Published in Image Processing On Line on 2015–03–11.
Submitted on 2014–07–07, accepted on 2015–01–20.
ISSN 2105–1232 © 2015 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2015.119

Accelerating Monte Carlo Renderers by Ray Histogram Fusion

Mauricio Delbracio[1], Pablo Musé[2], Antoni Buades[3] and Jean-Michel Morel[4]

[1] ECE, Duke University, USA (md211@duke.edu)
[2] IIE, Universidad de la República, Uruguay (pmuse@fing.edu.uy)
[3] CMLA, ENS Cachan, France and DMI, Universitat de les Illes Balears, Spain (toni.buades@uib.es)
[4] CMLA, ENS Cachan, France (morel@cmla.ens-cachan.fr)

Communicated by Guillermo Sapiro *Demo edited by* Mauricio Delbracio

Abstract

This paper details the recently introduced Ray Histogram Fusion (RHF) filter for accelerating Monte Carlo renderers [M. Delbracio et al., Boosting Monte Carlo Rendering by Ray Histogram Fusion, ACM Transactions on Graphics, 33 (2014)]. In this filter, each pixel in the image is characterized by the colors of the rays that reach its surface. Pixels are compared using a statistical distance on the associated ray color distributions. Based on this distance, it decides whether two pixels can share their rays or not. The RHF filter is consistent: as the number of samples increases, more evidence is required to average two pixels. The algorithm provides a significant gain in PSNR, or equivalently accelerates the rendering process by using many fewer Monte Carlo samples without observable bias. Since the RHF filter depends only on the Monte Carlo samples color values, it can be naturally combined with all rendering effects.

Source Code

The source code and the online demo are accessible at the IPOL web page of this article[1].

Keywords: rendering; Monte Carlo path tracing; non local filtering

1 Introduction

Synthesizing high quality realistic images in a reasonable amount of time remains a major challenge in computer graphics. The aim of realistic image synthesis is to generate new images from a complete three-dimensional description of a virtual scene. The scene description should contain at least the geometry, location and properties of objects, the camera viewpoint and a characterization of light sources. The generated picture should be as photorealistic as possible: if the three-dimensional scene is constructed and a photograph is taken from the same camera's point of view, the difference

[1] https://doi.org/10.5201/ipol.2015.119

should be negligible. Of course this requires a perfect knowledge of how the light interacts with the environment and extremely accurate material models; oversimplifications must be avoided.

The seminal paper by Kajiya [11] introduced in 1986 the *rendering equation*, an integral equation modeling the steady-state light distribution in a scene. Except for very simple scenes, analytical solutions are impossible to obtain, so most typical approaches are based on Monte Carlo numerical integration techniques. Image pixels are formed by averaging the contribution of stochastic rays cast from a virtual camera through the scene. The principal problem of Monte Carlo renderers is that the variance of the estimator decreases linearly with the number of stochastic samples. Thus the root mean square error to an ideal image decreases as the square root of the number of samples. Several hours or even days may be necessary to produce noiseless realistic images. Indeed, at present, the final image quality is indirectly topped by the available production time and computational resources.

In this paper we detail the recently introduced RHF filter [7] for accelerating Monte Carlo renderers, which are the most popular realistic renderers currently used. In order to synthesize an image with global illumination, a radiance value must be assigned to each pixel in the image. Path tracing (and more generally ray-tracing) is a popular technique for resolving the rendering equation ruling the steady state equilibrium of light. In a ray-tracing scenario, this value is computed as a weighted average of radiance values incident on the image plane, along light rays coming from the light sources, bouncing in the scene, passing through the pixel, and pointing to the virtual camera.

Unfortunately, only a finite number of rays can be cast, so the radiance value is computed only approximately. To avoid artifacts, rays are cast randomly. Mathematically, this is equivalent to solving the rendering equation through a Monte Carlo numerical integration procedure. The main problem of Monte Carlo rendering is that the variance of the estimator converges only linearly with the number of random samples. An example of a scene rendered with a varying number of rays per pixel is shown in Figure 1.

32spp 64spp 128spp 256spp 512spp

Figure 1: Example of an image rendered with Monte Carlo path-tracing. In a pure MC scenario the square error decreases linearly with the number of samples per pixel (spp), thus the convergence is quite slow.

Previous work. There are mainly two approaches to accelerate the convergence of Monte Carlo rendering to obtain good quality images. One of these approaches is *adaptive sampling*. This class of algorithms locally adapt the number of samples cast per pixel. The idea is to increase the number of rays in complex parts of the scene while maintaining a reduced number in simple parts, such as flat regions. Complex textures or defocused zones are typical elements that require large amounts of rays to be properly rendered. In [9] the authors proposed to adaptively distribute a set of samples in the full, multidimensional sampling domain where the rendering equation is computed. However, as more Monte Carlo effects are considered (e.g. depth-of-field, motion blur, area lighting), the dimension of this space becomes larger and suffers from the curse of dimensionality. One of the most significant adaptive sampling algorithms is certainly the Adaptive Wavelet Rendering [16].

This method adaptively distributes Monte Carlo samples in the screen space to reduce the variance of a wavelet basis scale coefficients. Then, the image is reconstructed from these non-uniformly distributed samples by using a suitable wavelet approximation.

The other approach is *adaptive filtering*. In this family of algorithms, the existing sets of samples are combined to produce a better pixel color estimator using ray information in a pixel and in its neighbors. Adaptive filtering may take place at sample level (i.e. primarily filtering the ray colors) or at pixel level (i.e. primarily filtering pixel color values). The simplest adaptive filters act at pixel level, like any filter used in classical image processing [10, 4, 28]. More complex filters make use of ray information available from the renderer in order to filter also at pixel level [22, 15, 6, 27]. The most sophisticated filters use additional ray information to adaptively filter the samples [24, 23, 19, 14].

The majority of these methods can actually be written as generalized versions of the bilateral filter (or the sigma-filter [13]) applying a weighted average of the samples (resp. of the pixels) in a neighborhood. The main disadvantage of traditional bilateral filters is that by comparing noisy pixel values, they cannot easily distinguish noise from intrinsic pixel variability. Thus, the clustering of similar pixels is potentially subject to errors and the filtering will result in a significantly biased image. As we will present in Section 2, in computer graphics, the statistics of ray samples permit to identify much more rigorously than in classic image processing the pixels sharing the same model. Indeed, all ray samples hitting a given pixel and its neighbors can be used for that purpose. Similar pixels can be detected by comparing their empirical ray color distributions using an adequate histogram distance. Since the order in which the samples are calculated is irrelevant, the sample color empirical distribution appears as a natural and complete descriptor of the compared sets. Figure 2 shows a small region of a Monte Carlo rendered image where two pixels are singled out. Although both pixels have different colors, their color distributions are strikingly similar. The difference in the pixel colors may be the consequence of the presence of a single very bright ray sample in one of the distributions. By comparing the ray color distributions, it is nevertheless possible to conclude that both pixels share the same "nature", while this conclusion could not be reached when only comparing the pixel values. The cornerstone of the RHF algorithm is to find and average the most similar patches by comparing the ray color sample distributions of each of their pixels.

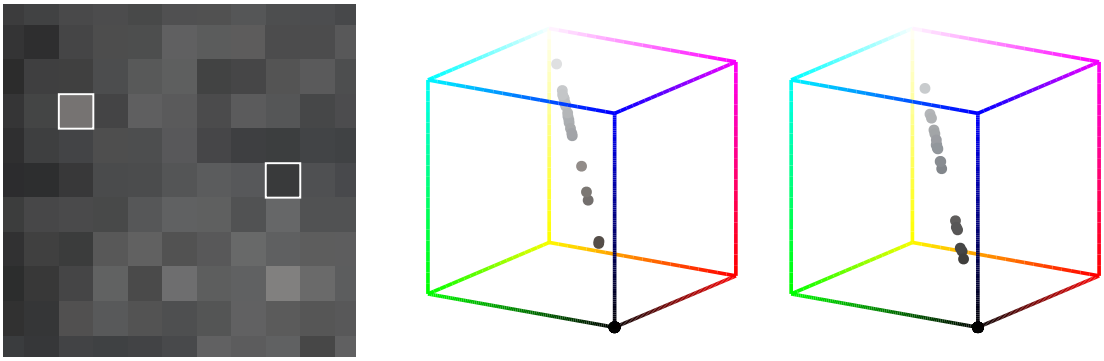


Figure 2: Monte Carlo rendered pixels can be grouped very efficiently by comparing their ray color distribution. Left: a crop of a Monte Carlo rendered image where two pixels with different colors are singled out. The difference in color is due to a poor estimate from a low number of rays cast at each pixel. Right: the color sample distributions of each pixel. The color sample distribution is represented in the RGB color box, where the color of each of the rays cast at a pixel is one point. The color distributions are strikingly similar and can be fused, which is the principle of the proposed algorithm.

In a pure Monte Carlo rendering the estimation error presents white noise characteristics meaning that all frequencies are equally contaminated by noise. The RHF has a natural multi-scale imple-

mentation that sequentially decomposes the input noisy image at each scale, filters each scale and reconstructs the multi-scale filtered image.

The RHF filter is consistent: as the number of samples increases, more evidence is required to average two pixels. Pushing the filter to its limit, two pixels will be averaged only if their color distributions are the same. Therefore, in practice, as the number of samples grows the method converges to the expected solution. The acceleration factor depends on the degree of self-similarity of the scene, which fortunately is usually high [12]. The algorithm provides a significant gain in PSNR, or equivalently accelerates the rendering process by using fewer samples without observable bias. Being based on the ray color values only, it can be combined with all rendering effects.

Table 1 summarizes the details of the adopted notation. The plan of the article is as follows. Section 2 defines a pixel similarity measure based on the corresponding cast rays color, and discusses some differences with metrics only comparing averages. In Section 3 we detail the RHF algorithm and its natural multi-scale extension. Section 4 presents several results showing the algorithm performance. We finally close with Section 5, discussing limitations of our approach and outlining future work.

2 Ray Distribution Similarity

2.1 Monte Carlo Path Tracing

The global illumination light transport problem can be stated in the space of light paths, as shown by Veach in [26]. Under this *path integral formulation*, each pixel color $u(\mathbf{x}) = (u_R(\mathbf{x}), u_G(\mathbf{x}), u_B(\mathbf{x}))$ is given by the integral over all possible light paths

$$u(\mathbf{x}) = \int_{\Omega_\mathbf{x}} f(\mathbf{p}) d\mu(\mathbf{p}), \tag{1}$$

where $\Omega_\mathbf{x}$ is the space of paths originated at pixel $\mathbf{x}$, $\mathbf{p}$ is a path of any length, and $d\mu(\mathbf{p})$ is a measure in the path-space. The function $f(\mathbf{p})$ describes the light contribution through a path $\mathbf{p}$ and is the product of several scene factors due to the interaction of light within the path plus initial self-emitted radiance and importance distributions. As a result of this formulation, the image color at pixel $\mathbf{x}$ can be estimated from $n_s(\mathbf{x})$ random paths $p_\mathbf{x}^1, \dots, p_\mathbf{x}^{n_s(\mathbf{x})}$, generated by an appropriate Monte Carlo sampling procedure. That is, if $c_\mathbf{x}^j$ denotes the color transported by random path $p_\mathbf{x}^j$ (for instance, in path tracing $c_\mathbf{x}^j = f(p_\mathbf{x}^j)$), the Monte Carlo approximation of (1) is computed as

$$\tilde{u}(\mathbf{x}) = \frac{1}{n_s(\mathbf{x})} \sum_{j=1}^{n_s(\mathbf{x})} c_\mathbf{x}^j. \tag{2}$$

Figure 3 illustrates this rendering procedure. The Monte Carlo approximation error $n(\mathbf{x})$ can then be written as $n(\mathbf{x}) = \tilde{u}(\mathbf{x}) - u(\mathbf{x})$. Even though the Monte Carlo approximation is unbiased, the mean squared error $E[n^2(\mathbf{x})]$ decays only linearly with the number of samples $n_s(\mathbf{x})$. Consequently, unless the rendering system produces thousands of samples (spending several hours or even days), the resulting images will be contaminated by noise. One possible solution to reduce the error while keeping the rendering time reasonable is to only compute a few samples, and to filter the pixel values afterwards. Filtering will result in a significant variance reduction, but, it may also increase the approximation bias. The only filtering processes that do not introduce bias are those combining pixels $\mathbf{x}$ having the same ideal value $u(\mathbf{x})$. While identifying two similar pixels $\mathbf{x}$ and $\mathbf{y}$ based on the unknown pixel values $u(\mathbf{x})$ and $u(\mathbf{y})$ is of course impossible, it is reasonable to expect that their color samples $\{c_\mathbf{x}^1, \dots, c_\mathbf{x}^{n_s(\mathbf{x})}\}$ and $\{c_\mathbf{y}^1, \dots, c_\mathbf{y}^{n_s(\mathbf{y})}\}$ will follow similar distributions. Moreover, if N

u	Digital images, defined in a rectangular grid $\mathbf{x} = (x, y) \in \{0, \dots, M-1\} \times \{0, \dots, N-1\}$.
$\tilde{u}$	Noisy digital images, generated with a finite number of Monte Carlo color samples (see (2)).
h	Color sample distribution, defined in a rectangular grid $\mathbf{x} = (x, y) \in \{0, \dots, M-1\} \times \{0, \dots, N-1\}$.
$h(\mathbf{x})$	Color sample distribution for pixel $\mathbf{x}$, $h(\mathbf{x}) \in \mathbb{R}^{n_b}$ where n_b is the number of histogram bins.
$h(\mathbf{x})[k]$	Bin k of the pixel color sample distribution h at pixel $\mathbf{x}$.
$c_{\mathbf{x}}^j$	Color sample transported by a random path $p_{\mathbf{x}}^j$ started at image plane position $\boldsymbol{\zeta} = (\zeta, \eta) \in \mathbb{R}^2$ from pixel $\mathbf{x}$.
$n_s(\mathbf{x})$	Number of color samples cast from pixel $\mathbf{x}$.
$\mathcal{C}_{\mathbf{x}}$	Set of color samples cast from pixel $\mathbf{x}$, i.e. $\mathcal{C}_{\mathbf{x}} = \{c_{\mathbf{x}}^1, \dots, c_{\mathbf{x}}^{n_s(\mathbf{x})}\}$.
$P_{\mathbf{x}}$	Patch of half size w centered at pixel $\mathbf{x}$.
$u(P_{\mathbf{x}})$	Evaluation of u on each pixel in patch $P_{\mathbf{x}}$.
$\mathsf{ReconFilter}(\boldsymbol{\zeta})$	Samples interpolation filter (e.g. box/Gaussian/Mitchell filter, see [17, Chapter 7]).
G_σ	Digital Gaussian convolution of standard deviation σ.
D_s	Gaussian subsampling operator by a factor s, $(D_s\mathbf{u})(\mathbf{x}) = G_{\sigma\sqrt{4^s-1}}u(2^s\mathbf{x})$, $s \geq 1$.
U_s	Digital bicubic interpolator by a factor s.

Table 1: Summary of the notation used in the article.

pixels share the same color sample distribution, the union of the samples can be seen as an N times larger super-set following the same underlying distribution. By simply averaging them the variance is reduced by a factor of N.

The cornerstone of the RHF filter therefore is to find similar pixels to each given pixel by comparing their underlying sample color distributions. This is what we describe next.

2.2 Color Distribution Pixel Similarity

Consider the empirical distribution of the color samples at a given pixel. In Figure 4 we depict this distribution for three different pixels on the `cornell box` scene, for samples generated by a Monte Carlo path-tracing algorithm. In this example the three pixels were selected because their colors are very similar. A quick visual inspection shows that the samples of the two edge pixels follow roughly the same color distribution, and that this distribution is conspicuously different from the one of the background pixel. This example illustrates to what extent the information provided by the sample

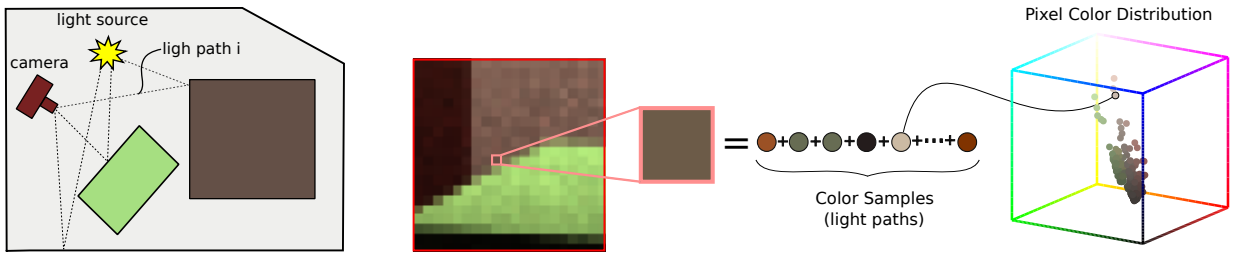


Figure 3: Pixel color computed as *average* of values along light paths cast from the image pixel, going through the camera aperture, bouncing in the scene and reaching a light source. During rendering a lot of information is computed. In particular, the color of each ray hitting a given pixel. The color distribution of the rays cast from every pixel can be generated and used to cluster similar pixels.

color distribution can help discriminate pixels of different nature, even when their colors are similar.

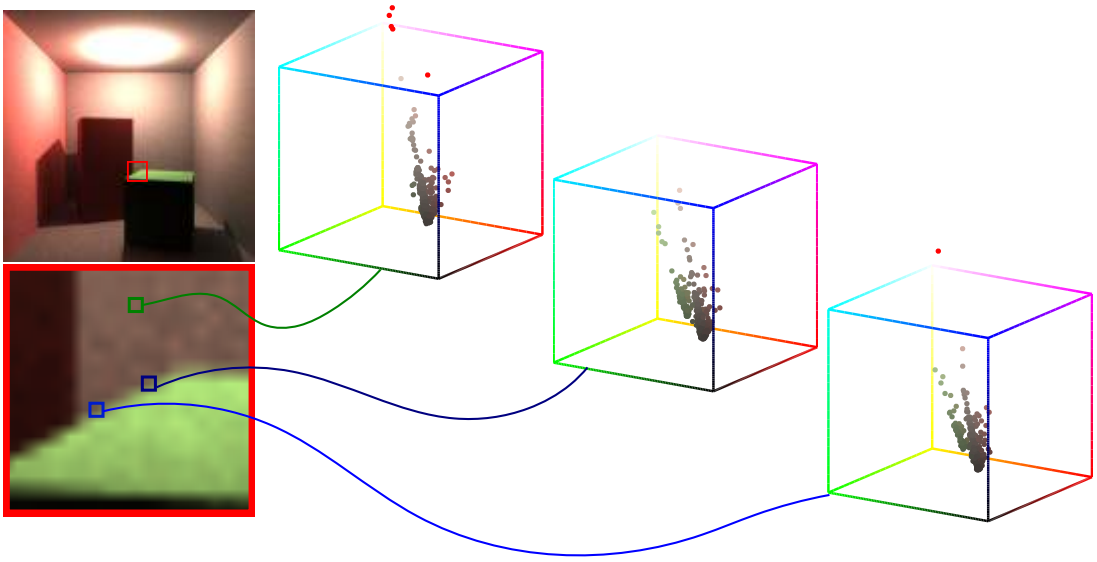


Figure 4: This figure singles out three pixels in the Cornell Box scene and their color sample distributions. (The samples with color values falling out of the $[0,1]^3-$box are by convention colored in red.) The first pixel, situated on the brown wall, has a unimodal sample color distribution. The other two pixels belong to an occlusion boundary showing a bimodal green-brown distribution.

Let us denote by $\mathcal{C}_{\mathbf{x}} = \{c_{\mathbf{x}}^1, \dots, c_{\mathbf{x}}^{n_s(\mathbf{x})}\}$ the set of color samples cast from pixel $\mathbf{x}$, and by $h(\mathbf{x})$ the corresponding empirical color distribution. To measure pixel similarity the binned empirical distributions will be used as pixel descriptors. Since in general one deals with tri-stimulus color images, we can choose to build this descriptor either as a single histogram in the three-dimensional color space, or as three one-dimensional histograms (one per color channel).

Given the color samples $\mathcal{C}_{\mathbf{x}}$ and $\mathcal{C}_{\mathbf{y}}$ at pixels $\mathbf{x}$ and $\mathbf{y}$, and their corresponding n_b-binned distributions (represented as n_b dimensional vectors) $h(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_{n_b}(\mathbf{x}))$ and $h(\mathbf{y}) = (h_1(\mathbf{y}), h_2(\mathbf{y}), \dots, h_{n_b}(\mathbf{y}))$, we consider the following metric, based on the Chi-Square distance

$$d_{\chi^2}(\mathcal{C}_{\mathbf{x}}, \mathcal{C}_{\mathbf{y}}) = \frac{1}{k(\mathbf{x},\mathbf{y})} \sum_{i=1}^{n_b} \frac{\left(\sqrt{\frac{n_s(\mathbf{y})}{n_s(\mathbf{x})}} h_i(\mathbf{x}) - \sqrt{\frac{n_s(\mathbf{x})}{n_s(\mathbf{y})}} h_i(\mathbf{y})\right)^2}{h_i(\mathbf{x}) + h_i(\mathbf{y})}, \tag{3}$$

where $n_s(\mathbf{x}) = \sum_i h_i(\mathbf{x})$ and $n_s(\mathbf{y}) = \sum_i h_i(\mathbf{y})$ are the number of color samples of $\mathbf{x}$ and $\mathbf{y}$ respectively, and $k(\mathbf{x},\mathbf{y})$ is the number of non-empty bins in $h(\mathbf{x}) + h(\mathbf{y})$. This normalization by $k(\mathbf{x},\mathbf{y})$ is necessary since only the bins carrying information should be considered in the comparison.

To take into account spatial coherence, the previous pixel-wise distance can be extended to patches of half-size w centered at $\mathbf{x}$ and $\mathbf{y}$ by

$$d_{\chi^2}(P_{\mathbf{x}}, P_{\mathbf{y}}) = \sum_{|\mathbf{t}|\leq w} d_{\chi^2}(\mathcal{C}_{\mathbf{x}+\mathbf{t}}, \mathcal{C}_{\mathbf{y}+\mathbf{t}}). \tag{4}$$

Comparing patches instead of pixels has two advantages. First, matching errors are reduced by enforcing spatial coherence. Second, the denoising will proceed by averaging similar patches. Since each pixel belongs to several patches, it will therefore receive several distinct estimates. Averaging them, an operation usually called *aggregation of estimates*, further improves the denoising performance. In practice, very small patches are used (i.e. 3×3), and thanks to the self-similarity and redundancy properties of images, many similar patches are typically found and averaged for any reference patch in the image.

The order in which the samples are calculated is irrelevant. Thus, the sample color distribution appears as a natural and complete descriptor of the compared sets. There are different ways of measuring the similarity between two distributions depending on the data type. For continuous data, the Cramer-von Mises [1, 2], the Kolmogorov-Smirnov [25, 18] or the Kantorovich-Mallows-Monge-Wasserstein distances (also known as the Earth Mover's Distance [21]) are all accepted ways to compare distributions. These three similarity measures are computed as L^p distances between the two cumulative distributions (L^∞, L^2 and L^1 respectively). For categorical data, the most popular measure to compare distributions is the χ^2 distance previously defined in (3).

By discretizing the data in a fixed number of histogram bins, the computational complexity of measuring the similarity between two data sets is kept bounded and independent of the number of samples. This is important, this distance being computed a large number of times. Thus, the color space will be divided into fixed bins, and the χ^2 distance fits well to this form of categorical data. Nevertheless, if an image is rendered with very few samples, one of the other two metrics would be preferable.

Why is it better to compare distributions than just comparing averages? State of the art image denoising algorithms measure pixel similarity by comparing pixel colors. Indeed, the bilateral filter and NL- Means replace each noisy pixel by a weighted average of the most similar ones. In the case of NL-Means, the pixel comparison is performed with patches centered around each pixel. Nevertheless, image denoising algorithms must know or measure the noise variance to evaluate properly the similarity of noisy samples. Monte Carlo rendering is an almost ideal situation where mean and variance values of the rays cast from each pixel can be directly estimated from their observed distributions. The main disadvantage of this formulation is that it cannot distinguish noise from intrinsic pixel variability. As a first example, suppose that a pixel is situated on an edge. In that case the sample color distribution will be at least bi-modal. Thus, it will probably have a large variance. This variance will result in a large tolerance to differences in the means, and consequently different pixel types may be wrongly mixed up. A case of this type is shown in Figure 4. On the other hand, by directly comparing distributions, pixels lit from several sources can be better clustered. In the case of the histogram comparison, no implicit nor explicit noise model assumption will be needed. By comparing the ray color distributions, it is nevertheless possible to conclude that pixels are of the same "nature", while this conclusion could not be reached when comparing only the averages.

2.3 Color Distribution-Driven Average

Let $\mathbf{x}$ be a pixel and $\mathcal{N}_\kappa(\mathbf{x})$ the set of pixels $\mathbf{y}$ whose centered patches $P_{\mathbf{y}}$ are such that $d_{\chi^2}(P_{\mathbf{x}}, P_{\mathbf{y}}) \leq \kappa$. If κ is such that these pixels are of the same nature as $\mathbf{x}$, the maximum likelihood estimator of the

noiseless pixel color is simply their arithmetic mean

$$\bar{u}(\mathbf{x}) = \frac{1}{|\mathcal{N}_\kappa(\mathbf{x})|} \sum_{\mathbf{y}\in\mathcal{N}_\kappa(\mathbf{x})} \tilde{u}(\mathbf{y}). \tag{5}$$

Note that one could adopt a different estimator, for example, by doing a non trivial weighted average of the similar pixels with weights depending on the pixels similarity. Some preliminary tests have not shown any particular advantage of such approach, so we opted to keep binary weights for simplicity. Nevertheless, more sophisticated estimators should be object of future investigation.

Unlike the previous estimator, where only the center of the patch is averaged, one can proceed to denoise the image patchwise. This is a very classic procedure in patch based image denoising [3, 5, 20]. Given a noisy patch $P_{\mathbf{x}}$ centered at pixel $\mathbf{x}$ its denoised version $V_{\mathbf{x}}$ is first computed by averaging all the patches being at a Chi-square distance smaller than κ:

$$V_{\mathbf{x}} = \frac{1}{|\mathcal{N}_\kappa(\mathbf{x})|} \sum_{\mathbf{y}\in\mathcal{N}_\kappa(\mathbf{x})} \tilde{u}(P_{\mathbf{y}}), \tag{6}$$

where we denoted by $\tilde{u}(P_{\mathbf{y}})$ the evaluation of $\tilde{u}$ on each pixel in patch $P_{\mathbf{y}}$.

In this way, we have denoised all patches, not just all pixels. Since each patch contains $(2w+1)^2$ pixels, each pixel is conversely contained in $(2w+1)^2$ patches and we therefore obtain a large number of estimates for its color. These estimates are finally aggregated at each pixel location to build the final denoised image:

$$\tilde{u}(\mathbf{x}) = \frac{1}{(2w+1)^2} \sum_{|\mathbf{y}-\mathbf{x}|\leq w} V_{\mathbf{y}}(\mathbf{y}-\mathbf{x}). \tag{7}$$

Taking the mean as done in the preceding formula is the simplest possible aggregation method as proposed in other denoising algorithms [3, 5].

2.4 Removing Low-Frequency Noise

In a pure Monte Carlo scenario the approximation error is a white random noise. This means that all frequencies are equally contaminated by noise. The RHF filtering procedure described so far filters noise at patch scale. Thus, long wavelength noise cannot be eliminated by this procedure, because large structures cannot be captured by small patches. Removing noise at lower frequencies therefore requires a multi-scale extension of the method.

Let

$$D_s u(\mathbf{x}) := (G_{\sigma\sqrt{4^s-1}} * u)(2^s\mathbf{x}) \tag{8}$$

be the $2^s\times$ Gaussian downsampling operator and U_s the $2^s\times$ bicubic interpolator. We denoted by $G_{\sigma\sqrt{4^s-1}}$ the convolution with a Gaussian function of standard deviation $\sigma\sqrt{4^s-1}$, where $\sigma = 0.55$. Now, for each scale s, the corresponding histograms $h^s(\mathbf{x})$ are computed as follows. Since each pixel at scale s results from the fusion of a set of neighboring pixels in the original finer scale, the new histograms are obtained by fusing the color histograms of all pixels in the same neighborhood. To obtain $h^s(\mathbf{x})$, the same down-sampling operator D_s is applied to the original color distribution $h(\mathbf{x})$. Then, at each scale, the resulting histograms are re-normalized so that the sum of their areas is preserved across scales (thus preserving the original total number of samples in the finer scale).

Given a noisy image input $\tilde{u}(\mathbf{x})$, its respective pixel color distribution $h(\mathbf{x})$, and the considered number of scales n_{scal}, the multi-scale histogram fusion proceeds as follows:

1. Generate the Gaussian multi-scale sequence: $\tilde{u}^0 = \tilde{u}$, $\tilde{u}^s = D_s\tilde{u}$, $s = 1, \ldots, n_{\text{scal}} - 1$, and their respective sample color distributions h^s.
2. Apply the RHF denoising algorithm separately to each scale to recover $\bar{u}^0, \bar{u}^1, \ldots, \bar{u}^N$.
3. Compute the final image $\bar{u} = \hat{u}_0$ by the recursion $\hat{u}_i = \bar{u}_i - U_1 D_1 \bar{u}_i + U_1 \hat{u}_{i+1}$ initialized with $\hat{u}_{n_{\text{scal}}-1} = \bar{u}_{n_{\text{scal}}-1}$.

3 Implementation Details

The RHF algorithm builds on two blocks: the estimation of each pixel sample color distribution, and a non-local multi-scale filtering based on averaging pixels having similar sample color distributions. This requires two kinds of data from the rendering system: the noisy Monte Carlo image $\tilde{u}(\mathbf{x})$ and the associated sample color histograms $h(\mathbf{x})$.

The computation of the pixel color distribution is coded on top of PBRT-V2 [17]. To fully understand how the function implementing the color distribution estimation is integrated on this system we refer the reader to the very complete and comprehensive book by Pharr and Humphreys [17] where the implementation of PBRT-V2 is detailed. In what follows we present the implementation details to estimate the pixel's sample color distribution and to perform the non-local multi-scale filtering.

3.1 Computing the Samples Color Distribution

A fundamental aspect of the method is that sample color histograms can be computed on the fly, in parallel with the Monte Carlo rendering process. This is extremely important, since it makes the memory requirements independent from the number of rendered samples.

To approximate the distribution of the samples using a histogram, the range of possible values has to be discretized into bins and the number of samples within each bin have to be counted. Smoother estimates can be produced using kernel density estimation, by interpolating the contribution of each sample using a kernel. In this work, we opted for a triangular kernel to linearly interpolate the contribution of each sample color value to its adjacent bins. Sample values have generally a three dimensional color representation. We can either compute one 3D distribution where bins are boxes in the 3D color space, or estimate three one-dimensional distributions, one for each color. Although 3D color distributions capture inter-color correlations, a much larger number of bins are required to keep the same quantization level, and consequently many more samples. Therefore, we opted to compute three one-dimensional distributions, one for each color. Let us denote by $h(\mathbf{x}) = (h_R(\mathbf{x}), h_G(\mathbf{x}), h_B(\mathbf{x}))$, the concatenation of the color histograms for each of the color channels. Thus, the total number of bins is $n_b = 3 \times n_{\text{bins}}$, where n_{bins} is the number of bins used to encode each of the color channels.

Despite the fact that the saturation value for pixels (perfect white) is one[2], the rays brightness may largely exceed that value. This does not mean that the pixel value would be saturated: indeed, pixel values are obtained by averaging sample color values. To take into account the fact that very bright samples are less frequent than low-energy ones, the bins are designed so that their sizes increase with the sample value, following an exponential law of exponent $\gamma = 2.2$. More precisely, the bins lower values b_i for $i = 0, \ldots, n_{\text{bins}}$ are computed as

$$b_i = \begin{cases} \left(\frac{M \cdot i}{n_{\text{bins}} - 2}\right)^\gamma & \text{if } i = 0, \ldots, n_{\text{bins}} - 2, \\ (M \cdot t_{\text{sat}})^\gamma & \text{if } i = n_{\text{bins}} - 1, \end{cases}$$

[2]Although this is an arbitrary choice it is consistent with the PBRT-V2 renderer.

where n_{bins} is the number of bins, $M = 7.5$ and $t_{\text{sat}} = 2$ are two constants that define the maximum value covered by the histogram. This choice was purely empirical. Algorithm 1 details the online implementation on top of PBRT-V2: given a new sample the algorithm computes the sample contribution to the pixel color distribution using linear interpolation. It is worth mentioning that although histogram comparison is not particularly sensitive to these parameters, they must be chosen to cover the dynamic range adequately.

3.2 The RHF Filter

The implementation of the RHF filter is straightforward. In addition to the parameters needed to compute the histogram, four parameters are involved in the algorithm: the number of scales n_{scal}, half the patch size w, half the search window size b, and the χ^2 distance threshold.

The search of similar patches is restricted to a window of size $(2b+1)\times(2b+1)$. This is reasonable since the probability that two patches are similar will be smaller if one is distant from the other. A threshold κ (the user parameter) is directly set on the normalized Chi-square distance. To guarantee that each patch has a minimum number of k_{NN} similar patches in the finest scale (i.e. $s = 0$), the κ threshold at this scale is set pixelwise as $\kappa_{\mathbf{x}} = \max\left(\kappa, d^{\mathbf{x}}_{k_{NN}}\right)$, where $d^{\mathbf{x}}_{k_{NN}}$ is the χ^2distance to the k_{NN} most similar patch of $P_{\mathbf{x}}$ centered at pixel $\mathbf{x}$. In the current implementation $k_{NN} = 2$.

The pseudo-code of both the filtering at each scale and the multi-scale generalization are presented in Algorithms 2 and 3, respectively. In Algorithm 2, the denoised version of patch P_i is obtained by averaging all patches Q_j such that $d^2_\chi(P_i, Q_j) < \kappa_i$.

The parameter κ controls the amount of noise that is removed, or in other words the trade-off between image smoothness and noise reduction. The optimal choice of κ depends mostly on the sample generation process, i.e. the considered renderer. An intuitive explanation for this dependence comes from the observation that the value of κ is related to the confidence associated to the color samples. If samples are computed with low confidence, the distance threshold should be less restrictive. For instance, in Monte Carlo path tracing, each sample represents the contribution of energy of a single light path, while in volumetric ray tracing each sample is computed as the average of several light paths. Therefore, the samples generated with pure path tracing have lower confidence, and this explains why the threshold should be less restrictive. Nevertheless, the tuning of κ is not time consuming. Since the distance between patches (the heaviest computational task) is independent of κ, its computation can be first performed and then several values of the parameter can be tested with practically no additional cost.

3.3 Computational Complexity and Memory Requirements

The complexity of the filtering at each scale is $O(N \cdot w \cdot b \cdot n_b)$ where N is the number of pixels, which is independent of the number of samples.

Since the low-frequency noise filtering is done on a much smaller image, the computational cost is not significantly increased. Indeed, let C be the computational cost of applying the filter at the input resolution (scale zero). Then, the cost of filtering the first scale is $\frac{C}{4}$, since the image size is one-fourth the size of the input one. By recursion, one can see that the final cost of the algorithm is $C_T = \sum_{i=0}^{n_{\text{scal}}-1} C(\frac{1}{4})^i \leq \frac{4}{3}C$. This means that the total computational cost is always upper bounded by 133% of the filtering time at the finest resolution, independently of the number of scales.

The memory consumption of the RHF filter is determined by the number of pixels in the image and the color histogram representation of each pixel. In all the examples shown here, the color distributions were computed using three histograms of $n_{\text{bins}} = 20$ bins, that is, 60 additional counters per pixel. If each counter is represented by a floating-point number, the additional memory consumption

Algorithm 1: Online Color Histogram Computation

input :

- A new color sample $\mathbf{c} = (c_R, c_G, c_B)$ and its 2D position in the image plane $\boldsymbol{\zeta} = (\zeta, \eta)$.
- Pixel $\mathbf{x}$ within the neighborhood of $\boldsymbol{\zeta}$, the number of current pixel color samples $n_s(\mathbf{x})$ and its current color histogram $h(\mathbf{x}) = (h_R(\mathbf{x}), h_G(\mathbf{x}), h_B(\mathbf{x}))$.
- The histogram parameters $t_{\text{sat}}, M, n_{\text{bins}}$ and γ.

output: updated pixel histogram $h(\mathbf{x}) = (h_R(\mathbf{x}), h_G(\mathbf{x}), h_B(\mathbf{x}))$, updated number of pixel color samples $n_s(\mathbf{x})$.

1 $w_{\boldsymbol{\zeta}}^{\mathbf{x}} = \text{ReconFilter}(\boldsymbol{\zeta} - \mathbf{x})$; *Reconstruction filter at position $\boldsymbol{\zeta}$ for pixel $\mathbf{x}$ (see caption [*])*

2 $n_s(\mathbf{x}) = n_s(\mathbf{x}) + w_{\boldsymbol{\zeta}}^{\mathbf{x}}$; *Update pixel total samples with sample contribution $w_{\boldsymbol{\zeta}}^{\mathbf{x}}$*

3 **for** *channel i in (R,G,B)* **do**

4 $v = w_{\boldsymbol{\zeta}}^{\mathbf{x}} \cdot \mathbf{c}_i$;

5 **if** $v < 0$ **then** $v = 0$; *Truncate negative values (see caption [†])*

6 $v = v^{\frac{1}{\gamma}}/M$; *Compress dynamical range and renormalize*

7 **if** $v > t_{sat}$ **then** $v = t_{\text{sat}}$; *Truncate to t_{sat}*

8 $\mathsf{fbin} = v \cdot (n_{\mathsf{bins}} - 2)$;

9 $\mathsf{ibinL} = \texttt{floor}(\mathsf{fbin})$;

Check out of bounds

10 **if** $\mathsf{ibinL} < n_{\mathsf{bins}} - 2$ **then**

11 `//inbounds`;

12 $\mathsf{wH} = \mathsf{fbin} - \mathsf{ibinL}$;

13 $\mathsf{ibL} = \mathsf{ibinL}$; *Low bin*

14 $\mathsf{wbL} = 1 - \mathsf{wH}$; *Low bin weight*

15 $\mathsf{ibH} = \mathsf{ibinL} + 1$; *High bin*

16 $\mathsf{wbH} = \mathsf{wH}$; *High bin weight*

17 **else**

18 `//out of bounds,` $v >= 1$;

19 $\mathsf{wH} = (v - 1)/(t_{\text{sat}} - 1)$;

20 $\mathsf{ibL} = n_{\mathsf{bins}} - 2$; *Low bin*

21 $\mathsf{wbL} = 1 - \mathsf{wH}$; *Low bin weight*

22 $\mathsf{ibH} = n_{\mathsf{bins}} - 1$; *High bin*

23 $\mathsf{wbH} = \mathsf{wH}$; *High bin weight*

24 $h_i(\mathbf{x})[\mathsf{ibL}]$ += wbL;

25 $h_i(\mathbf{x})[\mathsf{ibH}]$ += wbH;

[*] The reconstruction filter ReconFilter *is used to interpolate the samples near a particular pixel. The final value of a pixel is computed as a weighted average of the samples within the reconstruction filter support. For instance, the filter* ReconFilter *can be a box filter averaging with the same weight all the samples within a square window of side 1 pixel. Other popular interpolation filters are the Gaussian filter or the Mitchell filter (see [17, Chapter 7]).*

[†] Note that v can take negative values due to the use of a reconstruction filter $w_{\boldsymbol{\zeta}}^{\mathbf{x}}$ having negative side-lobes (e.g., Mitchell or Sinc filters). Nevertheless, for computing the pixel color's histogram, those negative values are clamped to zero (line 5).

Algorithm 2: Single-Scale Ray Histogram Fusion

input : MC image $\tilde{u}$, corresponding histograms h (computed by Algorithm 1), half patch size w, half search window size b, distance threshold κ, minimum number of similar patches k_{NN}.

output: Filtered image $\bar{u}$.

1 $\bar{u} \leftarrow 0$;
2 $n \leftarrow 0$; *auxiliary counter at each pixel in the image*
3 **for** *every pixel* $\mathbf{x}$ **do**
4 $P_{\mathbf{x}} \leftarrow$ patch centered at pixel $\mathbf{x}$;
5 $W_{\mathbf{x}} \leftarrow$ search window with size b for pixel $\mathbf{x}$;
6 $c \leftarrow 0$ and $V \leftarrow 0$;
7 **for** *every* $\mathbf{y} \in W_{\mathbf{x}}$ **do**
8 $Q_{\mathbf{y}} \leftarrow$ patch centered at pixel $\mathbf{y}$;
9 $d_{\mathbf{y}} \leftarrow \texttt{ChiSquareDistance}(h(P_{\mathbf{x}}), h(Q_{\mathbf{y}}))$; *Given by (4)*
10 $S_{\mathbf{x}} \leftarrow \texttt{compute\_knn}(k_{NN}, \{d_{\mathbf{y}}\})$; $S_{\mathbf{x}}$: *set of indices of* k_{NN} *most similar patches (smallest* $d_{\mathbf{y}}$ *values)*
Lines $11-18$ *implement Equation (6)*
11 **for** *every pixel* $\mathbf{y} \in S_{\mathbf{x}}$ **do**
12 $V \leftarrow V + \tilde{u}(Q_{\mathbf{y}})$;
13 $c \leftarrow c + 1$;
14 **for** *every* $\mathbf{y} \in W_{\mathbf{x}} \cap S_{\mathbf{x}}^{c}$ **do**
15 **if** $d_{\mathbf{y}} < \kappa$ **then**
16 $V \leftarrow V + \tilde{u}(Q_{\mathbf{y}})$;
17 $c \leftarrow c + 1$;
18 $V \leftarrow V/c$;
Aggregation given by Equation (7)
19 $n(P_{\mathbf{x}}) \leftarrow n(P_{\mathbf{x}}) + 1$; *+1 estimator for each pixel in* $P_{\mathbf{x}}$
20 $\bar{u}(P_{\mathbf{x}}) \leftarrow \bar{u}(P_{\mathbf{x}}) + \big(V - \bar{u}(P_{\mathbf{x}})\big)./n(P_{\mathbf{x}})$;

Notation convention: *$\tilde{u}(P_{\mathbf{x}})$ is the evaluation of $\tilde{u}$ on each pixel in patch $P_{\mathbf{x}}$ (the same applies for $\bar{u}, n, h$). The operator ./ (line 18) represents element-wise division.*

of the filter for a 1280×720 image would be approximately 0.2GB, regardless of the number of samples per pixel.

4 Examples

Different types of scenes containing complex geometries, indirect illumination, depth-of-field and other effects were rendered using the software PBRT-V2 [17]. As previously said, the color distribution estimation stage was implemented on top of PBRT, so the color histograms were produced online as the samples were computed. The filtering/reconstruction stage was implemented in a stand-alone application which makes use of the sample color histogram of each pixel and the noisy Monte Carlo image generated with a box filter.

In all cases three independent histograms were calculated, one for each channel (R, G, B) with $n_{\text{bins}} = 20$. The search for similar patches was limited to a 10×10 window centered on the filtered pixel. The patch size for the RHF filter is 3×3 ($w = 1$) for all the results shown in this paper. The

Algorithm 3: Ray Histogram Fusion

input : MC image $\tilde{u}$, corresponding histograms h (computed by Algorithm 1), half patch size w, half search window size b, distance threshold κ, minimum number of similar patches k_{NN} and number of scales n_{scal}.

output: Filtered image $\bar{u} = \bar{u}_0$.

$s \leftarrow n_{\text{scal}} - 1$ $n_T \leftarrow \sum_{\mathbf{x},k} h(\mathbf{x})[k]$ *total number of samples*
while $s \geq 0$ **do**
  $u^s \leftarrow D_s(\tilde{u})$;
  $h^s \leftarrow D_s(h)$, $n_T^s \leftarrow \sum_{\mathbf{x},k} h_k^s(\mathbf{x})$, $h^s \leftarrow \frac{n_T}{n_T^s} h^s$;
  if $s = 0$ **then**
    $\bar{u}_s \leftarrow \text{RHF}\ (u_s, h_s, w, b, \kappa, k_{NN})$; *Force a min. of k_{NN} similar patches (finest scale)*
  else
    $\bar{u}_s \leftarrow \text{RHF}\ (u_s, h_s, w, b, \kappa, k_{NN}=0)$;
  if $s < n_{scal} - 1$ **then**
    $\bar{u}_s \leftarrow \bar{u}_s - U_1 D_1 \bar{u}_s + U_1 \bar{u}_{\text{old}}$
  $\bar{u}_{\text{old}} \leftarrow \bar{u}_s$ $s \leftarrow s - 1$

κ threshold (the user parameter) was manually set to produce a good balance between smoothness and remaining noise and we always considered a minimum number of $k_{NN} = 2$ similar patches at the finest scale.

The effect of κ. The maximum distance authorized between two patches plays an important role in the bias-variance tradeoff of the method. If the threshold is conservative then very few pixels will be averaged. Thus, the filtering stage will not introduce bias, but the variance reduction will be low. On the other hand, if set too large then many pixels of different nature would be considered similar, and averaged by error. Then the resulting image would be smooth but also biased as shown in Figure 5.

The effect of the multi-scale procedure. Figure 6 shows the importance of dealing with noise at multiple scales. When filtering only at a single fine scale, conspicuous low frequency noise remains. This noise is almost completely eliminated by the multi-scale procedure with three scales.

The effect of the minimum number of similar patches k_{NN}. The minimum number of similar patches permits to force a minimum filter strength, i.e. a minimum on the variance reduction. In very complex scenes, such as the San Miguel cathedral scene shown in Figure 7, this is important since there are regions (e.g. shadows) from where paths very rarely find a way to reach a light source. When rendering with few samples, this produces impulse noise that can be mitigated by fixing a minimum number of similar patches.

5 Discussion

This paper detailed the recently introduced RHF filter [7] for accelerating Monte Carlo renderers. Although there have been several breakthroughs very recently, synthesizing high quality realistic images in a reasonable amount of time remains a major challenge in computer graphics. In this RHF filter described here, each image pixel is characterized by the set of rays that reach its surface.

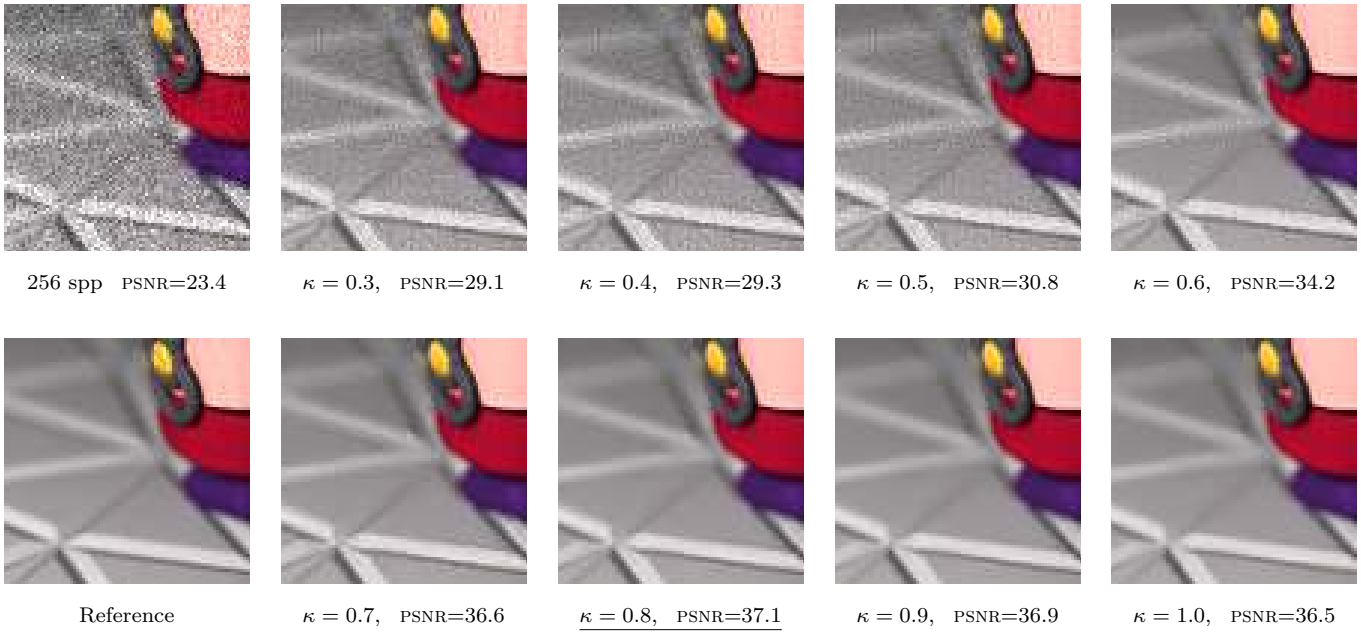
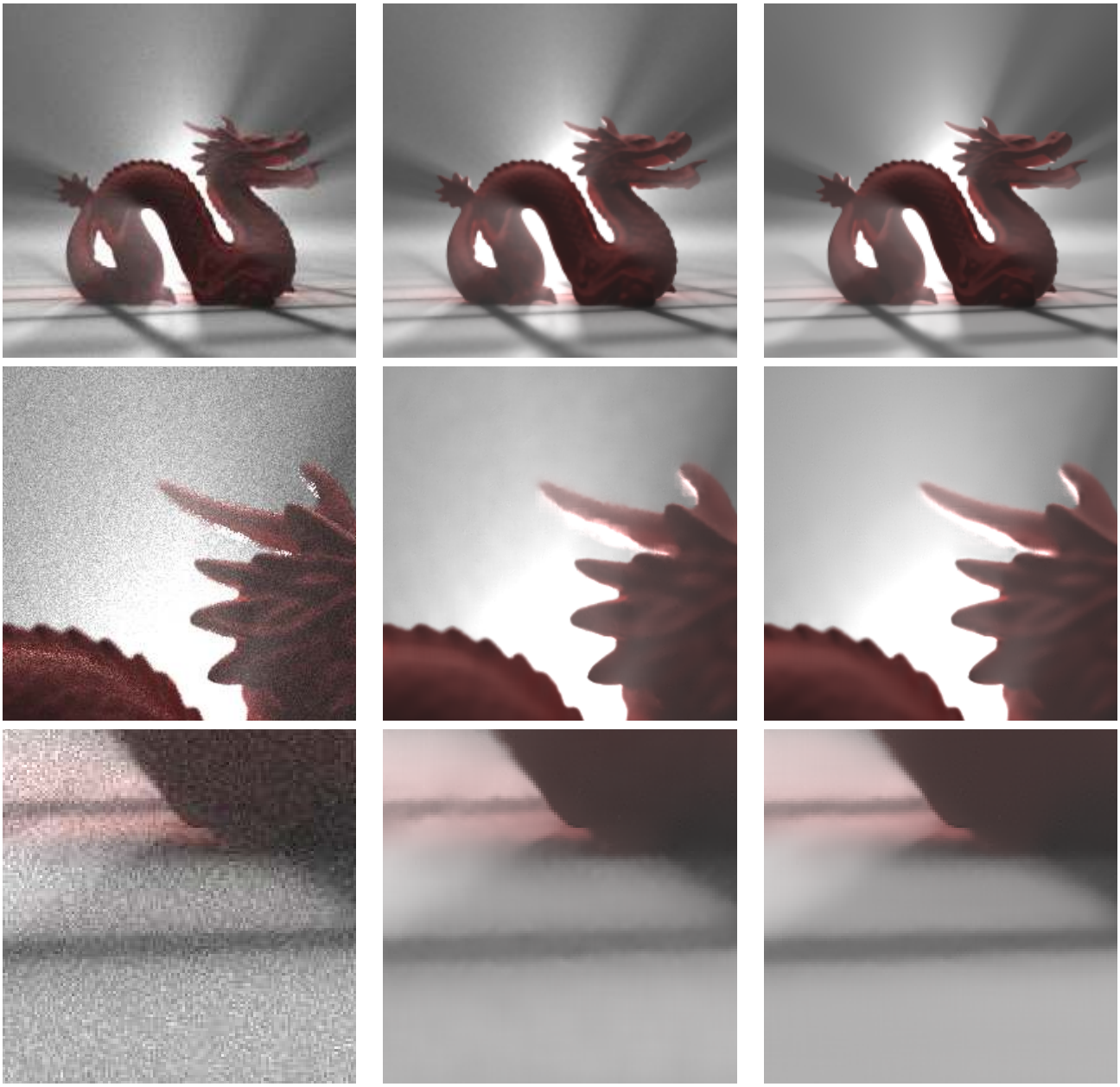


Figure 5: Changing the similarity parameter κ. This parameter controls the maximum distance that two color distributions can differ. A small detail in the `toasters` image filtered with the RHF algorithm with growing κ values. The PSNR presents a minimum for $\kappa = 0.8$. If κ is too small the test on the similarity is excessively conservative, and the noise is not reduced (high variance). If κ is large, too many pixels are averaged and the image is blurred (high bias). The results were calculated on the `toasters` scene generated with 256 samples per pixel (spp).

The algorithm uses a similarity measure on the empirical ray color distribution of each pixel, to decide whether two pixels can be fused. This simple procedure permits to boost the performance of any stochastic renderer by reusing samples without introducing significant bias. The RHF method achieves artifact-free high quality noise reduction on a variety of scenes, and is able to cope with multiple simultaneous rendering effects. Thanks to its natural multi-scale design, it can successfully remove noise at all scales.

As a future work, we would like to investigate how RHF can be applied to post-process other methods that re-synthesize samples using information from the scene, like the one recently proposed in [14]. Also, since a direct output of the method is the number of similar pixels that each pixel has, a decision on where to distribute new samples can be adopted. This may be the basis of an adaptive rendering version of the proposed filtering. The RHF filter assumes that the pixels grouped as similar have exactly the same expected value. In practice, this does not strictly hold and can therefore lead to the introduction of a small bias. We would like to explore whether the use of more general models (e.g. affine or more complex statistical models) can improve the performance and keep bias controlled.

Besides, it would also be interesting to explore different ways of reducing the computational cost of the RHF filter. Recently, [8] introduced a technique for accelerating filters based on the auto-similarity principle. Their algorithm, which reaches outstanding results, learns a set of manifolds that capture well the image structure, and then filters each of them separately. Hence, this is a natural research direction to reduce the computational cost of the RHF filter.



Input | Single scale | Three scales

Figure 6: Number of scales. Filtering only at a single fine scale will not eliminate large structure noise. However, this noise is almost completely eliminated by the multi-scale procedure with three scales.

Image Credits

We gratefully acknowledge the sources of the scenes used in this paper: CORNELL-BOX by Matt Pharr and Greg Humphreys (PBRT-V2 book), DRAGON model from the Stanford 3D Scanning Repository, SAN MIGUEL cathedral courtesy of Guillermo M. Leal Llaguno, SIBENIK cathedral by Marko Dabrovic and Mihovil Odak, and TOASTERS scene by Andrew Kensler via the Utah 3D Animation Repository.

References

[1] T. W. Anderson, *On the distribution of the two-sample Cramer-von Mises criterion*, The

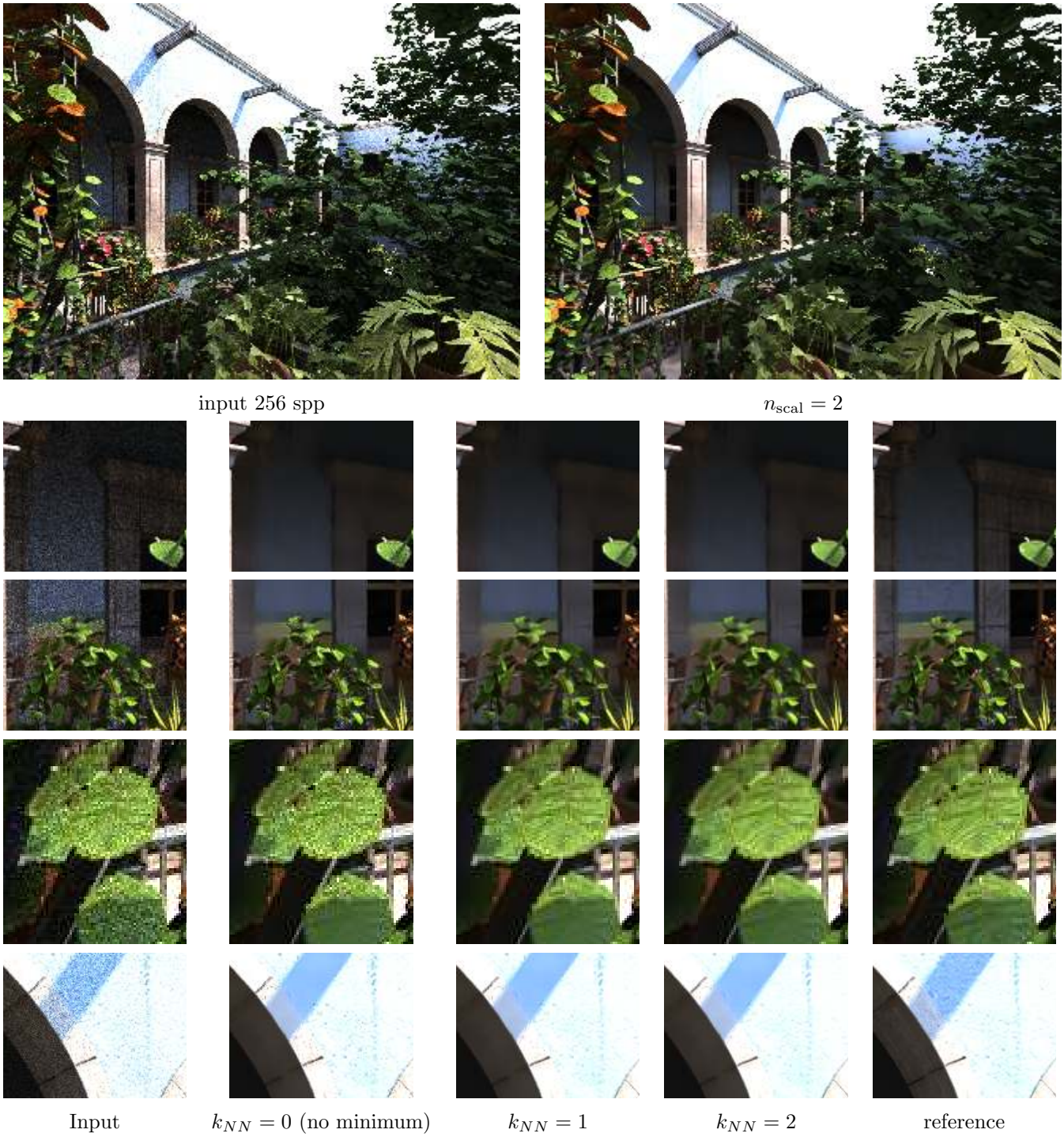


Figure 7: Minimum number of similar patches k_{NN}. In very complex scenes like the San Miguel cathedral scene shown here, there are some regions where light paths very rarely find a way to reach a light source. This can produce impulse noise that can be mitigated by fixing a minimum number of similar patches. This is equivalent to force a minimum variance reduction.

Annals of Mathematical Statistics, 33 (1962), pp. 1148–1159.

[2] T. W. Anderson and D. A. Darling, *Asymptotic theory of certain goodness-of-fit criteria based on stochastic processes*, The Annals of Mathematical Statistics, 23 (1952), pp. 193–212. http://www.jstor.org/stable/2236446.

[3] A. Buades, B. Coll, and J.-M. Morel, *A review of image denoising algorithms, with a new one*, SIAM Journal on Multiscale Modeling and Simulation, 4 (2005), pp. 490–530. http://dx.doi.org/10.1137/040616024.

[4] P. Choudhury and J. Tumblin, *The trilateral filter for high contrast images and meshes*, in Proceedings of the 14th Eurographics workshop on Rendering, 2003.

[5] K. Dabov, A. Foi, V. Katkovnik, and K. O. Egiazarian, *Image denoising by sparse 3-D transform-domain collaborative filtering*, IEEE Transactions on Image Processing, 16 (2007), pp. 2080–2095. http://dx.doi.org/10.1109/TIP.2007.901238.

[6] H. Dammertz, D. Sewtz, J. Hanika, and Hendrik P. A. Lensch, *Edge-avoiding À-trous wavelet transform for fast global illumination filtering*, in Proceedings of the Conference on High Performance Graphics, 2010.

[7] M. Delbracio, P. Musé, A. Buades, J. Chauvier, N. Phelps, and J.-M. Morel, *Boosting Monte Carlo Rendering by Ray Histogram Fusion*, ACM Transactions on Graphics, 33 (2014), pp. 8:1–8:15. http://doi.acm.org/10.1145/2532708.

[8] E.S.L. Gastal and M.M. Oliveira, *Adaptive manifolds for real-time high-dimensional filtering*, ACM Transactions on Graphics, 31 (2012), pp. 33:1–33:13. Proceedings of SIGGRAPH 2012.

[9] T. Hachisuka, W. Jarosz, R. P. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. W. Jensen, *Multidimensional adaptive sampling and reconstruction for ray tracing*, ACM Transactions on Graphics, 27 (2008), pp. 33:1–33:10. http://doi.acm.org/10.1145/1360612.1360632.

[10] H. W. Jensen and N. J. Christensen, *Optimizing path tracing using noise reduction filters*, in Proceedings of WSCG, 1995.

[11] J. T. Kajiya, *The rendering equation*, SIGGRAPH Computer Graphics, 20 (1986), pp. 143–150. http://doi.acm.org/10.1145/15886.15902.

[12] M. Lebrun, M. Colom, A. Buades, and J.-M. Morel, *Secrets of image denoising cuisine*, Acta Numerica, 21 (2012), pp. 475–576. http://dx.doi.org/10.1017/S0962492912000062.

[13] J.-S. Lee, *Digital image smoothing and the sigma filter*, Computer Vision, Graphics, and Image Processing, 24 (1983), pp. 255 – 269. http://dx.doi.org/10.1016/0734-189X(83)90047-6.

[14] J. Lehtinen, T. Aila, S. Laine, and F. Durand, *Reconstructing the indirect light field for global illumination*, ACM Trans. Graph., 31 (2012), pp. 51:1–51:10. http://doi.acm.org/10.1145/2185520.2185547.

[15] M. D. McCool, *Anisotropic diffusion for Monte Carlo noise reduction*, ACM Transactions on Graphics, 18 (1999), pp. 171–194. http://doi.acm.org/10.1145/318009.318015.

[16] R. S. Overbeck, C. Donner, and R. Ramamoorthi, *Adaptive wavelet rendering*, ACM Transactions on Graphics, 28 (2009), pp. 140:1–140:12. http://doi.acm.org/10.1145/1618452.1618486.

[17] M. Pharr and G. Humphreys, *Physically Based Rendering, Second Edition: From Theory To Implementation*, Morgan Kaufmann, 2010. ISBN 978-0123750792.

[18] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, New York, NY, USA, 3 ed., 2007. ISBN 0521880688, 9780521880688.

[19] F. Rousselle, C. Knaus, and M. Zwicker, *Adaptive sampling and reconstruction using greedy error minimization*, ACM Transactions on Graphics, 30 (2011), pp. 159:1–159:12. http://doi.acm.org/10.1145/2070781.2024193.

[20] ——, *Adaptive rendering with non-local means filtering*, ACM Transactions on Graphics, 31 (2012), pp. 195:1–195:11. http://doi.acm.org/10.1145/2366145.2366214.

[21] Y. Rubner, C. Tomasi, and L. J. Guibas, *A metric for distributions with applications to image databases*, in Proceedings of the Sixth International Conference on Computer Vision, ICCV '98, Washington, DC, USA, 1998, IEEE Computer Society, pp. 59–66. http://dl.acm.org/citation.cfm?id=938978.939133.

[22] H. E. Rushmeier and G. J. Ward, *Energy preserving non-linear filters*, in Proceedings of the 21st annual conference on computer graphics and interactive techniques, SIGGRAPH '94, New York, NY, USA, 1994, ACM, pp. 131–138. http://doi.acm.org/10.1145/192161.192189.

[23] P. Sen and S. Darabi, *On filtering the noise from the random parameters in Monte Carlo rendering*, ACM Transactions on Graphics, 31 (2012), pp. 18:1–18:15. http://doi.acm.org/10.1145/2167076.2167083.

[24] P. Shirley, T. Aila, J. Cohen, E. Enderton, S. Laine, D. Luebke, and M. McGuire, *A local image reconstruction algorithm for stochastic rendering*, in Proceedings of ACM SIGGRAPH 2011 Symposium on Interactive 3D Graphics and Games, ACM Press, 2011, pp. 9–13.

[25] M. A. Stephens, *Use of the Kolmogorov-Smirnov, Cramér-von Mises and related statistics without extensive tables*, Journal of the Royal Statistical Society Series B, 32 (1970), pp. 115–122. http://www.jstor.org/stable/2984408.

[26] E. Veach, *Robust Monte Carlo methods for light transport simulation*, PhD thesis, Stanford University, Stanford, CA, USA, 1997. ISBN 0-591-90780-1.

[27] Q. Xu, Y. Liu, R. Zhang, S. Bao, and R. Scopigno, *Noise reduction for path traced imaging of participating media*, in European Signal Processing Conference (Eusipco), Barcelona, Spain, September 2011.

[28] R. Xu and S. N. Pattanaik, *A Novel Monte Carlo Noise Reduction Operator*, IEEE Computer Graphics Applications, 25 (2005), pp. 31–35. http://dx.doi.org/10.1109/MCG.2005.31.