.

[11] H. Jensen, N. Christensen, Optimizing path tracing using noise reduction filters, in: J. Winter School of Computer Graphics (WSCG), 1995, pp. 134–142.

[12] R. Xu, S. N. Pattanaik, A novel Monte Carlo noise reduction operator, IEEE Comput. Graph. Appl. 25 (2) (2005) 31–35.

[13] C. Tomasi, R. Manduchi, Bilateral filtering for gray and color images, in: ICCV, IEEE Computer Society, 1998, p. 839.

[14] C. DeCoro, T. Weyrich, S. Rusinkiewicz, Density-based outlier rejection in Monte Carlo rendering, Computer Graphics Forum 29 (7) (2010) 2119–2125.

[15] M. D. McCool, Anisotropic diffusion for Monte Carlo noise reduction, ACM Trans. Graph. 18 (2) (1999) 171–194.

[16] H. Dammertz, D. Sewtz, J. Hanika, H. P. A. Lensch, Edge-avoiding A-Trous wavelet transform for fast global illumination filtering, in: High Performance Graphics, 2010, pp. 67–75.

[17] T. Blu, M. Unser, Image interpolation and resampling, in: Handbook of Medical Imaging, Processing and Analysis, Academic Press, 2000, pp. 393–420.

[18] P.-P. Sloan, N. K. Govindaraju, D. Nowrouzezahrai, J. Snyder, Image-based proxy accumulation for real-time soft global illumination, IEEE Comput. Graph. Appl. (2007) 97–105.

[19] M. Pharr, G. Humphreys, Physically Based Rendering: From Theory to Implementation 2nd, Morgan Kaufmann Publishers Inc., 2010.

[20] P. Sen, S. Darabi, Implementation of Random Parameter Filtering, Tech. Rep. EECE-TR-11-0004, University of New Mexico (May 2011).

[21] T.-M. Li, Y.-T. Wu, Y.-Y. Chuang, Sure-based optimization for adaptive sampling and reconstruction, ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2012) 31 (6) (2012) 186:1–186:9.

[22] F. Rousselle, C. Knaus, M. Zwicker, Adaptive sampling and reconstruction using greedy error minimization, in: SIGGRAPH Asia, 2011, pp. 159:1–159:12.

[23] F. Rousselle, C. Knaus, M. Zwicker, Adaptive rendering with non-local means filtering, ACM Trans. Graph. 31 (6) (2012) 195:1–195:11.