# On Filtering the Noise from the Random Parameters in Monte Carlo Rendering

PRADEEP SEN and SOHEIL DARABI
UNM Advanced Graphics Lab

Monte Carlo (MC) rendering systems can produce spectacular images but are plagued with noise at low sampling rates. In this work, we observe that this noise occurs in regions of the image where the sample values are a direct function of the random parameters used in the Monte Carlo system. Therefore, we propose a way to identify MC noise by estimating this functional relationship from a small number of input samples. To do this, we treat the rendering system as a black box and calculate the statistical dependency between the outputs and inputs of the system. We then use this information to reduce the importance of the sample values affected by MC noise when applying an image-space, cross-bilateral filter, which removes only the noise caused by the random parameters but preserves important scene detail. The process of using the functional relationships between sample values and the random parameter inputs to filter MC noise is called *random parameter filtering* (RPF), and we demonstrate that it can produce images in a few minutes that are comparable to those rendered with a thousand times more samples. Furthermore, our algorithm is general because we do not assign any physical meaning to the random parameters, so it works for a wide range of Monte Carlo effects, including depth of field, area light sources, motion blur, and path-tracing. We present results for still images and animated sequences at low sampling rates that have higher quality than those produced with previous approaches.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Raytracing*

General Terms: Rendering

Additional Key Words and Phrases: Monte Carlo rendering, global illumination

**ACM Reference Format:**

Sen, P. and Darabi, S. 2011. On Filtering the Noise from the Random Parameters in Monte Carlo Rendering. ACM Trans. Graph. XX, X, Article XXX (September 2011), 14 pages.
DOI = 10.1145/XXXXX.XXXXX
http://doi.acm.org/10.1145/XXXXXXX

Pradeep Sen and Soheil Darabi acknowledge support from National Science Foundation CAREER award IIS-0845396.
Authors' addresses: psen@ece.unm.edu, sdarabi@gmail.com


DOI 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY


Fig. 1. Our algorithm takes as input a small set of Monte Carlo samples, which are fast to compute but very noisy. We then estimate the functional dependency between the sample values and the random parameters used in the rendering system, which enables us to filter out the MC noise without blurring scene detail. The result is comparable to a rendering with a large number of samples but more than $100\times$ faster. This path-traced image shows the result of our method along with sections of the input rendering at 8 samples/pixel and the reference at 8,192 samples/pixel for comparison.

## 1. INTRODUCTION

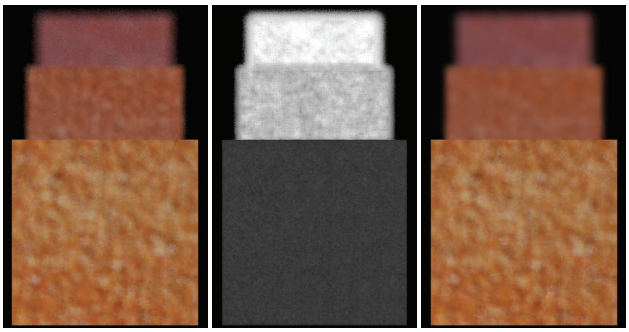
Monte Carlo (MC) rendering systems can produce beautiful, photo-realistic images by simulating light transport through a series of multidimensional integrals at every pixel of the image: integration of the radiance over the aperture of the camera, over the area light sources of the scene, over the time the shutter is open, etc. For a pixel in the image $I(i,j)$, this process can be written as:

$$I(i,j) = \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} \int_{j-\frac{1}{2}}^{j+\frac{1}{2}} \cdots \int_{-1}^{1} \int_{-1}^{1} \int_{t_0}^{t_1} f(x,y,\cdots,u,v,t)\, dt\, dv\, du \cdots dy\, dx.$$

MC renderers estimate these integrals by taking many point samples of the scene function $f()$, a black-box, functional representation of the ray-tracing system given a specific scene. This sampling process involves tracing rays with sets of random parameters that correspond to the dimensions of integration, e.g., $t$ (the moment in time of the ray for motion blur), $u$ and $v$ (the position of the ray on the aperture of the camera for depth of field) and so on.

If we evaluate the scene function at enough of these multi-dimensional samples, the MC rendering system will converge to the actual value of the integral, resulting in a physically correct image. Unfortunately, the variance of the estimate of the integral decreases as $O(1/N)$ with the number of samples, so while we can get a noisy approximation within a few minutes (as shown in the left inset of Fig. 1), we usually need a long time (as much as a day per image) to get a result that is acceptable for high-end rendering applications, shown in the right inset. This limits the use of Monte Carlo rendering systems in modern digital film production.

An obvious way to try to address this problem is to apply a noise-reduction filter to the noisy image. Indeed, this approach has been explored by researchers in the past but with limited success. The fundamental problem is that filters cannot easily determine what is unwanted noise (introduced by the MC integration process) and what is valid scene content. To see why, we consider the application of a bilateral filter [Tomasi and Manduchi 1998], which preserves



**(a)** Input MC (8 spp) **(b)** Dependency on $(u, v)$ **(c)** Our approach (RPF)

Fig. 2. **(a)** Depth-of-field (DoF) scene rendered at 8 samples/pixel (spp). The MC noise is similar to the noisy texture of the in-focus quad, so a bilateral filter that removes the noise will also remove this detail. **(b)** Our approach estimates the functional dependency of the sample color on the random position on the lens, $(u, v)$. The in-focus quad is dark because its color is not a function of $(u, v)$, but the dependency increases as the quads get more out of focus. **(c)** Result of our algorithm, which applies a bilateral filter to the samples in (a) but weights the color using the dependency in (b).

edges in the image by blending samples using weights computed from the differences in position as well as sample value, thereby avoiding blending samples together whose values differ greatly.

Unfortunately, bilateral filters work poorly for filtering general Monte Carlo noise, such as that in Fig. 2a. This depth-of-field (DoF) scene has three quads with noisy textures, with only the closest quad in focus. When rendered at 8 samples/pixel, the MC integration process produces noise in the blurred regions because rays from a pixel in these areas hit different parts of the quads and therefore have widely varying colors. A bilateral filter that uses the sample color to preserve the texture detail in the in-focus quad would also preserve the DoF noise, because here the color variations for both are very similar. This is why previous methods that use scene features for bilateral filtering do not work for general MC effects.

In this work, we propose a way to use bilateral filters to effectively reduce general MC noise while preserving scene detail. We begin with the observation that sample features are functions of the random parameters in the noisy regions of the image. In a DoF scene, for example, any ray through the lens from a focused point $x, y$ on the image will land on the same point on the scene, regardless of where on the lens the random point is located. So if the surface is diffuse, the color of these samples will not be a function of the random lens position (i.e., the scene function $f()$ is constant with respect to $u, v$ for these $x, y$) so these regions are not noisy. For pixels out of focus, on the other hand, the random lens position affects the final color of the sample. Here, the sample color will be a function of the random parameters, which is why these regions look noisy in the first place (the random parameters act as a noise generator)[1]. Our key insight is that if we estimate these functional relationships, we can reduce the importance of features that depend on random parameters during bilateral filtering to reduce MC noise.

[1]MC rendering systems always compute perfect, noise-free samples of scene function $f()$ given a set of random parameters. The noise we are referring to here lies in the differences (i.e., variance) between sample values, which affect both the computation of the final pixel value and bilateral filtering. Although the presence of noise requires $f()$ to vary with respect to the random parameters, the source of the noise is still the random parameters themselves. After all, using uniform parameters for integration replaces the noise with banding artifacts, which have different sample variance. We discuss this in more detail in the technical report [Sen and Darabi 2011b].

Unfortunately, finding the functional relationship between sample features and the random parameters in closed, mathematical form is impossible for complex scenes. Furthermore, finding where $f()$ is constant with respect to the random parameters is not easy with a small number of samples, since each sample has varying $x, y$ as well as $u, v$. This makes it difficult to determine whether differences in sample values are caused by changes in the random parameters $u, v$ or by changes in the image position $x, y$. For this reason, we propose to treat the inputs and outputs of the scene function $f()$ as random variables and to estimate the functional relationships by looking for statistical dependencies between them, using the concept of *mutual information* from the field of information theory.

In the example of Fig. 2, our approach estimates the statistical dependency of the sample color on its random position on the lens $(u, v)$, as shown in Fig. 2b. For the in-focus quad, since color is not a function of $(u, v)$ we consider it to be scene detail that should be preserved by making the color important during bilateral filtering. When out of focus, the color is a function of $(u, v)$ and so it is "corrupted" by noise. Therefore, the color in these regions is given low importance when bilaterally filtering the samples. Although the color here varies as much as in the in-focus regions, we still blend samples to produce smooth results, as shown in Fig. 2c.

This same principle works for other Monte Carlo effects. For a scene with motion blur, for example, the parts of the image with motion-blur noise are the ones where the sample features are a function of the random time parameter $t$. Those that do not depend on $t$ are static and hence do not need to be filtered. We demonstrate the extension of this simple idea to area light sources, path-tracing, Russian roulette, and other MC effects in this paper.

To summarize our contribution, we observe that traditional filters do not work for MC denoising because scene features vary between samples for two very different reasons: 1) scene content varying as a function of screen position (which we want to preserve) and 2) random parameters of the Monte Carlo process causing noise artifacts (which we want to filter away). We present an algorithm that distinguishes between the two using a small number of samples, thereby allowing us to filter only the noise from the random parameters. We call this algorithm *random parameter filtering* (RPF).

## 2. PREVIOUS WORK

The rendering community has been interested in Monte Carlo approaches since the seminal work by Cook et al. [1984] extended the Whitted ray tracer [1980] to produce effects such as depth of field, soft shadows, etc. Since then, there has been significant effort to address the noise problem. We cannot thoroughly review all previous work here, so we refer interested readers to texts on advanced rendering (e.g., [Dutré et al. 2006; Pharr and Humphreys 2010]).

**Algorithms to filter MC noise –** Shortly after the introduction of stochastic rendering methods, Lee and Redner [1990] proposed using nonlinear filters such as alpha-trimmed filters (which discard statistical outliers and average the remaining samples) for this purpose. Rushmeier and Ward [1994] presented a nonlinear filter that spreads out the contribution of "noisy" samples to smooth out the signal. They identify the noisy samples by finding pixels where the variance is still above a threshold after a certain amount of time. Jensen and Christensen [1995] described a method of filtering Monte Carlo renderings by filtering the low-frequency indirect illumination separately from the rest of the image.

McCool [1999] developed a filter based on anisotropic diffusion [Perona and Malik 1990] that preserves details in the image using a map of image coherence with color, depth, and normal information. Xu and Pattanaik [2005] proposed a modified bilateral

filter to compare the range values of a Gaussian-filtered version of the image. Others have proposed filtering global illumination using a geometry-based discontinuity buffer [Keller 1998] to adjust a filter [Segovia et al. 2006; Laine et al. 2007]. More recently, Dammertz et al. [2010] proposed the edge-avoiding À-Trous filter that incorporates a wavelet formulation into the bilateral filter. They, too, add additional information such as normal and world position to help identify edges in the scene.

Overall, the problem with these approaches is that scene information such as normals and world positions can be corrupted by MC noise in effects such as depth of field and motion blur, so filters that rely on variations in these values to preserve scene detail cannot denoise these kinds of scenes. This is why these approaches have all focused on denoising irradiance or other forms of global illumination, where the geometry scene information at each sample is unaffected by the random parameters. Our approach, on the other hand, can handle general MC effects with the same framework.

**Algorithms to reduce the source of MC noise** – Researchers have also studied the source of the noise in MC rendering in order to develop algorithms to mitigate the problem. Early on, Mitchell [1991] examined how to extend non-uniform sampling patterns from 2D to the number of dimensions of the random parameters in order to improve the quality of the final image. Other researchers introduced new Monte Carlo-based rendering algorithms with lower variance, such as irradiance caching [Ward et al. 1988], photon-mapping [Jensen 2001], and multidimensional lightcuts [Walter et al. 2006]. Others reduced the noise by fitting a smooth basis to the noisy data (e.g., [Meyer and Anderson 2006]).

Researchers have also studied the multidimensional sampling and reconstruction problem. Hachisuka et al. [2008] proposed the multidimensional adaptive sampling (MDAS) algorithm, which adaptively samples the space in all parameter dimensions. MDAS can handle a wide range of MC effects but suffers from the curse of dimensionality as the number of parameters grows. Overbeck et al. proposed another general method known as adaptive wavelet rendering (AWR) [2009], which positions samples based on the variance of a wavelet basis's scale coefficients and reconstructs the final image using a wavelet approximation. This smooths noisy areas and preserves detail, although it produces wavelet artifacts when the sampling rate is low. This work also claims to distinguish between the two sources of image-space variance (scene features and MC noise) using the wavelet coefficients. However, their proposed method would not work for the example of Fig. 2, since in image-space the MC noise here is similar to the noisy texture detail.

There is also work that uses transform domain analysis to optimize the adaptive placement of samples for specific MC effects. For example, Soler et al. [2009] used the Fourier domain to efficiently render depth-of-field effects, while Egan et al. [2009] leveraged frequency-space analysis to develop a sheared filter and sampling method for motion blur. Unlike these methods, our algorithm is general and can be applied to various effects. Recently, Sen and Darabi [2010; 2011a] used compressed sensing to reconstruct scene signal $f()$ assuming that it is sparse in a transform domain. This last method is not an adaptive-sampling algorithm, but a post-process reconstruction like our own. Unfortunately, it still needs a considerable number of samples to produce good results.

Finally, our approach uses information theory to measure the statistical dependencies between the inputs and outputs of the Monte Carlo rendering system. Information theory has been applied to improve ray tracing in the past using adaptive methods (e.g., [Sbert et al. 2007]), where the entropy of the color or geometry is used to determine the rendering quality of a part of the image.

Table I. Notation used in this paper

| | |
|---|---|
| $n$ | number of random parameters used to render the scene |
| $m$ | number of scene features available in the feature vector |
| $s$ | number of samples per pixel |
| $\mathbf{x}_i$ | sample vector containing the features of the $i^{\text{th}}$ sample |
| $\mathbf{p}_i$ | the floating-point $(x, y)$ position of the $i^{\text{th}}$ sample on the screen |
| $\mathbf{r}_i$ | $n \times 1$ vector of random parameters used to compute the $i^{\text{th}}$ sample |
| $\mathbf{f}_i$ | $m \times 1$ vector of scene features associated with the $i^{\text{th}}$ sample |
| $\mathbf{c}_i$ | original color vector of the $i^{\text{th}}$ sample |
| $\bar{\mathbf{v}}$ | vector with mean removed and normalized by standard deviation |
| $D^{\mathbf{v}}_{\mathbf{q}}$ | dependency of $\mathbf{q}$ on $\mathbf{v}$ |
| $W^{\mathbf{r}}_{\mathbf{f},k}$ | fractional contribution of *all* random parameters on the $k^{\text{th}}$ scene feature |
| $W^{\mathbf{f},k}_{\mathbf{c}}$ | fractional contribution of the $k^{\text{th}}$ scene feature on *all* color channels |
| $w_{ij}$ | weight that sample $j$ contributes to sample $i$ ($w_{ij} \neq w_{ji}$). |
| $\mathbf{c}'_i$ | final filtered color vector of the $i^{\text{th}}$ sample |
| $\mathcal{P}$ | a pixel in the image representing a set of $s$ samples |
| $\mathcal{N}$ | set that defines the neighborhood of samples used for filtering the samples |

## 3. RANDOM PARAMETER FILTERING ALGORITHM

We now present the theoretical framework for our algorithm, which is at its core a cross-bilateral filter that reduces the importance of sample scene features based on their dependence on the random parameters. Table I summarizes the notation used in this paper. In the depth-of-field example in Sec. 1, we had only two random parameters (the position on the lens) but in this discussion we generalize the problem to a Monte Carlo integration with $n$ random parameters, given for each sample as a vector $\mathbf{r} = \{r_1, r_2, \ldots, r_n\}$. In this way, the ray tracer in a typical MC rendering system calculates:

$$\mathbf{c}_i \Leftarrow f(\underbrace{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}}_{\substack{\text{screen}\\ \text{position}}}; \underbrace{\mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,n}}_{\substack{\text{random}\\ \text{parameters}}}),$$

where $f$ is scene dependent and takes as its only inputs the screen position of the sample $\mathbf{p}_i$ and the random parameters $\mathbf{r}_i$, and outputs the sample color[2]. In our approach, our bilateral filter uses the sample color value as well as other scene-dependent features for each sample (e.g., the world position, normals, and texture values of the ray intersection with the scene) to preserve scene detail. Therefore, instead of outputting color, our rendering system outputs sample vectors $\mathbf{x}_i$ that contain all relevant information for each sample: $\mathbf{x}_i \Leftarrow f(\mathbf{p}_{i,1}, \mathbf{p}_{i,2}; \mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,n})$, where (1)

$$\mathbf{x}_i = \{\underbrace{\mathbf{p}_{i,1}, \mathbf{p}_{i,2}}_{\substack{\text{screen}\\ \text{position}}}; \underbrace{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,n}}_{\substack{\text{random}\\ \text{parameters}}}; \underbrace{\mathbf{f}_{i,1}, \ldots, \mathbf{f}_{i,m}}_{\substack{\text{scene}\\ \text{features}}}; \underbrace{\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \mathbf{c}_{i,3}}_{\substack{\text{sample}\\ \text{color}}}\}.$$

Our approach filters the color of samples $\mathbf{x}_i$ using a weighted bilateral filter in which the importance of the color and scene features is adjusted to reflect their dependence on the random parameters:

$$\begin{aligned} w_{ij} = &\exp[-\frac{1}{2\sigma^2_{\mathbf{p}}} \sum_{1\leq k\leq 2} (\bar{\mathbf{p}}_{i,k} - \bar{\mathbf{p}}_{j,k})^2] \times \\ &\exp[-\frac{1}{2\sigma^2_{\mathbf{c}}} \sum_{1\leq k\leq 3} \alpha_k (\bar{\mathbf{c}}_{i,k} - \bar{\mathbf{c}}_{j,k})^2] \times \\ &\exp[-\frac{1}{2\sigma^2_{\mathbf{f}}} \sum_{1\leq k\leq m} \beta_k (\bar{\mathbf{f}}_{i,k} - \bar{\mathbf{f}}_{j,k})^2], \end{aligned} \quad (2)$$

where $w_{ij}$ is the contribution (or weight) of the $j^{\text{th}}$ sample to the $i^{\text{th}}$ sample during filtering. The bars above $\bar{\mathbf{p}}$, $\bar{\mathbf{c}}$, and $\bar{\mathbf{f}}$ indicate they have been normalized as described in Sec. 4.3.2. The first term reduces the weight based on the distance between samples in image-space by subtracting their screen positions, while the second does the same with their color values as in a standard bilateral filter. The third term compares the differences of a specific scene feature,

[2]Technically, the ray tracer outputs the radiance for the given sample, but we use the term "color" throughout the paper to simplify our explanation.

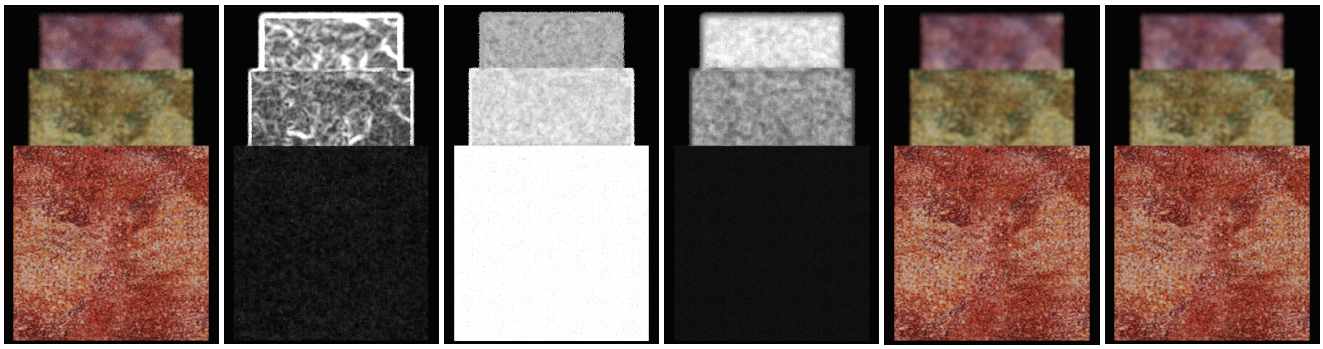



Fig. 3. In this scene, the front quad is focused so its sample values are not dependent on the random parameters $u, v$ but are a function of screen position. The out-of-focus quads are just the opposite. The fractional contribution of the random parameters on the color $W^{\mathbf{r}}_{\mathbf{c}}$ is used to calculate $\alpha_k$, which applies a standard bilateral filter to preserve detail for the quad in focus and approximates a Gaussian filter for the out-of-focus quads where the color is not important.

so that texture values, surface normals, etc., can be used to separate samples when they are not themselves functions of the random parameters. The $\alpha_k$ and $\beta_k$ terms specify the importance of the $k^{\text{th}}$ color channel/feature based on its dependency on the random parameters. Note that Eq. 2 is essentially a weighted cross-bilateral [Eisemann and Durand 2004] or joint-bilateral [Petschnigg et al. 2004] filter that uses additional scene feature information to preserve scene detail. As mentioned in Sec. 2, similar filters have been proposed before for Monte Carlo rendering (e.g., [Dammertz et al. 2010]). The subtle, but fundamental, difference is our introduction of the $\alpha_k$ and $\beta_k$ terms that reduce the importance of features that are functions of the random parameters during the filtering process. We now discuss how we compute these two terms.

As evidenced by Eq. 1, the sample features $\mathbf{f}_i$ for a scene are only functions of the random parameters $\mathbf{r}_i$ and the screen position $\mathbf{p}_i$, both selected by the renderer. High variation in these scene features represents valid scene detail, which we want to preserve, when they are only functions of screen position. However, when they are only functions of the random parameters, the variation is due to MC noise because the random number generator that produces these parameters acts as a noise source in Eq. 1. In these cases, the variation should be ignored by our cross-bilateral filter. We represent the functional dependency of the $k^{\text{th}}$ scene feature on all random parameters as $D^{\mathbf{r}}_{\mathbf{f},k}$, and on screen position as $D^{\mathbf{p}}_{\mathbf{f},k}$. Our bilateral filter should ignore features whose variation has been affected by MC noise, so we do this by computing the fractional contribution of the random parameters to this particular feature:

$$W^{\mathbf{r}}_{\mathbf{f},k} = \frac{D^{\mathbf{r}}_{\mathbf{f},k}}{D^{\mathbf{r}}_{\mathbf{f},k} + D^{\mathbf{p}}_{\mathbf{f},k}}, \tag{3}$$

This expression tells us how much the $k^{\text{th}}$ feature was affected by the random parameters as a fraction of the contributions from both sets of inputs, with the reasonable assumption that the position and random parameters are statistically independent. When the sample feature is a function only of the random parameters, this value will be close to 1, and when it is dependent only on the screen position it will be 0. In the common case where we have some contribution from both inputs (e.g., a partially out-of-focus object is dependent on both screen position and $u, v$), Eq. 3 simply interpolates between the two. Once we know the impact the random parameters had on a scene feature, we can use this to compute our $\beta_k$ term:

$$\beta_k = 1 - W^{\mathbf{r}}_{\mathbf{f},k}, \tag{4}$$

which gives scene features that are dependent on the random parameters less importance in the cross-bilateral filter. We should also exclude scene features that do not contribute to the final color (e.g., a shader might not use the surface normal, so differences in the normal should be ignored), so we multiply Eq. 4 with a $W^{\mathbf{f},k}_{\mathbf{c}}$ term that tells us how much the sample color depends on a specific feature:

$$\beta_k = W^{\mathbf{f},k}_{\mathbf{c}}(1 - W^{\mathbf{r}}_{\mathbf{f},k}). \tag{5}$$

We describe how to compute $W^{\mathbf{f},k}_{\mathbf{c}}$ in Sec. 4.4.1. Similar terms can be calculated for $\alpha_k$ using the dependencies of sample color:

$$W^{\mathbf{r}}_{\mathbf{c},k} = \frac{D^{\mathbf{r}}_{\mathbf{c},k}}{D^{\mathbf{r}}_{\mathbf{c},k} + D^{\mathbf{p}}_{\mathbf{c},k}}, \tag{6}$$

$$\alpha_k = 1 - W^{\mathbf{r}}_{\mathbf{c},k}. \tag{7}$$

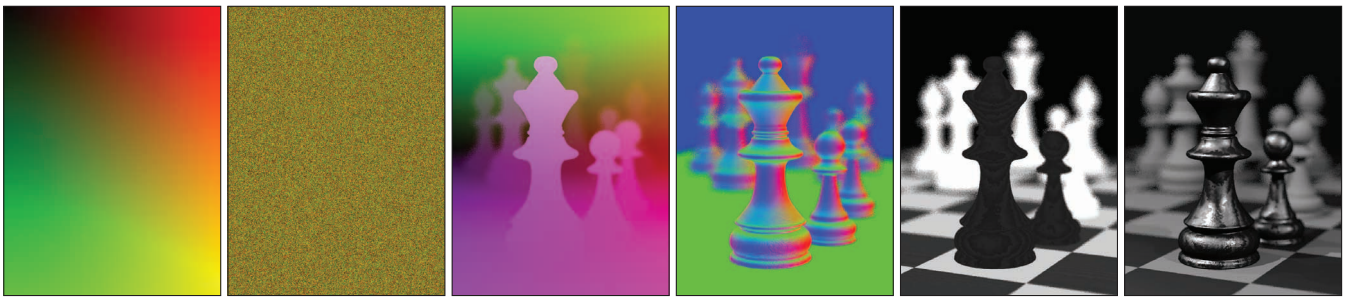
Fig. 3 shows a didactic example that visualizes some of the terms in our formulation. In our implementation, we use 3-channel RGB color and compute the weights for each channel separately. Note that although the samples' image positions can be chosen randomly as well (e.g., in jittered sampling), we treat them differently from the random parameters because we want to preserve features that are functions of the image position. A similar observation was made by Mitchell [1991], who noted that parameter dimensions such as $t$, $u$, and $v$ play a different role than image coordinates $x$ and $y$. Since our filtering process effectively integrates across the random parameter dimensions but does not filter detail that varies in $x$ and $y$, we must integrate over $x$ and $y$ ourselves after filtering the samples to compute the final pixel color. We now describe one of the key steps of our approach: how to estimate the functional relationships.

### 3.1 Estimating functional relationships between Monte Carlo inputs and outputs

Since it is difficult to derive an exact functional relationship between scene features and the inputs of the rendering system $\mathbf{p}_i$ and $\mathbf{r}_i$ for complex scenes, we propose instead to see if there is a statistical dependency (i.e., does knowing the inputs tell us something about the scene features). This is the basic meaning of mutual information, a concept from the field of information theory [Cover and Thomas 2006], which is the exact measure of dependence between two random variables and indicates how much information one tells us about another. The mutual information between two random variables $X$ and $Y$ can be calculated as:

$$\mu(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}, \tag{8}$$

where these probabilities are computed over the neighborhood of samples $\mathcal{N}$ around a given pixel. We describe the implementation details (including how we compute Eq. 8) in the next section.



**(a)** Screen position **(b)** Random parameters **(c)** World space coords. **(d)** Surface normals **(e)** Texture value **(f)** Sample color

Fig. 4. We visualize the elements of the sample vectors for the CHESS scene (with only DoF) by averaging them for each pixel and displaying them as colors. **(a)** The screen position $(\mathbf{p}_i)$ of the sample. **(b)** The two random parameters used to evaluate the $u, v$ position on the lens for depth of field, represented by $\{\mathbf{r}_{i,1}, \mathbf{r}_{i,2}\}$. Because these random parameters are selected uniformly in the range from [0,1], their average in each pixel is close to a constant color (0.5, 0.5, 0) with some noise. **(c)** The world coordinates for each sample. The $x$-axis is tilted toward the right, the $y$-axis is up, and the $z$-axis points out of the page and off to the left. **(d)** The surface normals for each sample. We show absolute value here to help emphasize the bump map texture on the black queen. **(e)** The texture value for each sample. **(f)** The sample color average, which is the result from traditional Monte Carlo if we simply box filter the samples together. Our algorithm computes the statistical dependency of scene features in **(c - f)** on the random parameters in **(b)**. Visually, we see that the regions with Monte Carlo noise are those where the scene features have similar structure to the random parameters, i.e., they are predominantly a function of these random parameters.

## 4. ALGORITHM IMPLEMENTATION

In this section, we outline the key implementation details of our algorithm, following the simplified pseudocode in Alg. 1. We refer those interested in implementing RPF to the technical report [Sen and Darabi 2011b] for more information and complete pseudocode.

### 4.1 Rendering samples and creating feature vectors

Because our denoising algorithm is a post-process filter, we first render the samples at a fixed sampling density and store vector $\mathbf{x}$ for each sample. For the scene features in $\mathbf{f}$, our algorithm stores for each sample the normal, world-space position, and texture values (the set of floats from texture lookups used by the surface shader to produce the surface color) for the first intersection point of the ray, and the world position and normal for the second intersection in a path tracer. The same features are stored for every scene, even if an object does not have the specific feature (a zero is substituted) or if the shader does not use the feature when computing the final color (features that do not affect the final color are ignored, as described earlier). Since all these features are available to the rendering system at some point during the tracing of the ray, outputting the feature vector for the sample is simply a matter of caching the information after it is calculated. This is standard practice in rendering systems when creating a G-buffer [Saito and Takahashi 1990] for deferred shading [Deering et al. 1988]. Fig. 4 shows a visualization of the feature vector for the CHESS scene.

### 4.2 Applying multiple filter iterations

To estimate the functional dependencies of sample values on the inputs to the MC rendering system using mutual information, we must select a set of samples to process. We cannot use every sample in the image because the functional dependencies change from region to region (e.g., an image may have some regions in focus and others out of focus, and these have different dependencies on the random parameters). Therefore, as we loop over every pixel in the image, we select a local neighborhood of samples $\mathcal{N}$ around that pixel to measure the local statistics for mutual information. However, we need to decide how big to make the block size that defines the extent of neighborhood $\mathcal{N}$.

If we use a large block size, there will be more samples to calculate statistics (improving the accuracy of our dependency estimates) and provide us with more samples to filter out noise. Unfortunately, larger block sizes have less locality and might cause problems when the block overlaps regions with different functional dependencies, such as regions where the amount of defocus blur changes. To resolve these two competing considerations, we found it best to use a multi-pass approach, where our algorithm loops over the image several times using different block sizes. We start at a larger block size and then shrink it down in a series of iterations. We found four iterations to be sufficient, starting with a block width of 55 pixels and then going down to 35, 17 and finally 7. At each step, we filter the samples' colors with the weighted bilateral filter of Eq. 2 using the samples in $\mathcal{N}$, and then use that new filtered color in the next pass of the algorithm (except to compute statistical dependencies, since they are always computed with the original sample color).

By going from larger to smaller, we first address the low-frequency noise that a smaller filter kernel would leave behind and then, as we reduce the block size, we eliminate the localized noise and clean up the detail. The multi-pass approach also reduces the maximum block size needed for filtering, since we can emulate a larger filter by progressively applying a smaller kernel. This allows us to get good performance and quality at the same time.

**Algorithm 1** Random Parameter Filtering (RPF) Algorithm

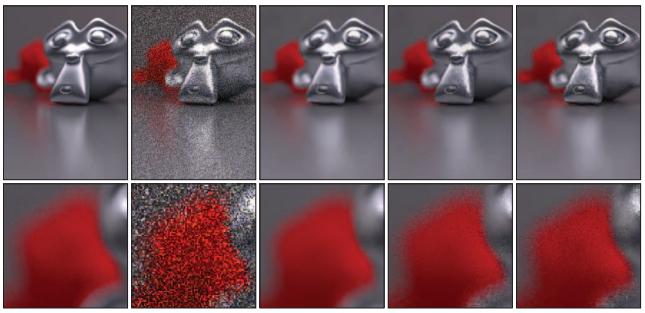
**Input:** scene to render and $s$ the number of samples/pixel
**Output:** final image

```
Render scene with s samples/pixel, output vector x for every sample (Sec. 4.1)
for iteration step t = 0, 1, 2, 3 (Sec. 4.2) do
    for all pixels P in image I do
        Preprocess samples in neighborhood N of P based on filter size (Sec. 4.3)
        Estimate statistical dependency of sample color/features on inputs p_i and
        r_i for samples in N, use them to compute weights α_k and β_k (Sec. 4.4)
        Filter samples' color in P using weighted bilateral filter of Eq. 2 (Sec. 4.5)
    end for
end for
After all samples are filtered, box filter each pixel to compute final pixel color
return final image
```

### 4.3 Preprocessing the samples

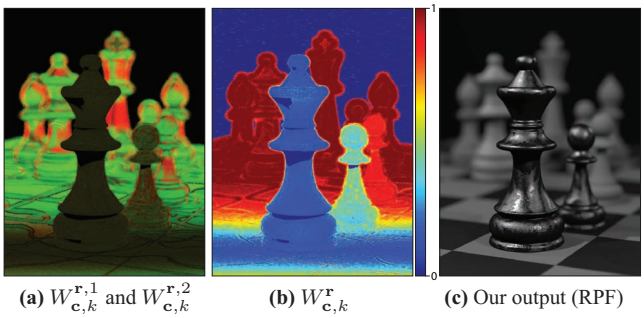
Before we use the samples in neighborhood $\mathcal{N}$ to compute statistical dependencies, we must perform some preprocessing steps.

4.3.1 *Clustering samples to avoid mixing statistics.* As mentioned earlier, a square block of pixels could overlap different regions of the scene, some that depend on the random parameters and some that do not. In these cases, if we simply compute the



**(a)** Reference **(b)** MC Input **(c)** RPF **(d)** no clustering **(e)** no DoF params

Fig. 5. This MONKEY HEADS scene shows the importance of clustering samples. **(a)** Reference MC at 8,192 samples/pixel. **(b)** Input Monte Carlo at 8 samples/pixel. **(c)** Random Parameter Filtering (RPF) with the complete clustering from Sec. 4.3.1. **(d)** RPF without clustering. The region where the two heads overlap is not properly denoised because sample statistics are getting mixed. **(e)** If the random parameters for lens position are removed from the sample vector, the algorithm can no longer denoise DoF.




**(a)** $W^{\mathbf{r},1}_{\mathbf{c},k}$ and $W^{\mathbf{r},2}_{\mathbf{c},k}$ **(b)** $W^{\mathbf{r}}_{\mathbf{c},k}$ **(c)** Our output (RPF)

Fig. 6. We visualize the dependency of color on the random point on the lens for the scene in Fig. 4. **(a)** Fractional contribution of the random $u$ and $v$ parameters ($W^{\mathbf{r},1}_{\mathbf{c},k}$ and $W^{\mathbf{r},2}_{\mathbf{c},k}$) shown in the red and green color channels, respectively. Blurry horizontal edges have more dependency on $u$, while blurry vertical edges have more dependency on $v$. **(b)** A visualization of $W^{\mathbf{r}}_{\mathbf{c},k}$ from Eq. 6. Areas that are more out of focus have higher values, which indicate larger blur filters. **(c)** Final output of our algorithm.

statistics using every sample in the block, the statistical dependency on the random parameters would be washed out, causing artifacts as shown in Fig. 5. For this reason, we only use samples with statistical properties similar to the pixel being filtered.

To do this, we compute the average feature vector $\mathbf{m}^{\mathbf{f}}_{\mathcal{P}}$ and the standard deviation vector $\boldsymbol{\sigma}^{\mathbf{f}}_{\mathcal{P}}$ for each component of the feature vector for the set of samples $\mathcal{P}$ at the current pixel. We then create our neighborhood $\mathcal{N}$ using only samples whose features are all within 3 standard deviations of the mean for the pixel. This helps us reduce the problem with mixing statistics. The underlying assumption here is that the statistics of the current pixel are a fair representation of that part of the image, which means that we need enough samples at every pixel to do the initial statistics. We have found experimentally that our algorithm works with 4 samples/pixel and becomes more robust as more samples are added.

4.3.2 *Normalizing scene features.* Before we compute the statistical dependencies for a set of samples in a neighborhood, we must first remove the mean and divide by the standard deviation for each of the elements in the sample vector. The reason for this is that the features in $\mathbf{f}$ reside in very different coordinate systems (world positions could be in the range of 0 to 1000, while the normal vector could have components in the range of 0 to 1, for example). If we do not correct for this discrepancy, we would inadvertently give larger weight to certain features when calculating dependency that may not necessarily be more important. This procedure is quite common in the machine learning community as well, since they are confronted with a similar problem [Hastie et al. 2001]. This is also related to Mahalanobis distance [Mahalanobis 1936], but in this case it is as if we assume that the covariance between scene features is zero, resulting in a diagonal covariance matrix. We represent vectors that have been normalized in this manner with a bar (e.g., $\mathbf{f}$ becomes $\bar{\mathbf{f}}$) in the equations in Sec. 3.

## 4.4 Computing dependencies and filter weights

4.4.1 *Estimating statistical dependencies on inputs.* First, we describe how we calculate the dependency of the $k^{\text{th}}$ scene feature on all random parameters ($D^{\mathbf{r}}_{\mathbf{f},k}$) using mutual information. Ideally, we would like to compute $D^{\mathbf{r}}_{\mathbf{f},k}$ using the joint mutual information $\mu(\mathbf{r}_{\mathcal{N},1}, \mathbf{r}_{\mathcal{N},2}, \ldots, \mathbf{r}_{\mathcal{N},n}; \mathbf{f}_{\mathcal{N},k})$, which tells us how much information all $n$ random parameters give us about $\mathbf{f}_k$ (the $\mathcal{N}$ notation indicates that the mutual information is measured over the set of samples in the neighborhood). Unfortunately, joint mutual information can be difficult and expensive to compute as the number $n$ gets larger, so our heuristic approximates this instead by measuring the dependency on individual random parameters and adding them up. Although this underestimates the total statistical dependency (see Sec. 5.3 of the technical report), in practice it gives us reasonable results quickly. Therefore, we first calculate the statistical dependency between $k^{\text{th}}$ scene feature and the $l^{\text{th}}$ random parameter by $D^{\mathbf{r},l}_{\mathbf{f},k} = \mu(\bar{\mathbf{f}}_{\mathcal{N},k}; \bar{\mathbf{r}}_{\mathcal{N},l})$, and then approximate the dependency of the $k^{\text{th}}$ scene feature on all $n$ random parameters as:

$$D^{\mathbf{r}}_{\mathbf{f},k} = \sum_{1 \le l \le n} D^{\mathbf{r},l}_{\mathbf{f},k} = \sum_{1 \le l \le n} \mu(\bar{\mathbf{f}}_{\mathcal{N},k}; \bar{\mathbf{r}}_{\mathcal{N},l}). \tag{9}$$

The dependency of the $k^{\text{th}}$ scene feature on screen position ($D^{\mathbf{p}}_{\mathbf{f},k}$) and color dependencies $D^{\mathbf{r}}_{\mathbf{c},k}$ and $D^{\mathbf{p}}_{\mathbf{c},k}$ are similarly computed:

$$D^{\mathbf{p}}_{\mathbf{f},k} = \sum_{1 \le l \le 2} D^{\mathbf{p},l}_{\mathbf{f},k} = \sum_{1 \le l \le 2} \mu(\bar{\mathbf{f}}_{\mathcal{N},k}; \bar{\mathbf{p}}_{\mathcal{N},l}), \tag{10}$$

$$D^{\mathbf{r}}_{\mathbf{c},k} = \sum_{1 \le l \le n} D^{\mathbf{r},l}_{\mathbf{c},k} = \sum_{1 \le l \le n} \mu(\bar{\mathbf{c}}_{\mathcal{N},k}; \bar{\mathbf{r}}_{\mathcal{N},l}),$$

$$D^{\mathbf{p}}_{\mathbf{c},k} = \sum_{1 \le l \le 2} D^{\mathbf{p},l}_{\mathbf{c},k} = \sum_{1 \le l \le 2} \mu(\bar{\mathbf{c}}_{\mathcal{N},k}; \bar{\mathbf{p}}_{\mathcal{N},l}). \tag{11}$$

4.4.2 *Calculating filter weights $\alpha_k$ and $\beta_k$.* The dependencies for the $k^{\text{th}}$ color channel $D^{\mathbf{p}}_{\mathbf{c},k}$ and $D^{\mathbf{r}}_{\mathbf{c},k}$ will be used to calculate the fractional contribution of the random parameters to this color channel $W^{\mathbf{r}}_{\mathbf{c},k}$ as shown in Eq. 6, which will then be used in the calculation of the importance of the $k^{\text{th}}$ color channel, $\alpha_k$, in Eq. 7. Fig. 6 shows a visualization of the components of $W^{\mathbf{r}}_{\mathbf{c},k}$.

Likewise, the dependencies of the $k^{\text{th}}$ scene feature $D^{\mathbf{p}}_{\mathbf{f},k}$ and $D^{\mathbf{r}}_{\mathbf{f},k}$ will be used to compute the $W^{\mathbf{r}}_{\mathbf{f},k}$ term with Eq. 3 that is then used to calculate $\beta_k$. The $W^{\mathbf{f},k}_{\mathbf{c}}$ term in Eq. 5 is computed as:

$$W^{\mathbf{f},k}_{\mathbf{c}} = \frac{D^{\mathbf{f},k}_{\mathbf{c}}}{D^{\mathbf{r}}_{\mathbf{c}} + D^{\mathbf{p}}_{\mathbf{c}} + D^{\mathbf{f}}_{\mathbf{c}}}, \tag{12}$$

where $D^{\mathbf{f}}_{\mathbf{c}}$ is the dependency of all colors on all features and $D^{\mathbf{f},k}_{\mathbf{c}}$ the dependency only on the $k^{\text{th}}$ scene feature. The $D^{\mathbf{r}}_{\mathbf{c}}$, $D^{\mathbf{p}}_{\mathbf{c}}$, and $D^{\mathbf{f}}_{\mathbf{c}}$ terms are calculated by summing over the color channels:

$$D^{\mathbf{r}}_{\mathbf{c}} = \sum_{1 \le k \le 3} D^{\mathbf{r}}_{\mathbf{c},k}, \quad D^{\mathbf{p}}_{\mathbf{c}} = \sum_{1 \le k \le 3} D^{\mathbf{p}}_{\mathbf{c},k}, \quad D^{\mathbf{f}}_{\mathbf{c}} = \sum_{1 \le k \le 3} D^{\mathbf{f}}_{\mathbf{c},k}. \tag{13}$$

Figs. 7 and 8 give the reader some intuition about the $\alpha_k$ and $\beta_k$ weighting terms by changing their values. Finally, we note that in our final algorithm we scale the size of $\alpha_k$ and $\beta_k$ slightly in each iteration of the multi-pass algorithm for improved results. Details are in Sec. 5.5 of the technical report.

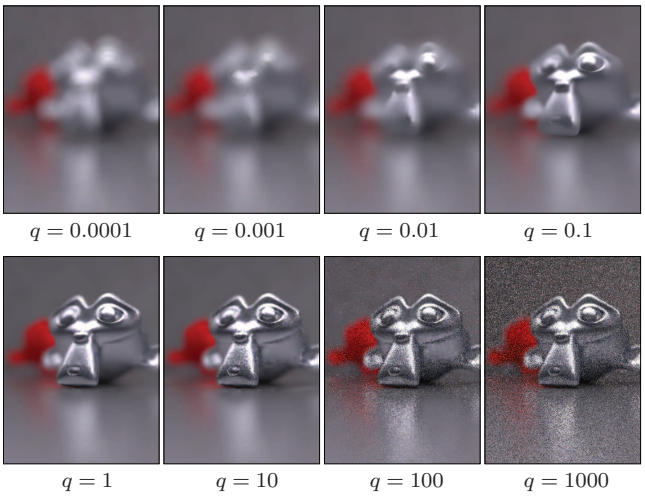



Fig. 7. To judge the impact of our color/feature weights $\alpha_k$ and $\beta_k$ in our bilateral filter, we pre-multiply them here by a variable $q$ that we vary from 0.0001 to 1000. When $q = 0.0001$, the color and feature differences are essentially ignored in the bilateral filter and it becomes a simple Gaussian blur. When $q = 1000$, differences between colors and features are weighted heavily (regardless of whether they depend on the random parameters), so the samples are left unfiltered, resulting in an image with MC noise.

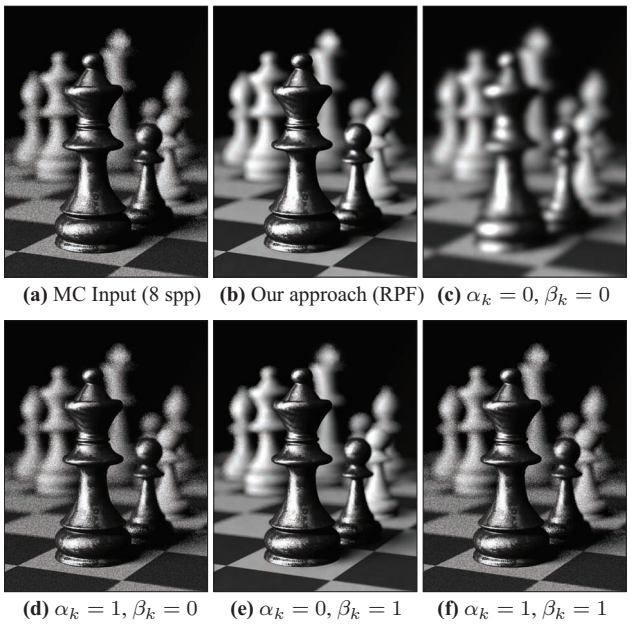



Fig. 8. We study the effect of $\alpha_k$ and $\beta_k$ by setting them to fixed values in this scene with DoF and area lighting. **(a)** MC input, and **(b)** the result of our approach ($\alpha_k \leftarrow$ Eq. 7, $\beta_k \leftarrow$ Eq. 5). **(c)** When we set $\alpha_k = \beta_k = 0$, the filter in Eq. 2 acts as a standard Gaussian filter and blurs the image (the clustering algorithm of Sec. 4.3.1 has been disabled here and in Fig. 7). **(d)** In this case, the filter preserves color differences so MC noise is preserved. **(e)** This performs cross-bilateral filtering using scene features, similar to previous work. The noise in the focused regions is removed because the scene features here are similar enough to blend the colors together (we correctly preserve the checkerboard pattern, though). The out-of-focus regions remain noisy because their features vary significantly from sample to sample. The hard shadow from the black queen is also overblurred because the filter ignores color differences, even though color here is not a function of the random point on the area light source. **(f)** When $\alpha_k = \beta_k = 1$, the filter preserves color/feature differences so all MC noise remains.

4.4.3 *Calculating mutual information.* To calculate the mutual information between two vectors $\mathbf{x}$ and $\mathbf{y}$ (or in our case, e.g., $\bar{\mathbf{f}}_{\mathcal{N},k}$ and $\bar{\mathbf{r}}_{\mathcal{N},l}$), we first calculate the histogram of each of them (for computing $p(x)$ and $p(y)$) as well as their joint histogram (for $p(x, y)$) and plug their probabilities into Eq. 8 to get $\mu(\mathbf{x}; \mathbf{y})$. To compute the histograms, we first make all the values positive by subtracting the minimum element in the vector and quantize the elements into integer bins by rounding their values. We count how many times the values of $\mathbf{x}$ fall inside each bin and find the probabilities by dividing by the length of $\mathbf{x}$. The joint histogram is calculated in a similar way, except with pairs of values $(\mathbf{x}, \mathbf{y})$. To implement this, we examined the mutual information code from the Matlab Central website [Peng 2007] and rewrote our own version in C. This sufficed for our algorithm, although it would be interesting to explore other ways of calculating mutual information in the future.

## 4.5 Filtering the samples

After calculating the color/feature weights $\alpha_k$ and $\beta_k$, we are ready to filter the samples' color. For every sample $i$ in the pixel $\mathcal{P}$, we loop over all the samples $j$ in neighborhood $\mathcal{N}$ and compute filter weights $w_{ij}$ using the weighted bilateral filter of Eq. 2 and use these weights to blend in the color contributions from these samples:

$$\mathbf{c}'_{i,k} = \frac{\sum_{j \in \mathcal{N}} w_{ij} \mathbf{c}_{j,k}}{\sum_{j \in \mathcal{N}} w_{ij}}, \tag{14}$$

where the denominator is never zero because at least $w_{ii} = 1$ (a sample fully contributes to itself). Note that this process filters the colors of individual samples (not pixels), and we perform this separately for every pixel in the image, since statistics change from pixel to pixel. After all samples in the image have been filtered, we repeat the process with a new iteration as shown in Alg. 1.

4.5.1 *Setting the variance of the filter.* To compute Eq. 2, we need variances $\sigma^2_{\mathbf{p}}, \sigma^2_{\mathbf{c}}, \sigma^2_{\mathbf{f}}$. The screen position variance $\sigma^2_{\mathbf{p}}$ is set by the filter box size, such that the standard deviation $\sigma_{\mathbf{p}}$ is one quarter the width of the box. Note that for speedup we compute this by randomly selecting samples with a Gaussian distribution with $\sigma^2_{\mathbf{p}}$ variance around the pixel of interest (see Sec. 4.1 of the technical report). The variances of the Gaussians for both the color and the feature are set to the same value $\sigma^2_{\mathbf{c}} = \sigma^2_{\mathbf{f}} = \frac{\sigma^2}{(1-W^{\mathbf{r}}_{\mathbf{c}})^2}$.

We divide these variances by $(1 - W^{\mathbf{r}}_{\mathbf{c}})^2$ because, in the end, we only care about the sample color and want a large filter wherever the color depends a lot on the random parameters (i.e., is very noisy). This term adjusts the size of the Gaussian based on the overall noise level, making it large when needed. We could have rolled the $\sigma^2_{\mathbf{c}}$ and $\sigma^2_{\mathbf{f}}$ terms into the $\alpha_k$ and $\beta_k$ coefficients in Eq. 2, but because the $\sigma^2_{\mathbf{c}}$ and $\sigma^2_{\mathbf{f}}$ terms depend on all three color channels (because of the $W^{\mathbf{r}}_{\mathbf{c}}$ term) as opposed to $\alpha_k$ (whose $W^{\mathbf{r}}_{\mathbf{c},k}$ term varies per color channel), it was easier to separate them. This way, the $\sigma^2_{\mathbf{c}}$ and $\sigma^2_{\mathbf{f}}$ terms modulate the overall size of the Gaussian while $\alpha_k$ and $\beta_k$ adjust it further based on dependencies with the random parameters. The $\sigma^2$ parameter was selected by experimenting with scenes at 8 samples/pixel, but is scaled inversely by the number of samples per pixel $s$ (so as $s$ grows, $\sigma^2$ gets smaller): $\sigma^2 = 8\sigma^2_8/s$. For noisy scenes (e.g., indoor path-tracing such as Figs. 11 and 13) we used $\sigma^2_8 = 0.02$, while for all others we set $\sigma^2_8 = 0.002$.

In the end, our approach is a biased but consistent estimator because it converges to the value of the integral as the number of samples per pixel $s$ goes to infinity. As $s \to \infty$, $\sigma^2_{\mathbf{c}} = \sigma^2_{\mathbf{f}} \to 0$, which produces a weight $w_{ij} = 1$ only when $i = j$ and zero elsewhere. Therefore, the colors of the samples are not filtered at all, so our approach converges to standard Monte Carlo, which is a consistent

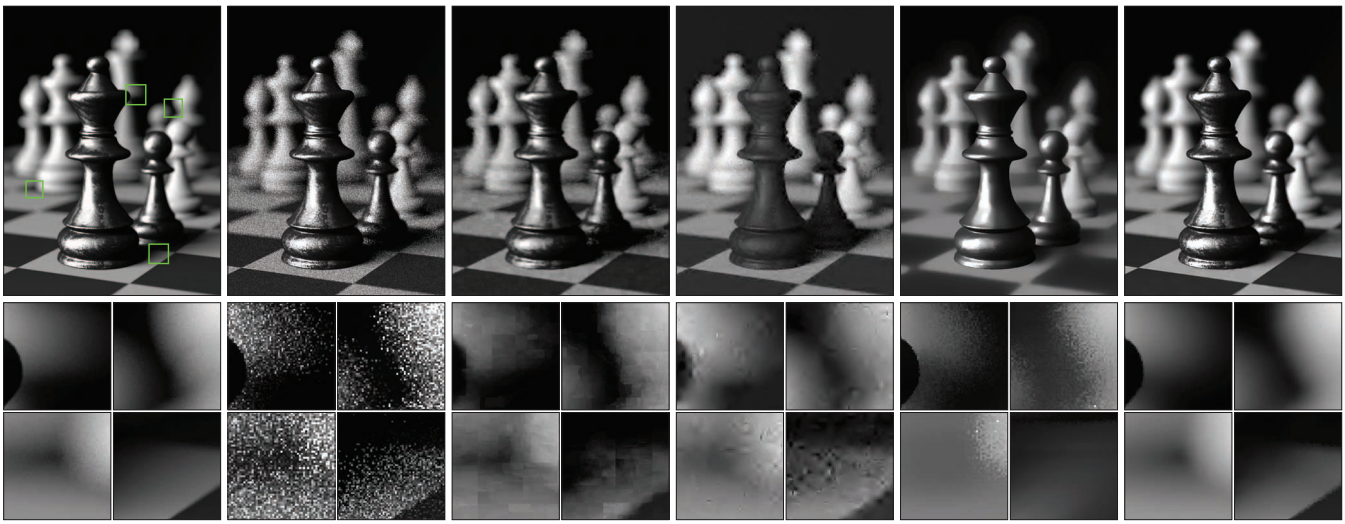



Fig. 9. Here we compare various approaches on the CHESS scene (a 6D integration problem with depth of field and an area light source) rendered with 8 samples/pixel. The MDAS algorithm exhibits blockiness at this low sampling rate because of its nearest neighbor reconstruction, while AWR has wavelet artifacts. The À-Trous method is unable to handle the depth-of-field effect because the scene features (normals, world position) change significantly in the noisy out-of-focus region, so it does not filter across these pixels. Our algorithm, on the other hand, determines the statistical dependency between these scene features and the random parameters so it can produce noise-free images that are comparable to the reference image. Although the reference and our result are different, the reference image still has visible noise even at 8,192 samples/pixel while our result is completely smooth.

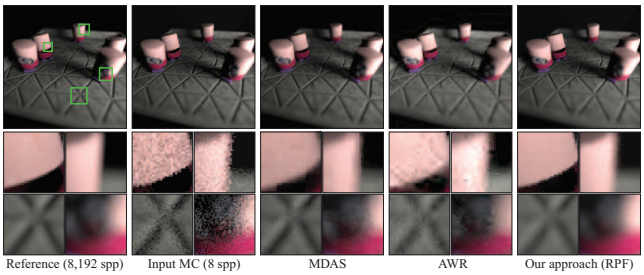



Fig. 10. This figure compares various approaches on the TOASTERS scene (a 6D integration problem with DoF and area light source) at 8 spp.

estimator. For a small number of samples, however, our filter introduces bias because the expected value of the estimator is no longer equal to the integral. We discuss this in more detail in Sec. 6.1.

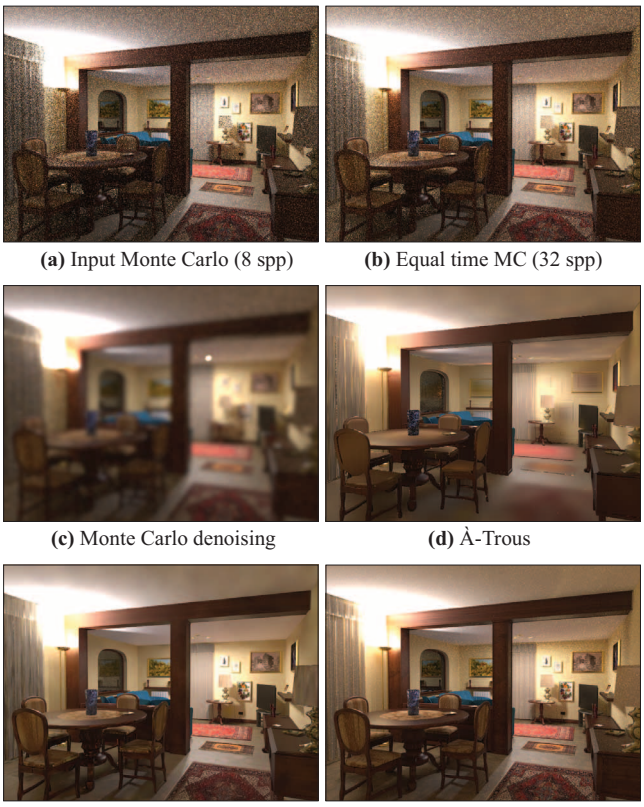
4.5.2 *Handling sample spikes in high dynamic range.* The samples computed by MC rendering systems are usually in a high-dynamic range representation in order to properly calculate illumination effects. Sometimes a sample can be several orders of magnitude larger than its neighbors, e.g., in a path-traced indoor scene where a sample might hit a light source directly in an early bounce. Even if their color depends on the random parameters, these sample spikes do not get blended properly with their neighbors because their color is so different that the bilateral filter does not mix them, causing speckling artifacts in the final image. An example of this is shown in Fig. 5 of the technical report [Sen and Darabi 2011b].

To address this, we simply examine the color of the samples *after* our filtering step to look for samples that are still significant outliers, i.e., greater than a standard deviation from the average pixel color. These samples have not been filtered by our process, so we simply set their color to be equal to the pixel average. In this way, the contribution of the spike is taken into account when we take the pixel average, but it does not overwhelm the other samples when integrating them into a final pixel color. We admit this is a simplistic way to address this issue. Others have tried more principled approaches (e.g., [DeCoro et al. 2010]) and it would be interesting to combine such an approach with RPF in the future.

## 5. RESULTS

We implemented the random parameter filtering (RPF) algorithm in C++ and integrated it into the PBRT2 [Pharr and Humphreys 2010] and LuxRender [LuxRender 2011] rendering systems for our experiments. All results shown here were computed on an Intel dual quad-core Xeon X5570 3.06GHz machine with 16GB of memory. To test our algorithm against state-of-the-art sampling/reconstruction techniques, we compare with the multi-dimensional adaptive sampling (MDAS) algorithm of Hachisuka et al. [2008], the adaptive wavelet rendering (AWR) work of Overbeck et al. [2009], the sheared-filter motion blur approach of Egan et al. [2009] (we call it SFMB, for short), and Sen and Darabi's compressive integration (CI) framework [2010]. For all of these approaches, we use the implementations provided by the respective authors. We also compare our approach to two MC noise-filtering approaches: Xu and Pattanaik's denoising algorithm [2005] and Dammertz et al.'s À-Trous edge-avoiding filter [2010].

To test our algorithm, we ran it on a variety of complex scenes with various Monte Carlo effects such as depth of field, area light sources, motion blur, path-tracing, and Russian roulette. When describing the dimensionality of a scene, we include the pixel integration for antialiasing to match the nomenclature used in the literature (e.g., [Hachisuka et al. 2008; Overbeck et al. 2009]). We only count path-tracing as a 2D integration because we only use the first reflection direction in our list of random parameters. We also experimented on a large set of simple test cases to ensure correctness of our algorithm, a subset of which appears in the associated technical report [Sen and Darabi 2011b]. The scenes were also rendered at a variety of sizes, from $512 \times 512$ for the TOASTERS (Fig. 10) and CAR (Fig. 12) scene to full HD resolution ($1920 \times 1080$) for the TOY GYRO (Fig. 13) and SAN MIGUEL scenes (Figs. 1 and 19).



**(a)** Input Monte Carlo (8 spp) **(b)** Equal time MC (32 spp)

**(c)** Monte Carlo denoising **(d)** À-Trous

**(e)** Our approach (RPF) **(f)** Reference Monte Carlo (8,192 spp)

Fig. 11. We compare RPF with previous methods for filtering MC noise using the path-traced PERSIAN ROOM scene. **(a)** Input at 8 samples/pixel. **(b)** With equal time as RPF, standard MC gets only 32 samples/pixel and is still noisy. **(c)** The MC denoising algorithm of Xu and Pattanaik [2005] and **(d)** the À-Trous filter of Dammertz et al. [2010] cannot overcome the magnitude of the noise without significant overblurring. **(e)** Our algorithm properly eliminates the noise in this scene but preserves detail. **(f)** Reference at 8,192 samples/pixel, for comparison. An equal-quality comparison was difficult for this scene, because even the reference image still has noise.

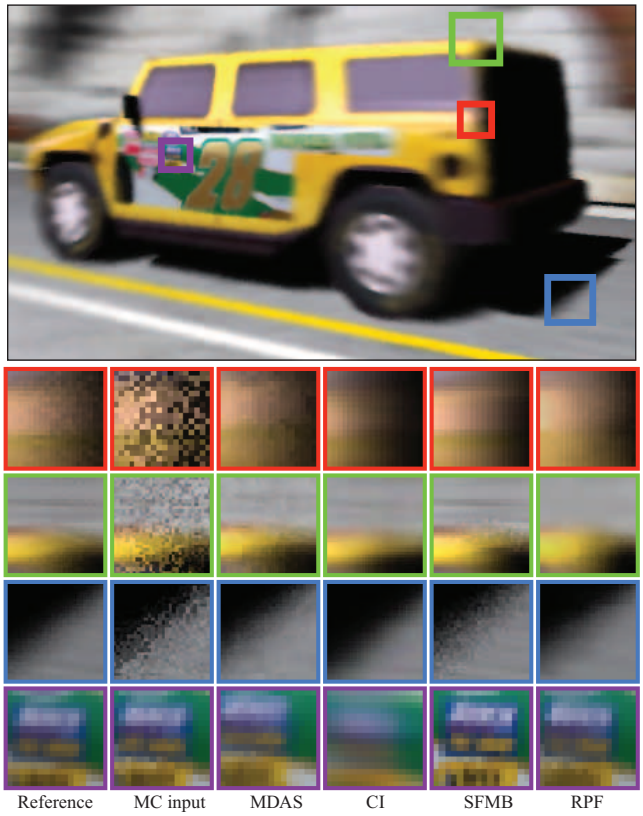



Fig. 12. This motion-blurred CAR scene was rendered at 4 spp with various algorithms (top image is from RPF). The first row of insets shows a region where compressive integration (CI), sheared-filter motion blur (SFMB), and RPF all work well. In the second row, SFMB reconstructs detail in the back wall that our algorithm overblurs, but it also leaves noise on the roof of the car that RPF removes. The third inset is left noisy by SFMB but is denoised by ours. Finally, the fourth row shows a static region of the scene that should be sharp but is overblurred by the CI approach. The full images are available in the technical report [Sen and Darabi 2011b].

**Quality comparisons –** We first examine the quality of images produced with our approach. Fig. 1 is a frame from the SAN MIGUEL video sequence which requires 4D integration with path-tracing and was reconstructed from the 8 sample/pixel noisy input data shown in the left inset. Our algorithm can handle refraction, reflection and other effects while preserving shadows and other details in the scene. We also include a seamless inset with an 8,192-sample/pixel reference image to show that the differences between the two are subtle, even though the reference took more than 24 hours to compute and our result just a few minutes.

Fig. 9 compares MDAS, AWR, and À-Trous with our approach for the CHESS scene, which requires 6D integration (DoF, area light source). The reference image still has some noise even at 8,192 samples/pixel, while ours produces a smooth result with only 8. Fig. 10 shows another comparison of MDAS, AWR, and our approach for the TOASTERS scene, a 6D integration problem (DoF, area light source). Fig. 11 shows the PERSIAN ROOM, a complex, path-traced scene that produces noisy results, even with 8,192 samples/pixel. Here, we compare our method against the two filtering methods (À-Trous and Monte Carlo denoising), both of which eliminate important scene features in the process of reducing the MC noise. RPF gives better results since our cross-bilateral filter is able to preserve scene detail. In addition, we size our filter kernel based on the dependence of the color on the random parameters (Sec. 4.5.1), thereby reducing the amount of blur in parts of the image where the random parameters do not affect the final color.

Fig. 12 shows the CAR scene, a 3D integration problem with motion blur to compare against the sheared filter motion blur (SFMB) and the compressive integration (CI) algorithms. The comparison with SFMB is interesting because this algorithm is not general but is specifically designed to handle motion blur. In some parts of the image we are able to match the quality of SFMB (first row of insets), while in others SFMB is better at blurring the noise in the direction of motion (second row of insets) since it has specific motion blur information that tracks points on the scene and establishes a well-defined motion field. However, SFMB also has artifacts and it does not denoise the shadows below the car (third row of insets), or at the boundary between the car and the back wall, which our method handles correctly. The CI approach works reasonably for the motion-blurred regions, but its Fourier basis tends to overblur regions of the image that should be sharp (last row of insets).

Fig. 13 shows our result for the TOY GYRO scene with 8D integration (DoF, motion blur, path-tracing, and Russian roulette), rendered with 8 spp. We also show how we handle the Russian roulette

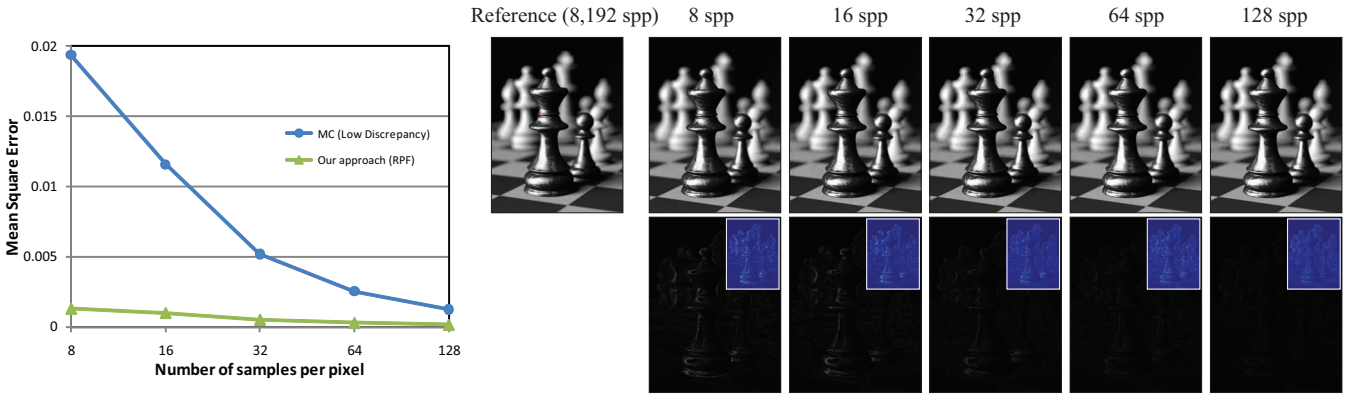



Fig. 14. Convergence of RPF to the ground truth as we increase the number of samples. On the left we show a plot of MSE versus the number of samples for standard MC and for our approach. Unlike Fig. 15, MDAS was excluded because it took too long for this scene at higher sampling rates. On the right we show the output of our algorithm in the top row, and difference images (with "heat" image insets) that compare our results to the reference in the bottom.


Fig. 13. Result of RPF on the TOY GYRO scene with 8D integration: DoF, motion blur, path-tracing, and Russian roulette for the cockpit window. The top image shows the MC input at 8 spp, the middle is produced by our algorithm. The bottom insets show that there is still noise after filtering if we do not include the Russian roulette term in the list of random parameters.

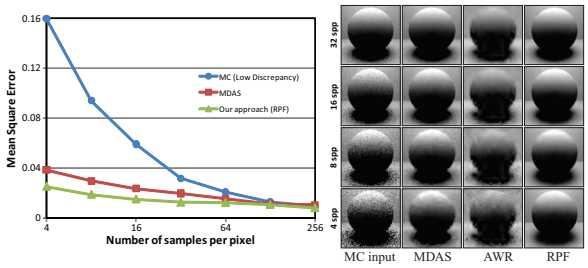



Fig. 15. We compare the quality of RPF, MDAS, and AWR as a function of the sampling rate. The MSE curve only compares against MDAS because the AWR implementation from Overbeck et al. uses a different rendering system and has shading differences that make fair MSE comparisons impossible. Therefore, we only show quality comparisons in the form of images to the right.

parameter used to trace rays through the window of the cockpit in the insets at the bottom. To measure the quality of our output in a quantifiable way, we perform a mean-squared-error (MSE) comparison between our result and the ground truth as a function of the number of samples in Fig. 14 and compare against MDAS in Fig. 15. Both plots show that our algorithm converges as expected to the standard Monte Carlo with a large number of samples.

In terms of memory consumption, our algorithm has to store additional information at each sample to maintain the sample vector and associated features. The scenes that required the largest sample vectors turned out to be the path-tracing scenes where we store the world position and normal information for the second intersection point, in addition to the other standard information. In these cases, we store up to 27 floats (108 bytes) per sample, which is reasonable especially considering that we are able to get a satisfactory image with a low number of samples. We note that deferred shading rendering systems [Deering et al. 1988] have similar memory requirements. For comparison, the MDAS algorithm uses 400 bytes/sample [Hachisuka et al. 2008] and requires more samples (often 32 or more) to get reasonable results.

**Timing results** – Our algorithm runs in a few minutes for all scenes tested. For example, for the CHESS scene in Fig. 9 our algorithm runs in 4.9 minutes (including the rendering time for the input samples). For comparison, the MDAS algorithm using the authors' available implementation takes 20.5 minutes to render the same scene and produce the results shown. This algorithm is slowed down because it suffers from the curse of dimensionality and in this case we are dealing with a 6D integration problem.

For equal-timing comparisons, we compare against standard MC in Figs. 11, 16, and 17. Equal-time comparisons with MDAS do not make sense, since that algorithm is both slower and produces results of lower quality. Although our algorithm is comparable in speed to AWR and SFMB, both of those algorithms use different rendering systems which makes fair timing comparisons impossible. To get a breakdown of computation time for the different stages of our algorithm, we instrumented our code running on the PERSIAN ROOM scene. Our RPF implementation took approximately

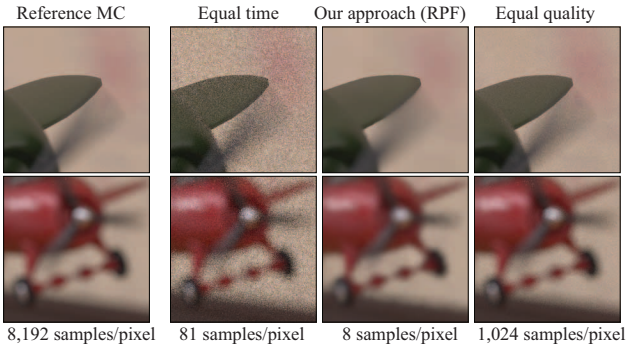



Fig. 16. Here are equal time/quality comparisons for two insets of the TOY GYRO scene of Fig. 13. Our algorithm at 8 samples/pixel took about 12 minutes total, and in that amount of time standard MC could render 81 samples/pixel. For equal quality, we increased the sampling rate of standard MC to match the mean-squared error (MSE) of our approach, which happened at 1,024 samples/pixel and took 2.5 hours to render.

Table II. Timing breakdown for our algorithm for Fig. 11.

| Task | Time (secs.) | Overall Percentage |
|---|---|---|
| Initial rendering (8 spp) | 76.0 | 33.7% |
| Preprocess samples | 48.3 | 21.4% |
| Compute feature weights | 77.1 | 34.2% |
| Filtering samples | 22.4 | 9.9% |
| Additional Overhead | 1.8 | 0.8% |
| Total time for RPF algorithm | 225.5 | 100% |

225 seconds to produce the image in Fig. 11e at $800 \times 600$ resolution, while the reference rendering, for comparison, took more than 17 hours and is still noisy. The complete timing breakdown is shown in Table II. The "Preprocess" stage clusters the samples and removes their mean and standard deviation (Sec. 4.3), "Compute feature weights" calculates the statistical dependencies and the color and feature weights $\alpha_k$ and $\beta_k$ (Sec. 4.4), and "Filtering samples" calculates the weights and filters the samples (Sec. 4.5).

Our algorithm took the longest on the SAN MIGUEL video frames, which were rendered at full, $1920 \times 1080$ HD resolution and took about 14 minutes to render and filter each frame. For comparison, each reference frame took more than 24 hours to render and still had visible noise. This means that rendering the full, 300-frame video sequence at reference quality would have taken most of a year using a fairly powerful machine. Our algorithm produces reasonable results $100\times$ faster and is therefore a viable alternative for pre-visualization in production environments.

## 6. DISCUSSION

### 6.1 Convergence properties

As discussed earlier, our algorithm is biased when the number of samples is small. The reason for this is that a few samples could miss scene information that our weighted bilateral filter cannot reconstruct. Therefore, the expected value of our pixel value estimates will differ from the correct "ground truth" value at low sampling rates, the definition of estimator bias. However, our algorithm is consistent because as the number of samples is increased the algorithm converges to the ground truth, as shown in Figs. 14, 15, and 17 where RPF produces results with lower MSE than standard MC at all sampling rates tested. Fig. 4 in the technical report shows this trend continues for the ROBOTS scene until 2,048 spp.

The reason for this consistency is that we calculate the RPF filter's variance $\sigma^2$ by dividing a fixed variance parameter by the number of samples ($\sigma^2 = 8\sigma_g^2/s$), as described in Sec. 4.5.1.

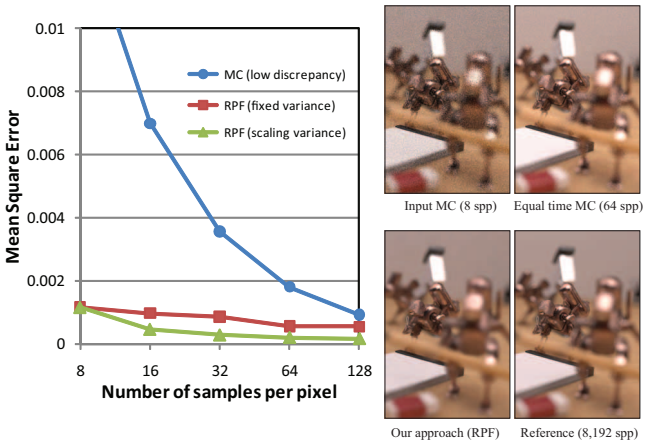


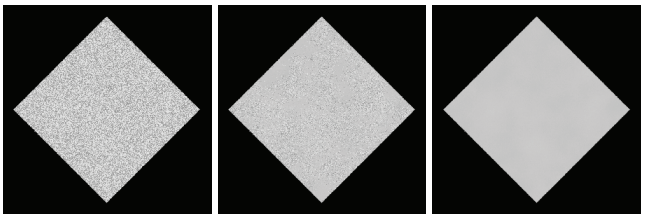

Fig. 17. The proposed RPF algorithm (green curve) is consistent because it reduces to standard Monte Carlo as the number of samples is increased. If we fix the variance of the filter, however, we do not get a consistent result. In terms of timing, it takes 15 secs. for the input samples to render, approximately 106 secs. for our algorithm and for the equal time MC (64 samples/pixel), and almost 4 hours for the reference image.

Therefore, the filter variance goes to zero as the number of samples $s \to \infty$, which means that at high sampling rates our algorithm reduces to brute force Monte Carlo, a consistent technique. To test this, we ran the RPF algorithm but fixed the variance $\sigma^2 = \sigma_g^2$ to be independent of the sampling rate, shown in the red curve in Fig. 17. As expected, fixing the variance results in an algorithm that is not consistent, and standard MC produces results with lower MSE at sampling rates higher than 128 samples/pixel.

### 6.2 Generality of our approach

Previous approaches, such as Á-Trous [Dammertz et al. 2010] and Blender's bilateral blur filter [Blender 2011], use a cross-bilateral filter on a set of scene features to try to remove the MC noise while preserving scene detail. Our framework explains why these approaches work when the scene features are not themselves functions of the random parameters (effectively, these approaches apply the bilateral filter of Eq. 2 with $\alpha_k = 0, \beta_k = 1$). This is why these methods have been predominantly used to denoise effects such as area light sources and global illumination, where the scene features (surface normal, world position, etc.) are not functions of the random parameter (i.e., the position on the light source). However, these algorithms cannot handle effects such as depth of field or motion blur, because the scene features in these cases are also corrupted by the MC noise and their incorporation into the cross-bilateral filter prevents the MC noise from being removed.

Our algorithm, on the other hand, can handle a wide range of Monte Carlo effects easily because we estimate the dependencies of sample values on the random parameters regardless of what the random parameters mean physically. To handle a particular effect, we only need to add the appropriate random parameter(s) to the sample vector. In this way, we are able to denoise effects like depth of field (Figs. 2, 3, 5, 7, 9, 10, 13, 14, 16, and 17), motion blur (Figs. 12, 13, and 16), glossy reflection (Figs. 5, 7, 13, and 17), area light sources (Figs. 9, 10, 14, 15, and 21), and path-tracing (Figs. 1, 11, 13, 17, and 19). More interestingly, we can also correctly handle subtle MC effects such as integrating over multiple light sources using a discrete random number to select between them (used in the PERSIAN ROOM scene in Fig. 11), or using Russian roulette [Arvo and Kirk 1990] to randomly either transmit or reflect a ray. For example, the



Input Monte Carlo (4 spp) RPF (correlation) RPF (mutual information)

Fig. 18. This figure compares different metrics of statistical dependence. Here, a diffuse square is illuminated by a uniform hemisphere and shaded by taking four samples at random angles and cosine weighting them, so the color for each sample is simply $c_i = \cos(\theta_i)$, where $\theta_i$ is the random parameter. We would like our approach to be able to detect a relationship between the color and the random parameter so that it can remove the MC noise. Correlation, unfortunately, cannot recover this relationship correctly as explained in the text, while mutual information works quite well.

TOY GYRO scene of Fig. 13 uses Russian roulette for the glass in the cockpit, resulting in noisy speckles when rendered with only 8 samples/pixel. With our algorithm, however, we are able to determine the relationship between the color and this random parameter and appropriately filter this effect without significantly overblurring the objects behind the glass.

On a related note, because our algorithm determines what is noise by calculating the statistical dependencies of sample values on the random parameters, the removal of a known random parameter from the sample vector will lead the algorithm to the incorrect conclusion that the noise from this parameter is actually part of the scene content and it will not be filtered. Examples of this are shown in Figs. 5 and 13. This is not an artifact of our algorithm, but rather it highlights how it effectively removes noise from a scene.

### 6.3 Other ways to estimate functional relationships

We also experimented with other metrics for estimating functional dependence (e.g., correlation and PCA) to see if they offered improvement. We found these measures to be significantly inferior to mutual information, even for simple cases. For example, Fig. 18 shows a simple, diffuse square lit by a uniform hemispherical light source where we shade our samples by $c_i = \cos(\theta_i)$, with random parameter $\theta_i$ uniformly distributed from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. The correlation between the two is given by $\text{corr}(\theta, c) = E[(\theta - \mu_\theta)(c - \mu_c)]/\sigma_\theta\sigma_c$, which analytically gives us a value of 0 for this example. Since there is obviously a relationship between $c$ and $\theta$, correlation is not a good metric for our work.

Mutual information, on the other hand, detects a relationship and produces the correct result in this case. Of course, it can sometimes also fail to detect a functional relationship between the random parameters and sample values. For example, if the scene function varies with respect to the random parameters in an apparently random fashion (e.g., from a very noisy, random texture), mutual information would determine that the random variables are independent. In practice, however, we found that mutual information worked well for the scenes shown, where the relationship between the samples' values and the random parameters is complex.

### 6.4 Extension to animated scenes

The rendering of animated scenes is important in high-end rendering. Fortunately, it is easy to extend our algorithm to handle a 3D spatio-temporal volume. Here, each frame of the sequence is rendered independently (as is typically done in industry), and the sample vectors for each frame are passed to our post-process filter. To create a 3D block around a pixel to compute statistics and perform filtering, we do a quick search of the nearby regions in the adjacent images to find the best match for the given pixel using the supplementary scene information. This increases the temporal correlation across frames and improves the filtering process. Since we are now operating on a 3D block of pixels, our block size can be smaller and still have enough samples for computing statistics, thereby improving locality. This extension is also an advantage of our algorithm over general adaptive-sampling techniques (e.g., MDAS and AWR), which are difficult to extend to animated sequences.

Fig. 19. This is a 1920 × 1080 frame from the SAN MIGUEL video sequence. The top image is rendered at 8,192 spp, which took more than 24 hours to compute. The middle image is the input to our algorithm at 8 spp. The bottom image is our result, produced in about 14 minutes. We can recover fine features even though the detail is not visible in the noisy input.

To test this extension, we generated three animated sequences for the CHESS, PERSIAN ROOM and SAN MIGUEL scenes in the supplementary video. Fig. 19 shows a frame from the SAN MIGUEL video sequence, which was rendered at full, 1920 × 1080 HD resolution. Although the path-tracing noise requires us to blur over large regions of the image, our bilateral filter finds and preserves much of the important scene content. The insets show that although much of the detail is not visible in the noisy input (many pixels are completely black), our algorithm is able to recover it reasonably. It can do this because it determines that the colors of the samples are highly dependent on the random parameters of the path tracer, so the bilateral filter ignores them completely. On the other hand, the texture value (which contains the detail in this scene) is not dependent on the random parameters, so the filter blends in the color from samples that have similar texture values to remove the noise.


Fig. 20. Our result differs from the reference MC, as shown by these difference and intensity images (showing the norm of the error) for the scene in Fig. 19. The differences occur in regions of high-frequency detail that our bilateral filter tries to preserve but ends up slightly overblurring due to the significant amount of noise in the input image. The transparent vase on the table also exhibits differences because the path tracer does not have enough rays at 8 samples/pixel to compute the inter-reflections inside the glass.

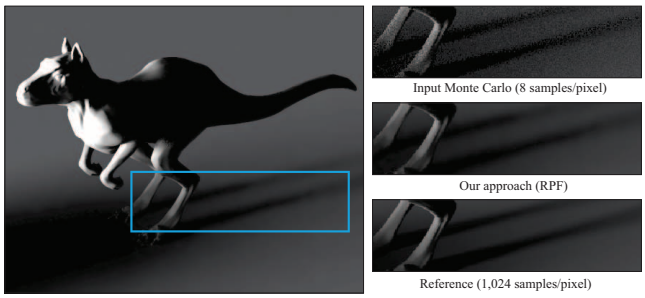



Fig. 21. In this scene, the KILLEROO casts overlapping hard and soft shadows from the same light source onto the floor. Although our algorithm detects the transitions from hard to soft and tries to preserve the edge between them, it overblurs the hard shadows in the final result. The image on the left is produced by our algorithm.

## 6.5 Limitations and future work

Although our approach is reasonably fast, its bias at low sampling rates produces results that are visibly different from the ground truth. Figs. 14 and 20 help to visualize this by showing the difference in images produced by our algorithm and a reference rendering. However, the images produced are of sufficient quality to be used for pre-visualization in film production, especially given that the result is achieved in only a few minutes.

Some of these differences are the result of limitations of our approach. First of all, the cross-bilateral filter we are proposing is isotropic in terms of the dimensions of the integral, since we simply compute the $\alpha_k$ and $\beta_k$ weights by measuring the dependency on all random parameters without distinguishing between them. This could lead to artifacts such as the overblurring of the king's crown in the CHESS scene in Fig. 9 or the back wall of the CAR scene in the second row of insets in Fig. 12. Extending the algorithm to handle anisotropic filtering effects could lead to improved results.

Furthermore, our filter can have difficulty with regions that have a lot of high-frequency scene detail but also a significant amount of Monte Carlo noise. We call this the "dueling filter" problem: on the one hand the filter kernel should be as large as possible to filter out the noise, but on the other it should be as small as possible so as to not overblur the important detail. Our bilateral filter tries to preserve the scene detail that is not a function of the random parameters, but it is not perfect and some of it is blurred. This can be seen in the TOY GYRO scene of Fig. 13, where the pilot's face in the inset is slightly blurred because our filter is trying to denoise the glass in front of it. This also happens in the SAN MIGUEL scene of Figs. 1 and 19, where the floor has a lot of textured detail but is also very noisy because of the global illumination. The bilateral filter is able to recover a remarkable amount of detail from the noisy information, but some of the texture is blurred.

A related failure case is shown in Fig. 21, where an area light source is casting both hard and soft shadows from the same object onto the ground. Our formulation is able to detect the appropriate dependencies (hard shadows do not depend on the random position of the sample on the light source, while soft shadows do) and attempts to preserve the edges between them, but because of the large amount of noise in the soft shadow the edge gets slightly blurred out. It would be interesting to find ways to extend our formulation to address this dueling filter problem.

Another limitation is the way the screen coordinates are handled differently from other random parameters, which means that our RPF algorithm does not filter across these dimensions to perform pixel antialiasing. Therefore, we see no benefit in regions where there is little dependency on random parameters (e.g., pixels in focus) because we do not filter there at all. These pixels might remain noisy at low sampling rates, especially if there is a large amount of detail in the pixel. This is really a sampling problem, since we do not have enough samples to antialias the pixel correctly. It might be possible in the future to reformulate our algorithm and incorporate the screen coordinates into the filtering process.

In terms of directions of future work, there might be better ways to address the mixing-of-statistics problem than the clustering algorithm presented here. Currently, our clustering can produce artifacts in certain scenes, for example in the blurred region in a DoF scene where a single pixel might get only foreground samples (instead of a mix of foreground and background) which would cause this pixel to not be blended properly. In addition, one might explore new metrics for estimating functional dependency instead of mutual information. Since our algorithm focuses exclusively on the reconstruction filter, it might be possible to combine it with one of the advanced adaptive sampling algorithms such as MDAS or AWR for improved performance. In terms of speed, our current algorithm was sufficiently fast for the purposes of this paper, so we did not worry about its optimization. However, it has the potential of being significantly accelerated because it is inherently parallel and a possible candidate for implementation on the GPU. This could reduce the filtering time considerably and make the algorithm suitable even for accelerating fast, GPU-based Monte Carlo rendering systems.

We note that this work represents only an initial effort in exploring a new approach to denoising Monte Carlo rendering, and we hope that other researchers will build upon the proposed approach and explore its applications. For example, the problem of trying to determine whether a particular feature is part of the scene or is an artifact of the stochastic process is an issue in many MC rendering algorithms. Our insight that we can answer this question using the functional dependencies between the feature and the random parameters could also be used to improve these other algorithms.

## 7. CONCLUSION

We have presented a novel algorithm called random parameter filtering that removes noise in Monte Carlo rendering by estimating the functional dependency between sample features and the random inputs to the system. In order to approximate these functional relationships, we use a measure of statistical dependency called mutual information, which is applied to a local neighborhood of samples in each part of the image. This dependency is then used to reduce the importance of certain scene features in a cross-bilateral filter, which preserves important scene detail even in parts where we must blur heavily to remove MC noise. The results produced by our technique are computed in just a few minutes but are comparable to reference renderings with a thousand times more samples. This paper

is only the first effort to leverage the functional dependencies on random parameters to denoise Monte Carlo renderings in this manner. We expect that future improvements on the algorithm by other researchers will continue to improve the quality of the results.

## ACKNOWLEDGMENTS

Lei Xiao conducted the experiments to produce several images in this paper. Mauricio Gómez produced the supplementary video. Frances Strong helped proofread drafts of the final manuscript. We also thank Toshiya Hachisuka, Ryan Overbeck, and Kevin Egan for help with the implementations of their respective algorithms. Finally, we gratefully acknowledge the sources of the scenes in the paper (in order of appearance): SAN MIGUEL – Guillermo M. Leal Llaguno (PBRT2 book), CHESS – Wojciech Jarosz, MONKEY HEADS – Blender, TOASTERS – Andrew Kensler, PERSIAN ROOM – Luca Cugia, CAR – Turbosquid user graphicdoom (car model), Wikipedia user Jongleur100 (wall texture), Kevin Egan (scene), TOY GYRO and ROBOTS – Jesper Lloyd, KILLEROO – headus/Rezard (PBRT2 book). This work was funded by National Science Foundation CAREER award IIS-0845396.

## REFERENCES

ARVO, J. AND KIRK, D. 1990. Particle transport and image synthesis. In *ACM SIGGRAPH '90*. ACM, New York, NY, USA, 63–66.

BLENDER. 2011. Bilateral blur filter compositing node for denoising ray-traced ambient occlusion. http://www.blender.org/development/release-logs/blender-246/compositing-nodes/.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. In *ACM SIGGRAPH '84*. ACM, New York, NY, USA, 137–145.

COVER, T. AND THOMAS, J. 2006. *Elements of Information Theory*, Second ed. John Wiley & Sons, Hoboken, New Jersey.

DAMMERTZ, H., SEWTZ, D., HANIKA, J., AND LENSCH, H. P. 2010. Edge-avoiding À-trous wavelet transform for fast global illumination filtering. In *Proceedings of High Performance Graphics 2010*. 67–75.

DECORO, C., WEYRICH, T., AND RUSINKIEWICZ, S. 2010. Density-based outlier rejection in Monte Carlo rendering. In *Pacific Graphics*. Vol. 29.

DEERING, M., WINNER, S., SCHEDIWY, B., DUFFY, C., AND HUNT, N. 1988. The triangle processor and normal vector shader: a VLSI system for high performance graphics. In *ACM SIGGRAPH '88*. ACM, New York, NY, USA, 21–30.

DUTRÉ, P., BALA, K., AND BEKAERT, P. 2006. *Advanced Global Illumination*. A K Peters.

EGAN, K., TSENG, Y.-T., HOLZSCHUCH, N., DURAND, F., AND RAMAMOORTHI, R. 2009. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. 28,* 3, 1–13.

EISEMANN, E. AND DURAND, F. 2004. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph. 23*, 673–678.

HACHISUKA, T., JAROSZ, W., WEISTROFFER, R. P., DALE, K., HUMPHREYS, G., ZWICKER, M., AND JENSEN, H. W. 2008. Multi-dimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27,* 3, 1–10.

HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. H. 2001. *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer-Verlag.

JENSEN, H. W. 2001. *Realistic Image Synthesis Using Photon Mapping*. A K Peters.

JENSEN, H. W. AND CHRISTENSEN, N. J. 1995. Optimizing path tracing using noise reduction filters. In *Winter School of Computer Graphics (WSCG) 1995*. 134–142.

KELLER, A. 1998. Quasi-Monte Carlo methods for photorealisitic image synthesis. Ph.D. thesis, Universität Kaiserslautern.

LAINE, S., SARANSAARI, H., KONTKANEN, J., LEHTINEN, J., AND AILA, T. 2007. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007*. Eurographics Association, 277–286.

LEE, M. AND REDNER, R. 1990. A note on the use of nonlinear filtering in computer graphics. *IEEE Computer Graphics and Applications 10,* 3 (May), 23–29.

LUXRENDER. 2011. http://www.luxrender.net/.

MAHALANOBIS, P. C. 1936. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences of India 2,* 1, 49–55.

MCCOOL, M. D. 1999. Anisotropic diffusion for Monte Carlo noise reduction. *ACM Trans. Graph. 18,* 2, 171–194.

MEYER, M. AND ANDERSON, J. 2006. Statistical acceleration for animated global illumination. *ACM Trans. Graph. 25,* 3, 1075–1080.

MITCHELL, D. P. 1991. Spectrally optimal sampling for distribution ray tracing. *SIGGRAPH Comput. Graph. 25,* 4, 157–164.

OVERBECK, R. S., DONNER, C., AND RAMAMOORTHI, R. 2009. Adaptive wavelet rendering. *ACM Trans. Graph. 28,* 5, 1–12.

PENG, H. 2007. Matlab package for mutual information computation. http://www.mathworks.com/matlabcentral/fileexchange/14888.

PERONA, P. AND MALIK, J. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12,* 7 (jul), 629 –639.

PETSCHNIGG, G., SZELISKI, R., AGRAWALA, M., COHEN, M., HOPPE, H., AND TOYAMA, K. 2004. Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. 23*, 664–672.

PHARR, M. AND HUMPHREYS, G. 2010. *Physically Based Rendering: From Theory to Implementation*, Second ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

RUSHMEIER, H. E. AND WARD, G. J. 1994. Energy preserving non-linear filters. In *ACM SIGGRAPH '94*. New York, NY, USA, 131–138.

SAITO, T. AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-D shapes. In *ACM SIGGRAPH '90*. ACM, New York, NY, USA, 197–206.

SBERT, M., FEIXAS, M., RIGAU, J., VIOLA, I., AND CHOVER, M. 2007. Applications of information theory to computer graphics. In *EUROGRAPHICS '07*. Prague, Czech Republic, 625–704.

SEGOVIA, B., IEHL, J. C., MITANCHEY, R., AND PÉROCHE, B. 2006. Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of the ACM Symposium on Graphics Hardware '06*. ACM, New York, NY, USA, 53–60.

SEN, P. AND DARABI, S. 2010. Compressive estimation for signal integration in rendering. *Computer Graphics Forum 29,* 4, 1355 1363.

SEN, P. AND DARABI, S. 2011a. Compressive rendering: A rendering application of compressed sensing. *IEEE Transactions on Visualization and Computer Graphics 17*, 487–499.

SEN, P. AND DARABI, S. 2011b. Implementation of random parameter filtering. Tech. Rep. EECE-TR-11-0004, University of New Mexico.

SOLER, C., SUBR, K., DURAND, F., HOLZSCHUCH, N., AND SILLION, F. 2009. Fourier depth of field. *ACM Trans. Graph. 28,* 2, 1–12.

TOMASI, C. AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *ICCV 98*. IEEE, 839.

WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. 2006. Multidimensional lightcuts. *ACM Trans. Graph. 25,* 3, 1081–1088.

WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A ray tracing solution for diffuse interreflection. In *ACM SIGGRAPH '88*. ACM, New York, NY, USA, 85–92.

WHITTED, T. 1980. An improved illumination model for shaded display. *Commun. ACM 23*, 343–349.

XU, R. AND PATTANAIK, S. N. 2005. A novel Monte Carlo noise reduction operator. *IEEE Computer Graphics and Applications 25*, 31–35.

Received January 2011; accepted July 2011