Boosting Monte Carlo Rendering by Ray Histogram Fusion

MAURICIO DELBRACIO
ENS-Cachan, France and Universidad de la República, Uruguay
PABLO MUSÉ
Universidad de la República, Uruguay
ANTONI BUADES
ENS-Cachan, France and Universitat de les Illes Balears, Spain
JULIEN CHAUVIER
e-on software
NICHOLAS PHELPS
e-on software
and
JEAN-MICHEL MOREL
ENS-Cachan, France

This paper proposes a new multi-scale filter accelerating Monte Carlo renderers. Each pixel in the image is characterized by the colors of the rays that reach its surface. The proposed filter uses a statistical distance to compare with each other the ray color distributions associated with different pixels, at each scale. Based on this distance, it decides whether two pixels can share their rays or not. This simple and easily reproducible algorithm provides a PSNR gain of 10 to 15 decibels, or equivalently accelerates the rendering process by using 10 to 30 times fewer samples without observable bias. The algorithm is consistent, does not assume a particular noise model, and is immediately extendable to synthetic movies. Being based on the ray color values only, it can be combined with all rendering effects.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Raytracing*; I.4.3 [**Image Processing and Computer Vision**]: Enhancement—*Filtering*

General Terms: Algorithms

Additional Key Words and Phrases: Monte Carlo rendering, global illumination, histogram distances, non-local methods, adaptive filtering

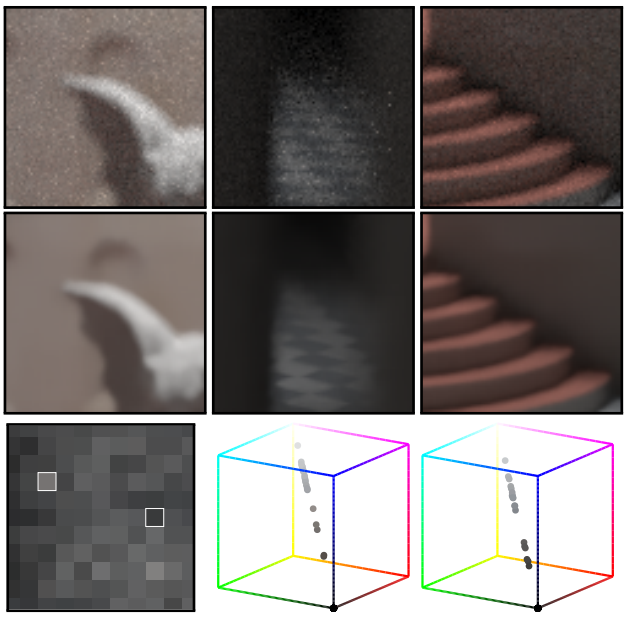

Fig. 1. Monte Carlo rendering can be boosted by detecting similar pixels and letting them share their rays. Similar pixels can be detected by comparing their empirical ray color distributions using an adequate histogram distance. In the shown example the two pixels have different colors, but their histograms are strikingly similar and can be fused. This filter increased the PSNR of this Monte Carlo image by $+11.6$ decibels. To get an equal PSNR with pure Monte Carlo, $15\times$ more samples would have been needed.

This work was partially funded by: Région Ile de France (CAP DIGITAL, DIG1-AAP FEDER 4), Office of Naval Research under grant N00014-97-1-0839, and the European Research Council, advanced grant "Twelve labours". Authors' addresses: M. Delbracio (corresponding author), ENS-Cachan, France, mdelbra@fing.edu.uy; P. Musé, Universidad de la República, Montevideo, Uruguay; A. Buades, ENS-Cachan, France; J. Chauvier, e-on software; N. Phelps, e-on software and JM Morel, ENS-Cachan, France.

DOI 10.1145/1559755.1559763
http://doi.acm.org/10.1145/1559755.1559763

1. INTRODUCTION

In computer graphics, producing high quality realistic images in a reasonable amount of time is still a major challenge. The goal of a global illumination algorithm is to estimate the light distribution in a scene. The color of each pixel in the image results from the superposition of light rays transported by an infinite number of paths that lead to it, either directly from light sources, or indirectly after bouncing in the scene. The light distribution in a scene can be obtained as a solution of the rendering equation, an integral equation that models the radiance equilibrium as a light transport in a scene [**?**]. Solving this equation for real scenes is an intractable problem. Approximate solutions are usually obtained by Monte Carlo numerical integration techniques where image pixels are formed by averaging the contribution of stochastic rays cast from the camera through the scene. The main limitation of Monte Carlo rendering is that the variance of the estimator decreases linearly with the number of stochastic samples. Thus the root mean squared error of the estimated image decreases as the square root of the number of primary rays cast from the camera (which we call samples from now on). While variance reduction techniques such as importance sampling, Russian roulette, or Markov Chain Monte Carlo methods can be used to accelerate convergence, still several hours or even days may be necessary to produce noiseless photorealistic images.

To reduce the time required by Monte Carlo rendering to produce good quality images, two main strategies have been proposed, that may be called *adaptive rendering* and *rendering post-processing*. In the first strategy, the idea is to act during the rendering process by locally adapting the number of rays cast per pixel, depending on the complexity of certain zones. Post-processing is applied once rendering has been completed, and mainly consists of filtering or interpolating either samples or pixels. Both strategies can be combined.

It is worth noting that the target quality of the images may vary depending on the application. The quality required for previsualization, where important time constraints have to be met, is clearly not as high as for applications where photo-realistically rendered scenes are an objective by themselves. In pre-visualization scenarios, computational time reduction is obtained by using the renderer to produce only a very small number of samples (say, 2 to 8 samples per pixel). In order to produce images of high enough quality from such a sparse and noisy data, it is necessary to filter or re-synthesize samples using as much information as possible. Indeed, for each sample the rendering system keeps track of relevant information associated to the ray path: geometric, color and texture features, object and material properties, Monte Carlo random parameters, etc. Using these *fat samples*, state of the art methods such as the ones proposed by **?**] or by **?**] manage to significantly improve the quality of the input sampling. However, the quality of the results obtained by this approach remains scene dependent, being potentially affected by the strong under-sampling of high dimensional data. The larger the number of effects that are simultaneously present, the higher the risks of this under-sampling. Proper up-sampling or interpolation of the sample space is therefore only possible under strong regularity conditions on the fat samples distribution. This explains why the best performances are observed for highly diffusive scenes (where impressive results are obtained from only one sample per pixel). As pointed out by **?**], poor performance is instead expected when the scene contains high frequency illumination effects, incompatible with a low sampling rate.

In short, the generation of high quality images, specially when simulating complex effects such as anti-aliasing, indirect illumination, depth-of-field, motion, requires a large number of rendered paths to correctly sample the path space. The required number of fat samples is certainly too large, not only because of the computational time that would be required to process them but, most fundamentally, because its memory storage would exceed any reasonable capacity limit (more than 100 bytes per fat sample [**?**; **?**]). The natural alternative is to give up using fat samples, and to store only part of their information. In the limit, the information can be reduced to *color samples*, that is the final color transported by each ray when hitting the screen. In this case, we say that the method works on the *screen space*, as opposed to the previous methods which work in the space of paths. Working on the screen space allows one to avoid memory saturation, while keeping a number of samples which may be large enough to capture the sample space variability. The works by **?**], **?**] and **?**] are among the most representative ones of this kind of approach.

Other similar approaches were concurrently reported during the reviewing process of our work. **?**] introduced the classical non-local means denoising algorithm in an adaptive rendering framework. Given a current distribution of Monte Carlo samples, their algorithm denoises the noisy image by using the non-local means filter. An important output of the filtering stage is an estimate of the residual per pixel error that guides the sample distribution in the next iteration. This work shows that the use of the classical non-local means denoising algorithm produces better results than previous filtering methods [**?**].

?] are among the first authors to raise and address the important question of the lack of noise model in synthetic images, which is a drastic difference with classic image processing. This paper therefore proposes a hybrid method, where in a first stage an adaptive sampling permits to equalize the noise variance throughout the image; then an adaptive classic denoiser is applied, using the noise variance information previously estimated at each pixel, to tune the denoiser.

In the present paper we propose and study a new, intermediate, filtering approach that works on the screen space but keeps and uses the color samples at each pixel. Thus, it can be coupled with any Monte Carlo renderer keeping a record of the samples' color. The cornerstone of the proposed method is to measure the similarity between any two pixels as the statistical distance between the histograms of rays color that hit them. If the comparison is positive, the ray color histograms of the similar pixels can be fused. The final color of a pixel is then obtained as the average of all ray colors of all similar pixels. This fusion is made still more reliable by comparing patches instead of pixels, and by allowing long range interaction by a multiscale procedure. The ray color histogram characterizes much better the physical and geometric properties of a pixel than just its color or the color of its neighbors.The proposed approach is related to bilateral filters, which were first applied to denoise Monte Carlo rendered scenes by **?**]. The idea of comparing patches instead of individual pixels goes back to **?**'s Non-Local Means [**?**]. Our approach still presents a fundamental difference with Non-Local Means or any classic variation of bilateral filters proposed in image processing or computer graphics: instead of defining similarity by computing distances between pixels color, we compute distances between color distributions and fuse them when the distance is small enough. For this reason, we will call this method *Ray Histogram Fusion* RHF. Distances based on distributions are of course much more informative than comparing just their averages, as bilateral filters do.

RHF is simple, easy to implement, and therefore fully reproducible. It is independent of the sample generation process. It can be easily coupled with any renderer and even with any other accel-

eration method. Most importantly, the method does not make any particular assumption on the scene. As will be demonstrated by our experiments, it therefore copes with a wide range of scenes and multiple simultaneous effects. Finally, its time and memory complexities grow linearly with the image size and are independent of the number of input samples.

The limitations of RHF are also clear. Its performance depends on the degree of self similarity of the scene, which fortunately is usually high [**?**], and the price to pay for its generality is the requirement of a relatively large number of input samples.

The plan of the article is as follows. In Section 0**??** we review relevant previous work. Section 0**??** defines a pixel similarity measure based on the corresponding cast rays color, and discusses its statistical interpretation. Section 0**??** describes the RHF algorithm, and shows how it successfully makes use of the whole ray color histogram information. Section 0**??** reports quantitative and qualitative results on the algorithm performance. We close with Section 0**??**, discussing limitations of our approach and outlining future work and conclude in Section 0**??**.

2. PREVIOUS WORK

A thorough analysis of Monte Carlo rendering is far beyond the scope of the present work. The interested reader may consult the introductory book by **?**] and the one by **?**]. However, for what follows it is enough to note that there are mainly two approaches to reduce the time required by Monte Carlo rendering to obtain good quality images.

One of these approaches is *adaptive sampling*. This class of algorithms locally adapt the number of rays cast per pixel. The idea is to increase the number of samples 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. **?**] (MDAS) 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, etc.) the dimension of this space will be larger and thus will suffer from the curse of dimensionality. One of the most significant adaptive sampling algorithms is certainly the Adaptive Wavelet Rendering (AWR) by **?**]. 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.

?] proposed to analyze the depth of field effect in the Fourier domain. By properly predicting the local bandwidth their algorithm adaptively samples the multidimensional domain. In a similar fashion, **?** [**?**; **?**; **?**] addressed motion blur, soft shadows and directional occlusions respectively, by adaptively sampling the multidimensional domain followed by a sheared reconstruction. This allowed reusing samples between pixels in specific effects.

The reconstruction scheme proposed by **?**] attempts to minimize the mean squared error. The idea is that given the current distribution of samples in the screen space, the algorithm chooses the best reconstruction filter (among a set of predefined filters - e.g. Gaussian filters) at each pixel to minimize some error criterion. Next, given the current filter selection, new samples are distributed to minimize the error. Thus, this algorithm is both an adaptive sampling and reconstruction filtering. This state of the art algorithm will be used in the experimental section for comparison.

The other approach is *denoising* or *adaptive filtering*. In this family of algorithms, the existing set of samples are combined to produce a better estimator of the pixel color using sample information in a pixel and in its neighbors. Adaptive filtering may take place at sample level (i.e. primarily filtering samples) or at pixel level (i.e. primarily filtering pixel values). The majority of these methods can actually be written as *generalized bilateral filters* [**?**] applying a weighted average of the samples (resp. of the pixels) in a neighborhood. The complexity of the method depends on whether it is applied at a pixel or sample level and how deep the method digs into the rendering information (e.g. information about each sample history: color, normal, object of the last impact; information about the random parameters used to calculate the sample) in order to compute the weights of the samples. In order to show how the conception of the proposed filter appears as the natural evolution of the previous work, we briefly present significant contributions, from an historical perspective. We will see that the general trend in this evolution is to rely more strongly in the auxiliary information available from the rendering system.

The simplest adaptive filters act at pixel level, like any filter used in classical image processing. **?**] presented a seminal work defining an alpha trimmed filter (a generalization of the median/mean). **?**] proposed to apply Gaussian or median filters with 3×3 pixels support to light having been reflected diffusely at least twice. The trilateral filter of **?**] involves an adaptive neighborhood function and the image gradient. Again a classic image bilateral filter was proposed by **?**]. Notice that unlike the work by **?**], classical bilateral filters cannot remove outliers. To overcome this limitation, the weights of the bilateral filter by **?** are computed based on a denoised version of the original image.

More complex filters make use of sample information available from the renderer in order to filter still at a pixel level. **?**] proposed to spread out noisy pixels (e.g. pixels whose variance is larger than a threshold after a fixed number of iterations) into a region of influence. A noisy pixel will contribute to several output denoised pixels, and since the filter is normalized no energy will be leaked. **?**] proposed another classical filter that uses pixels' geometric information. It is an anisotropic diffusion (of the Perona-Malik type) removing noise from Monte Carlo rendering. The conductance function that models the strength of the diffusion scheme in a pixel is estimated from a coherence map using depth and normal information gathered during rendering (contained in the G-buffer) along with a color coherence map./ More recently **?**] presented a fast wavelet filtering scheme designed for ray traced Monte Carlo global illumination images. For that purpose the filter uses RT-buffer information about direct or indirect illumination, and the buffer information on normals and position. The bilateral filter is also invoked by **?**] to denoise images created with complex light paths in smoke or fog. In this work, additional bilateral weights based on the path gradient direction are used to better guide the denoising scheme.

The last class of filters use the additional sample information to adaptively filter the sample values. **?**] addressed the question of noise in defocused or motion blurred regions. The image filter is adapted to the *a priori* knowledge of the kind of blur in a given image region. This is a very natural and successful *ad hoc* strategy for these regions. Probably the most impressive results are those recently reported by **?**]. This method uses the whole information of the rendering process and the whole information on each numerical photon to denoise by bilateral filtering. The bilateral filter takes simultaneously into account the sample position and spatial neighborhood in the image, the random synthesis parameters, the scene features (normal, world space position, texture values) and finally the sample color. It computes as a mutual information the statistical dependence on the random generation parameters of the pixels sharing the same features and colors. Although the results are out-

standing at very low sample rate per pixel, the complexity of the method makes it not scalable to generate high quality images from a large number of input samples.

?] described a reconstruction technique that allows rendering a combination of motion blur, depth of field and soft shadows by exploiting the anisotropy in the temporal light field. The effective sampling rate is increased by a large factor by efficiently reusing samples between pixels. Recently **?**] generalized these ideas to deal with indirect illumination. By contemplating the properties of diffuse surfaces, their algorithm permits to interpolate the light field to produce results similar to those that would have been obtained by rendering a much larger number of samples. For instance, from an input image of 8 samples per pixel, they synthesize images of 256 samples per pixel, whose quality is similar to 512 samples per pixel generated by standard path tracing. While the quality increase is impressive, the noise level in these images is still too strong for applications requiring high quality images. Our algorithm is complementary to this approach. Indeed, it can be used to boost the performance of a pure Monte Carlo renderer or any other set of samples like the ones generated by **?**], and can perfectly deal with a number of samples in this order of magnitude.

The above bibliographical analysis has shown that most Monte Carlo denoising methods are generalizations of the bilateral filter (or sigma-filter [**?**]). The general principle behind the bilateral filter is that similar pixels must be denoised jointly, being different samples of the same model. This is also implicitly used by the sigma-filter and by the NL-means algorithm [**?**]. In computer graphics, ray information permits to identify still 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.

3. PROPOSED APPROACH

3.1 Rationale

In contrast to classical photography where only the energy arriving at the sensor plane can be measured, in a rendering scenario much more information about the pixel formation is available. In particular, the light contribution and the screen position of each path can be stored, as well as the associated geometrical and scene information about the objects encountered along the ray path.

As pointed out by **?**], the light transport problem can be stated in the space of paths, and the global illumination can be estimated by computing a transport measure over each individual path. 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

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

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 energy 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. Thanks to this formulation, the image color at pixel $\mathbf{x}$ can be estimated from $n_\mathbf{x}$ random paths $p^1_\mathbf{x}, \dots, p^{n_\mathbf{x}}_\mathbf{x}$, generated by an appropriate Monte Carlo sampling procedure. If $c^j_\mathbf{x}$ denotes the color transported by random path $p^j_\mathbf{x}$ (for instance, in path tracing $c^j_\mathbf{x} = f(p^j_\mathbf{x})$), the Monte Carlo approximation of $u(\mathbf{x})$ is computed as

$$\tilde{u}(\mathbf{x}) = \frac{1}{n_\mathbf{x}} \sum_{j=1}^{n_\mathbf{x}} c^j_\mathbf{x}. \quad (1)$$

Consider now the Monte Carlo approximation error $n(\mathbf{x})$, given by

$$n(\mathbf{x}) = \tilde{u}(\mathbf{x}) - u(\mathbf{x}). \quad (2)$$

The Monte Carlo approximation is asymptotically unbiased, but the mean squared error $E[n^2(\mathbf{x})]$ decays linearly with the number of samples $n_\mathbf{x}$. Consequently, unless the rendering system spends several hours or even days producing samples, the resulting images will be contaminated by white noise. This is a consequence of the fact that the Monte Carlo samples are independent and therefore the random process $\{n(\mathbf{x}), \mathbf{x} \text{ image pixels}\}$ is white.

One possibility to reduce the approximation error while keeping the rendering time reasonable is to render fewer samples, and to filter the pixel values afterwards. Filtering will always result in a significant variance reduction, however, it may also severely increase the approximation bias. The only filtering processes that do not introduce bias are those that combine pixels of the same "nature", that is 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 samples color $\{c^1_\mathbf{x}, \dots, c^{n_\mathbf{x}}_\mathbf{x}\}$ and $\{c^1_\mathbf{y}, \dots, c^{n_\mathbf{y}}_\mathbf{y}\}$ will follow similar distributions. Moreover, if N pixels share the same sample color distribution, the union of the samples can be seen as an N times larger super-set following the underlying distribution. By simply averaging them the variance reduction is increased by a factor of N.

The cornerstone of the proposed approach is to find similar pixels to each given pixel by comparing their underlying sample color distributions. This is the object of the next section.

3.2 Distribution-Driven Pixel Similarity

Consider the empirical distribution of the samples color at a given pixel. Figure 0**??** depicts this distribution for five different pixels on two different scenes, for samples generated by a Monte Carlo path-tracing algorithm. In the first example (top row) the three pixels were selected because their colors are extremely close. A quick visual inspection shows that the samples of the two edge pixels follow roughly the same color distribution, and that this distribution is considerably different from the one of the background pixel. This example illustrates to what extent the information provided by the sample color distribution can help discriminate pixels of different nature, even when their pixels color are similar.

In the following, we denote by $\mathcal{C}_\mathbf{x} = \{c^1_\mathbf{x}, \dots, c^{n_\mathbf{x}}_\mathbf{x}\}$ the set of the color of samples cast from pixel $\mathbf{x}$, and by $h(\mathbf{x})$ the corresponding empirical color distribution. To measure pixel similarity we propose to use the binned empirical distributions as pixel descriptors. Since in general we deal 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 samples color $\mathcal{C}_\mathbf{x}$ 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,

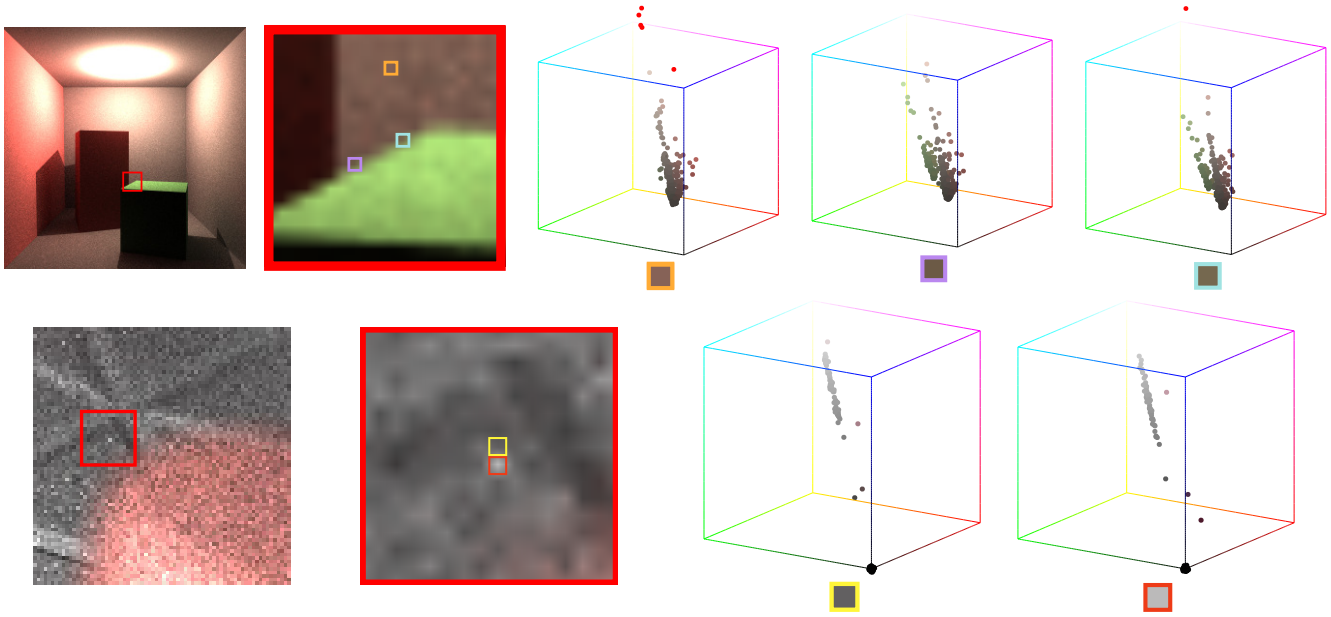

Fig. 2. The top row singles out three pixels in the Cornell Box scene and their sample color distributions. (The samples with color values falling out of the $[0,1]^3-$box are by convention colored in red.) The first pixel, situated on the brown wall, has a unimodal sample color distribution. The other two pixels belong to an occlusion boundary showing a bimodal green-brown distribution. This feature is shared by many pixels on the same boundary, which can therefore share their samples. The bottom row shows two pixels of the `toasters` scene with different colors. Their sample color distributions are nevertheless very similar and will therefore be merged as well.

based on the Chi-Square distance:

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

where $n_\mathbf{x} = \sum_i h_i(\mathbf{x})$ and $n_\mathbf{y} = \sum_i h_i(\mathbf{y})$ are the total number of samples in each set, 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.

In order 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 $\mathbf{y}$ as follows,

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

Comparing patches instead of pixels has two advantages. First, it reduces matching errors by enforcing the spatial coherence of matches. Second, the denoising will proceed by averaging similar patches. Each pixel belonging to several patches will therefore receive several distinct estimates. Averaging them again, an operation usually called *aggregation of estimates*, improves the denoising performance still further. The patches being small, in practice 3×3, and thanks to the self similarity and redundancy properties of images, many similar patches are generally found and averaged for any reference patch in the image.

Since the order in which the samples are calculated is irrelevant, 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. In the case of continuous data, the Cramer-von Mises [**?**; **?**], the Kolmogorov-Smirnov [**?**; **?**] or the Kantorovich-Mallows-Monge-Wasserstein distances (also known as the Earth Mover's Distance [**?**]) 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 (0**??**).

By discretizing the data in a fixed number of histogram bins, the computational complexity of measuring the similarity between two data sets can be kept bounded and independent of the number of samples. This is important since this similarity measure is evaluated 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 for the data. However, if an image is rendered with very few samples, one of the other two metrics would be preferable.

Note that other distances based on discretized, fixed number of bins could be used. For instance, any L^p distance between the histograms or their cumulative counterparts could be used in order to keep the computational complexity bounded, independently of the number of samples. In Section 0**??**, Table 0**??**, we present a comparison of these different discrete metrics, that shows that the χ^2 distance slightly outperforms the others.

Remark: Comparing Pixel Values vs. Comparing Distributions. 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. For a very recent review on patch based denoising methods, we refer to [**?**] and for a fast implementation to [**?**].

Nevertheless, image denoising algorithms must know or measure the noise variance to evaluate properly the similarity of noisy

samples. Fortunately, 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 0**??** (top row).

On the other hand, by directly comparing distributions, pixels lit from several sources can be better clustered. In the case of the histogram comparison, we will need no implicit nor explicit noise model assumption.

The bottom row of Figure 0**??** shows two pixels with very different pixel values. This is 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 are from the same "nature", while this conclusion could not be reached by comparing the averages.

3.3 Distribution-Driven Average

For each pixel $\mathbf{x}$, we define $\mathcal{N}_\kappa(\mathbf{x})$ as the set of pixels $\mathbf{y}$ whose centered patches $P_\mathbf{y}$ are such that $d_{\chi^2}(P_\mathbf{x}, P_\mathbf{y}) \leq \kappa$. Then, 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

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

Unlike the previous estimator, where only the center of the patch is averaged, we can proceed to denoise the whole patch, and to denoise the image patchwise. This is a very classic procedure in patch based image denoising [**?**; **?**; **?**]. In a first step, given a noisy patch $P_\mathbf{x}$ centered at pixel $\mathbf{x}$ we compute its denoised version $V_\mathbf{x}$ by averaging all the patches which are at a Chi-square distance smaller than κ:

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

where we use the convention that $\tilde{u}(P_\mathbf{y})$ is the evaluation of u on each pixel in patch $P_\mathbf{y}$.

But 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 can be finally aggregated at each pixel location in order to build the final denoised image:

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

Taking a simple mean as done in the preceding formula is the simplest possible aggregation method as proposed in other denoising algorithms [**?**; **?**]. This patchwise implementation is the one considered in this paper.

3.4 Removing Low-Frequency Noise

As already mentioned, 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 proposed filtering procedure described so far filters noise at patch scale. Long wavelength noise cannot be eliminated by this procedure, because long wavelength structures cannot be captured by small patches. Removing noise at lower frequencies requires a (straightforward) multi-scale extension of the method. Let us define two useful operators, the $s\times$ Gaussian downsampling $D_s u(\mathbf{x}) := (G_{2^s\sigma} * u)(2^s\mathbf{x})$ and U_s the $s\times$ bicubic interpolator.

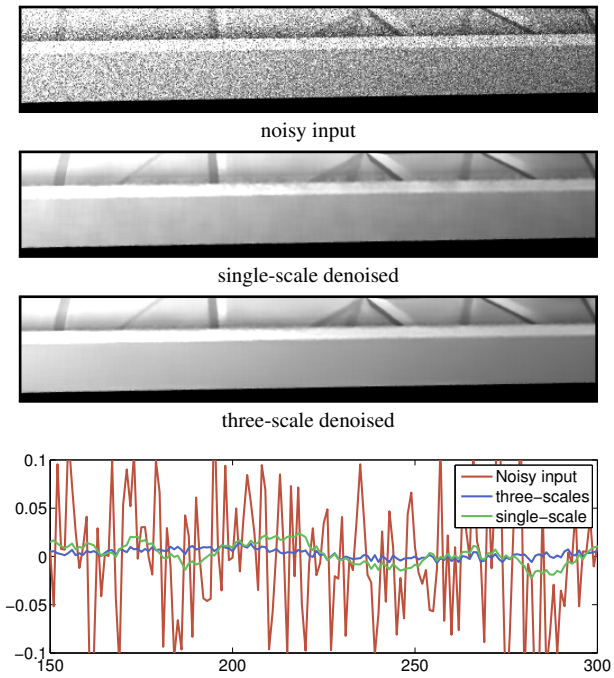

Fig. 3. The multi-scale approach eliminates low-frequency noise, as can be seen in the second and third row, and in the profile for a particular line shown in the bottom row.

Now, for each scale s, the corresponding histograms $h^s(\mathbf{x})$ have to be computed. 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 can be 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 $\tilde{u}$ and its respective pixel color distribution $h(\mathbf{x})$ the multi-scale histogram fusion proceeds as follows:

1. Generate the Gaussian multi-scale sequence: $\tilde{u}^0 = \tilde{u}$, $\tilde{u}^s = D_s\tilde{u}$, $s = 1, \ldots, N$, and their respective sample color distributions.
2. Apply the 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 $\hat{u}_N = \bar{u}_N$ for $i = N$.

Figure 0**??** 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.

4. IMPLEMENTATION DETAILS

As previously stated, our approach builds on two basic blocks: the estimation of the sample color distribution at each pixel, and a non-local multi-scale filtering based on averaging pixels sharing 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 $h(\mathbf{x})$.

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. The memory complexity bounds are fixed by the number of pixels, and therefore, as pointed out in the introduction, the input Monte Carlo images may consist of a large enough number of samples to produce high quality images.

This section gives the implementation details to estimate the sample color distribution and to perform the non-local multi-scale filtering.

4.1 Computing the color distribution of samples

To approximate a distribution using a histogram, one has to divide the range of possible values into discrete bins and count the number of samples within each bin. Smoother estimates can be obtained using kernel density estimation, by interpolating the contribution of each sample using a kernel. In this work, we used a triangular kernel to linearly interpolate the contribution of each sample color value to its adjacent bins.

Despite the fact that the saturation value for pixels (perfect white) is one, the rays brightness may largely exceed that value. This does not mean that the pixel value would be saturated: indeed, we recall that pixel values are obtained by averaging sample color values. In order to take into account the fact that high-energy (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 2.2. The range covered by the histograms is set to $[0, 7.5]$, and all samples exceeding this range are assigned to the last bin. 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.

In general, sample values have a tri-stimulus color representation. Therefore we can either compute one 3D distribution where bins are boxes in the full 3D color space, or compute three one dimensional distributions, one for each color. Although distributions in the 3D color space can capture inter-color correlations, a much larger number of bins are required to keep the same quantization step, and consequently a larger number of samples. In Section 0**??** we present a comparison of both strategies. This comparison shows that there is no advantage in using the full 3D color space.

4.2 Filtering

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_s, 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. The pseudo-code of both the filtering at each scale and the multi-scale generalization are presented in Algorithms 0**??** and 0**??**, respectively. In Algorithm 0**??**, the denoised version of patch P_i is obtained by averaging all patches Q_j such that $d^2_\chi(P_i, Q_j) < \kappa$.

Note that the only user parameter is κ. This parameter controls the amount of noise that is removed, or in other words the trade-off between image smoothness and noise. Its optimal choice depends mostly on the sample generation process (the considered renderer). The dependence on the rendered scene is actually very weak, as will be demonstrated by experiments in Section 0**??**. A simple intuitive explanation for the dependence of the optimal κ on the rendering method comes from the observation that the value of κ is related to the confidence associated to the color samples. If the samples values are computed with low confidence, the distance threshold should be less restrictive. For instance, in pure Monte Carlo path tracing, each sample carries the energy of a single light path, while in volumetric ray tracing each sample value 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.

The practical implication of this fact is that, once a rendering method has been chosen, the value of κ can be safely fixed once for all. Moreover this tuning is not time consuming: indeed, since the distance between patches (the heaviest computational task) is independent of the parameter, its computation can be first performed and then several values of the parameter can be tested with practically no additional cost.

The multi-scale implementation in Algorithm 0**??**, as detailed in Section 0**??**, sequentially decomposes the input noisy image at each scale, filters each scale using Algorithm 0**??** and reconstructs the multi-scaled filtered image.

Algorithm 1 Single-Scale Ray Histogram Fusion

Input: MC image $\tilde{u}$, corresponding histograms h, patch size w, search window size b, distance threshold κ
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 i **do**
4: $P_i \leftarrow$ patch centered at pixel i
5: $W_i \leftarrow$ search window with size b for pixel i
6: $c \leftarrow 0$ and $V \leftarrow 0$
7: **for** every $j \in W_i$ **do**
8: $Q_j \leftarrow$ patch centered at pixel j
9: $d \leftarrow \text{ChiSquareDistance}(h(P_i), h(Q_j))$
10: **if** $d < \kappa$ **then**
11: $V \leftarrow V + \tilde{u}(Q_j)$
12: $c \leftarrow c + 1$
13: **end if**
14: **end for**
15: $V \leftarrow V/c$
16: $n(P_i) \leftarrow n(P_i) + 1$ `// +1 for each pixel in` P_i
17: $\bar{u}(P_i) \leftarrow \bar{u}(P_i) + \big(V - \bar{u}(P_i)\big)./n(P_i)$
18: **end for**

Notation convention: $\tilde{u}(P_i)$ is the evaluation of $\tilde{u}$ on each pixel in patch P_i (the same applies for $\bar{u}, n, h$). The operator $./$ (line 17) represents element-wise division.

Algorithm 2 Ray Histogram Fusion

Input: MC image $\tilde{u}$, corresponding histograms h, patch size w, search window size b, distance threshold κ, number of scales n_s.
Output: Filtered image $\bar{u} = \bar{u}_0$

$s \leftarrow n_s - 1$
$n_T \leftarrow \sum_{\mathbf{x},k} h_k(\mathbf{x})$ `// total number of samples`
while $s \geq 0$ **do**
  $u^s \leftarrow D_s(\tilde{u})$
  $h^s \leftarrow D_s(h), \quad n_T^s \leftarrow \sum_{\mathbf{x},k} h_k^s(\mathbf{x}), \quad h^s \leftarrow \frac{n_T}{n_T^s} h^s$
  $\bar{u}_s \leftarrow \text{RHF}\,(u_s, h_s, w, b, \kappa)$
  if $s < n_s - 1$ **then**
    $\bar{u}_s \leftarrow \bar{u}_s - U_1 D_1 \bar{u}_s + U_1 \bar{u}_{\text{old}}$
  end if
  $\bar{u}_{\text{old}} \leftarrow \bar{u}_s$
  $s \leftarrow s - 1$
end while

4.3 Time complexity and Memory Consumption

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. Note that the computational cost is independent of the number of samples.

In the case that two scales are used the computational cost increases by about 25%, the low-frequency noise filtering being done on a four times smaller image. The computational cost is upper bounded by 133% of the filtering time at the finest resolution, independently of the number of scales being used.

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 in this paper, the color distributions were encoded using 3 histograms of 20 bins, that is, 60 additional counters per pixel. If each counter is represented by a floating-point number, the additional memory consumption of the filter for a 1280×720 image would be approximately 0.2GB, regardless of the number of samples per pixel.

5. EXPERIMENTAL SET-UP AND RESULTS

Different types of scenes containing complex geometries, indirect illumination, depth-of-field and other effects were rendered using the software PBRT-V2 [**?**]. 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.

We compared the proposed algorithm to three different methods, both regarding image quality and execution time. The first one is a pure Monte Carlo rendering (MC): this is the basic approach to generate photorealistic images. This technique is asymptotically unbiased but the variance shows slow (linear) decay with the number of samples.

The second algorithm chosen for comparison is an adaptation of the classic NL-means [**?**]. In image processing, NL-means performs denoising by averaging similar patches. In the rendering scenario, this method is obviously valid and can be improved by considering the noise level at each pixel, estimated from the variance of the samples that are cast from it. The main difference is in the way similar patches are identified. The performance comparison with NL-means will show that the knowledge of the sample color distribution adds a very significant amount of information, not yet contained in the patch colors.

Finally, we also consider comparison with the Adaptive Sampling and Reconstruction technique ASR by **?**]. As already discussed in Section 0**??**, this Monte Carlo based method can estimate the reconstruction error and control the number of samples cast from each pixel to reduce it. This method is similar to RHF in the sense that it does not rely on fat samples, and uses only the final color of each rendered sample. As such, the method scales well with the number of samples and can be used to produce high quality renderings of complex scenes. ASR is a state of the art algorithm in this class of methods. Comparison is made using the code provided by the authors, and manually setting the parameters to produce the highest possible image quality, while matching the execution time of RHF (including both the samples rendering time and the filtering stage).

The success criterion is to get an image that is very close to the ground truth in a much shorter time. Image quality is assessed by comparing results to reference images, generated by pure Monte Carlo rendering with a very large number of samples per pixel. The performance measure is the standard peak-signal-to-noise ratio (PSNR) calculated as $\text{PSNR} = 10 \log \frac{1}{\text{MSE}}$ where MSE is the mean square error to the reference image. The PSNR is a reliable criterion to characterize the quality of the reconstruction. It will nonetheless be complemented by some close-ups of difficult image details. All experiments were performed on a $2\times$ Intel Xeon CPU X5450 @ 3.00GHz (4 cores) with 16GB of RAM.

All the algorithms were run on several scenes from the PBRT software, simulating various effects with varying complexity levels.

In all cases three independent histograms were calculated, one for each channel (R, G, B) with `nbins=20`. The search for similar patches was limited to a 13×13 search window centered at the filtered pixel. The patch size for the RHF filter is 3×3 ($w = 1$) for all the results shown in this paper. The κ threshold (the user parameter) was manually set to produce a good balance between smoothness and remaining noise. As previously explained, the optimal value for this trade-off depends mostly on the rendering method. The values of κ that were chosen in the experiments are shown in Table 0**??**. Note that for all the renderings performed with a pure path tracing, we set $\kappa = 1$, while for the scene rendered using volumetric ray tracing (`plants-dusk`), $\kappa = 0.37$. This is consistent with the fact that the color samples generated with volumetric ray tracing result from an average of several light paths, and therefore are more precise than in pure path tracing.

A summary indicating all the considered effects, rendering method and image size is shown in Table 0**??**.

Table I. Summary of the tested scenes.

Scene	Effects	Size	Generation	κ-RHF
`cornell-box`	AI	256×256	path tracing	1.00
`toasters`	AILD	512×512	path tracing	1.00
`plants-dusk`	ALPD	1920×1080	ray tracing	0.37
`sibenik`	AILD	1024×1024	path tracing	1.00
`yeahright`	AI	800×800	path tracing	1.00
`dragons`	AILDP	1024×1024	photon mapping + final gathering	0.60

Considered effects: anti-aliasing (A); indirect illumination (I); area lights (L); depth-of-field (D); participating media (P). The scenes `cornell-box`, `plants-dusk` and `sibenik` are from [**?**] while the scenes `dragons` and `toasters` were taken from [**?**].

3D vs $3\times$ *1D Histograms.* Table 0**??** illustrates the performance of the method as a function of the number of bins of the rays color histogram. The experiments do not support the use of 3D color space bins. Thus, independent histograms were generated for the R, G, B channels. The number of bins must be large enough to capture the histogram structure, but not too large to grant a robust histogram comparison.

Robustness: Comparing Means vs. Comparing Distributions. Suppose that an external Oracle tells us the exact number of closest patches that should be averaged in each pixel to minimize the MSE. By computing the resulting PSNRs, we can compare the maximum theoretical performance that can be obtained using color distributions versus pixel colors. A comparison for different patch sizes is given in Table 0**??**. The results show that using histogram information to compare patches permits to better discriminate similar and non similar patches, thus yielding a better PSNR.

Comparisons for several scenes. RHF systematically outperforms ASR, as shown in Figure 0**??**. Even if in some scenes both algorithms reach similar PSNRs, the proposed algorithm does not introduce artifacts while ASR often fails to capture the geometry and causes spots. The PSNR gain by RHF filtering is significantly larger than the one that would be obtained by generating more Monte Carlo samples using the same time span. Indeed, a 3db PSNR gain by a pure MC algorithm requires to double the number of samples. Instead, the filtering increases the PSNR by 15 to 20 decibels. This amounts to decreasing the overall sampling time by a factor ranging from 15 to 40.

In the case of the `yeahright` image in Figure 0**??** the ASR algorithm produces a slightly better PSNR. This scene is best suited for this algorithm, because several regions are flat and ASR can distribute most of the samples in the problematic parts. Nevertheless, in the shadows RHF produces a more natural smooth result. As previously commented, the NL-Means based approach cannot distinguish between a large histogram variance due to pixel complexity, from a variance due to MC noise. This fact is well illustrated in the metal edge of Figure 0**??**, which is removed by NL-Means, while it is well preserved by RHF.

The `plants-dusk` scene with participating media, in Figure 0**??** is a very challenging one. Here, the principal problem is the complex geometry of the vegetation. The proposed algorithm tends to blur and to slightly flatten some texture details. Nevertheless, contrarily to ASR, no artifacts are introduced.

The proposed RHF filter works with motion and defocus blur exactly as for the other effects. Figure 0**??** shows how defocus blur is correctly filtered, while noiseless textures remain sharp (see caption for details). Figure 0**??** illustrates the same denoising effect in blurry parts on the dragons scene simulating a depth-of-field effect and strong motion blur.

To illustrate the fact that the proposed denoising method is independent from the rendering system, in Figure 0**??** we present a filtering experiment that runs on a image generated by photon-mapping and final gathering. This scene comes from [**?**]. The noise has been properly removed, and no artifacts are observed.

Finally, Figure 0**??** shows a comparison of ASR and the proposed method on the very realistic `san miguel` courtyard scene. This image presents objects with very fine geometry and complex corridors where it is very difficult to capture all the details with few samples. Nonetheless, the proposed algorithm produces an image with acceptable quality and PSNR.

Extension to animated sequences. The ideas behind this approach can be immediately extended to video sequences where pixels on neighboring frames can be included in the search box. The supplementary video shows the result of denoising an animated sequence, by implementing this simple generalization. Similar patches are searched within a temporal search window of size 3 (namely in the previous, actual and next frame). Although no explicit temporal correlation is enforced, the filtered sequence does not show significant flicker. This is a consequence of the stabilization provided by the multiscale procedure.

Table II. Performance comparison: estimating the histogram.

	$3D$			$3 \times 1D$				
	3^3	4^3	5^3	3×5	3×10	3×15	3×20	3×25
`cornell-box`	40.2	41.0	41.7	39.8	41.6	42.5	43.0	43.3
`toasters`	30.5	31.3	31.7	30.4	32.2	32.8	33.1	33.3
`plants-dusk`	46.2	46.1	46.0	46.2	46.0	45.9	45.8	45.7
`yeahright`	36.0	36.2	36.2	36.3	36.1	36.0	36.0	36.0
`sibenik`	41.4	42.5	43.1	40.3	43.1	43.8	44.2	44.3

Average of the $k = 15$ closest neighbors, the performance metric is the PSNR with respect to a ground-truth image. Two different ways of estimating the ray color histogram: bins in the 3D color space or three one-dimensional histograms one for each color. In most cases, taking more bins increases the performance, but also the computational cost of the algorithm. Estimating three independent histograms one for each color gives better results than estimating the histogram in the original 3D color space.

Table III. Oracle performance comparison.

	NL-means			RHF		
	1×1	3×3	5×5	1×1	3×3	5×5
`cornell-box`	33.4	40.7	44.4	40.1	48.1	49.9
`toasters`	24.0	30.6	34.7	29.8	37.9	40.5
`plants-dusk`	49.0	54.4	55.3	51.4	55.4	55.8
`yeahright`	36.3	43.2	46.0	39.7	46.6	48.2
`sibenik`	34.9	41.5	45.4	40.4	48.6	50.8

For each pixel the ideal number of closest patches to minimize the error with respect to the ground-truth image has been computed and fixed. The table compares NL-means and RHF for different patch sizes. It shows that RHF permits to reduce significantly the patch size.

Table IV. Performance comparison of different histogram metrics.

	χ^2	L^1	L^2	L^∞	$L^{0.7}$	AL^1	AL^2	AL^∞	$AL^{0.7}$
`cornell-box`	43.1	43.2	**43.4**	42.7	42.9	41.2	41.7	42.4	40.9
`toasters`	**33.4**	31.2	30.3	29.0	31.7	29.5	29.3	30.3	29.6
`plants-dusk`	**46.6**	46.1	46.4	45.8	45.8	46.2	**46.6**	46.0	45.9
`yeahright`	35.9	36.1	36.1	35.1	35.8	36.7	**36.8**	35.9	36.5
`sibenik`	**44.3**	43.3	42.8	42.2	43.3	41.6	41.7	42.5	41.4
global performance	**1.03**	1.01	1.01	0.98	1.01	0.99	0.99	1.00	0.98

To illustrate the influence of the metric selection, for each patch in the noisy input image, we computed the average of the 15 closest patches according to the corresponding histogram distance: χ^2 and L^p distances between histograms, as well as their cumulative counterparts (denoted by AL^p). The global performance for the whole test set was computed by normalizing each of the PSNR by the mean performance on that particular image, and then averaging all the normalized PSNR. The χ^2 distance slightly outperforms the others.

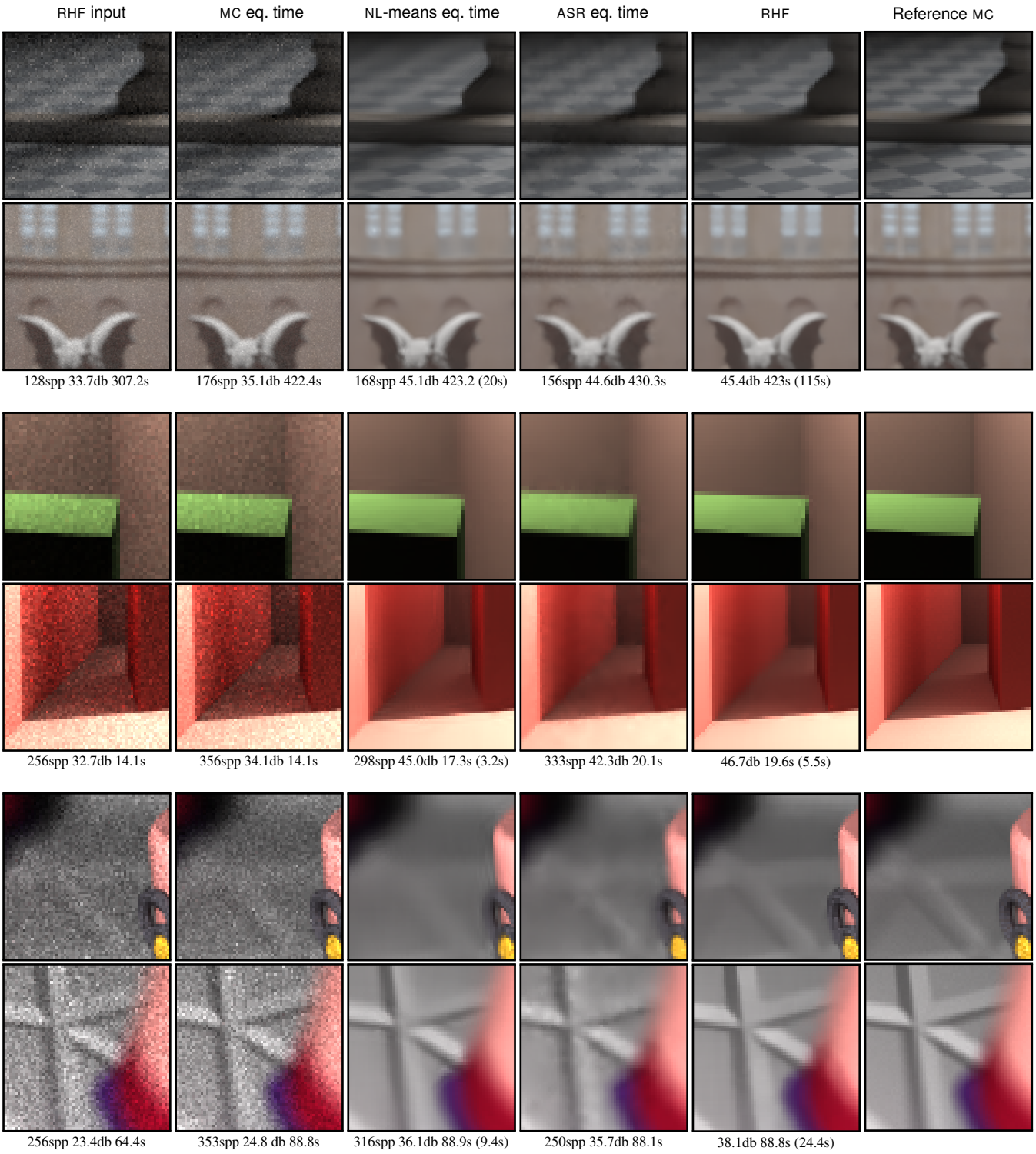

Fig. 4. Results in a variety of scenes, with close-ups on difficult parts. In all cases, the PSNR values are given for the whole image. In general ASR tends to create artifacts near edges. NL-means destroys complex edges such as the one on the bottom left box of the Cornell image. RHF produces the best PSNR, with no visible artifacts. The reference images were generated with 65536 samples per pixel. For NL-means and RHF the indicated time follows the format *total time (filtering time)*.

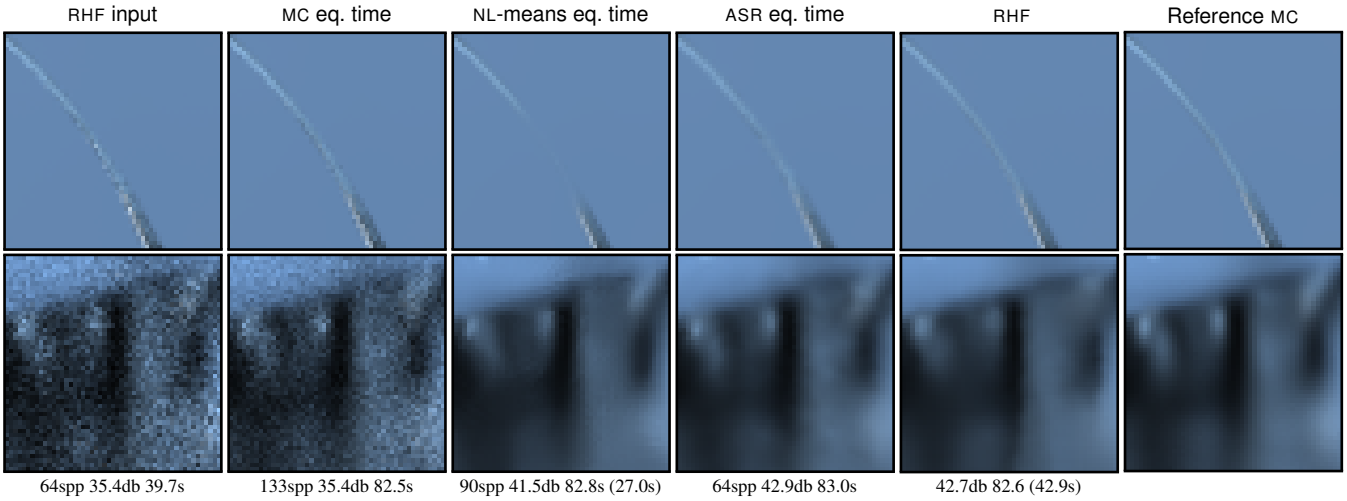

Fig. 5. Fine geometry details, glossy surfaces and indirect illumination presented in the `yeahright` scene are rendered with the PBRT-V2 path-tracing algorithm. The comparison is done in such a way that the ASR computational time matches the Monte Carlo samples generation + RHF filtering time. NL-means looses fine structures such as the thin metallic edge (first line). In this particular scene, ASR performs well, but creates artifacts in the shadow (second line). RHF produces a similar PSNR with no artifacts. The reference image was generated with 65536 samples per pixel. For NL-means and RHF the indicated time follows the format *total time (filtering time)*.

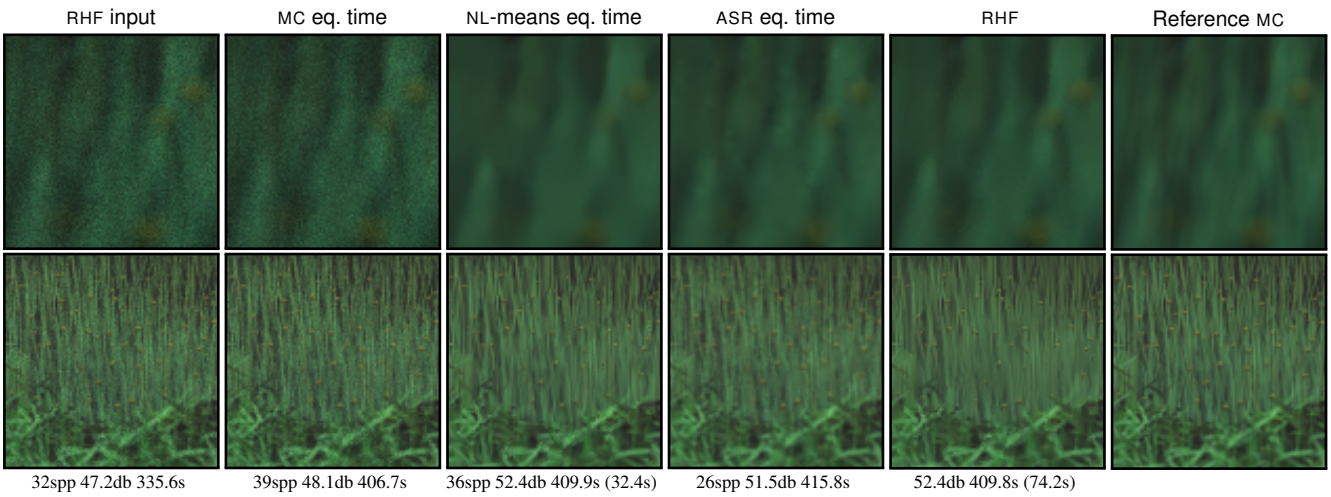

Fig. 6. Comparison of Monte Carlo rendering, ASR and the proposed methods on the `plants-dusk` scene. The original Monte Carlo image was rendered with 32 spp. The PSNR values are given for the whole image. This image presents a very complex fine geometry which is very difficult to capture with few samples. Nonetheless, the proposed algorithm produce acceptable quality and PSNR. The reference image was generated with 4096 samples per pixel. For NL-means and RHF, the indicated time follows the format *total time (filtering time)*.

6. DISCUSSION, LIMITATIONS AND FUTURE WORK

The maximum distance authorized between two patches plays an important role in the bias-variance tradeoff of the method. If the threshold is set in a conservative way 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 (see Figure 0**??**).

If we accept that the selection of the most similar pixels for each noisy input pixel is independent of the number of samples, then the gain in PSNR when casting more samples is only due to the averaging of less noisy pixels. By the randomness of the MC rendering, the noise of the input pixels is reduced by +3db/octave, thus the ideal (best) slope should be +3db/octave. This is the ideal, because it assumes that there is no error in the selection of similar pixels. Therefore, we can consider the difference in slope to the ideal +3db/octave as a measure of *experimental bias* (introduced error). While the proposed algorithm RHF has an experimental bias of 0.2db, the NL-means bias is three times larger. More important, the relative bias to the MC gain is $0.2/3 \approx 0.07$, which demon-

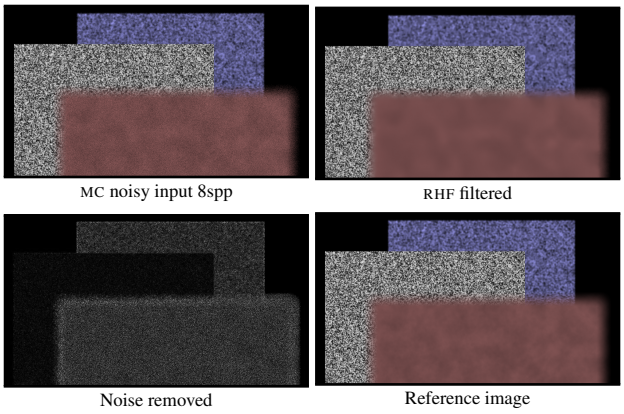

Fig. 7. Texture with noise characteristics rendered simulating a very shallow depth-of-field effect at 8 samples per pixel (spp). The texture in the in-focus plane presents white noise characteristic so classical image denoising filters will remove the textured detail. The same texture in an out-of focus plane presents Monte Carlo noise due to the random sampling of the aperture point. The RHF filter only smoothes those regions that are out-of-focus since the others do not introduce Monte Carlo noise. The reference image was generated with 2048 samples per pixel.

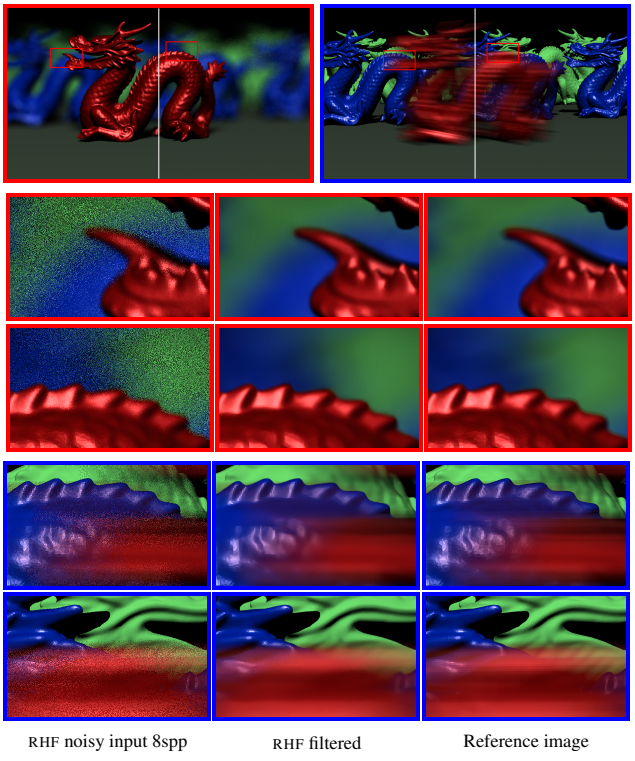

Fig. 8. Depth-of-field (red) and motion blur effects (blue). Noise is removed while details in noiseless parts are preserved. The reference images were generated with 8192 samples per pixel.

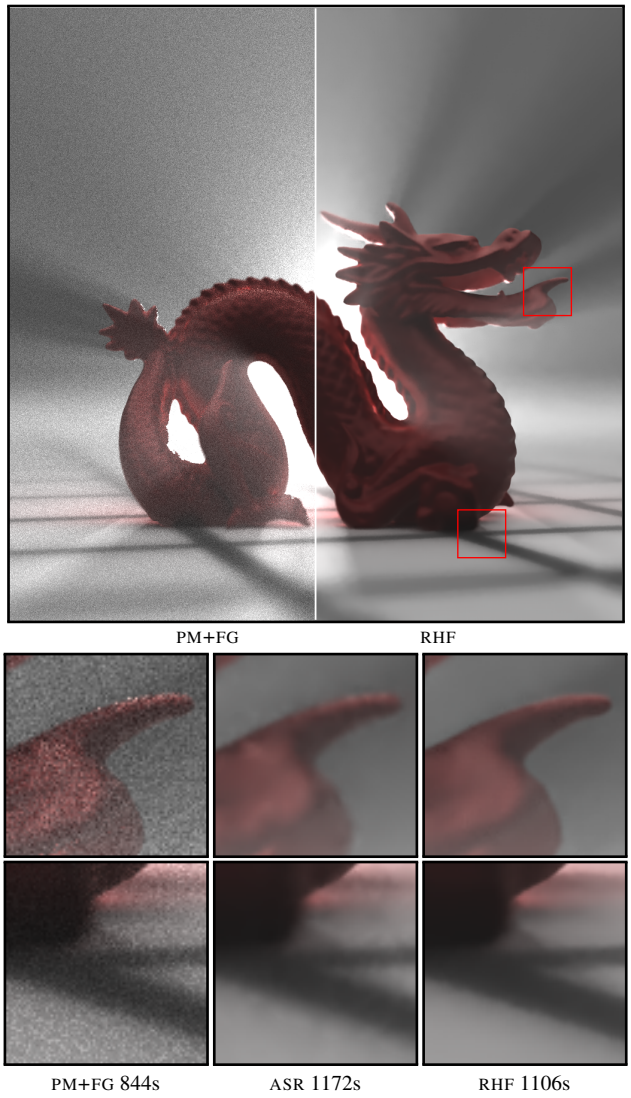

Fig. 9. Light interaction with participating media rendered through a photon mapping algorithm shows the generality of the proposed filtering. The comparison is done in such a way that the ASR computational time matches the Monte Carlo samples generation + RHF filtering time.

strates that the proposed algorithm makes very few wrong ray color attributions.

Moreover, the proposed algorithm is consistent up to the discretization of the color distribution. As the number of samples increases, more evidence is required to average two pixels. In the limit two pixels will be averaged only if their color histograms are the same. Therefore, in practice, as the number of samples grows the method converges to the expected solution, as illustrated by the experiment in Figure 0**??**.

The sharing of radiance estimates across pixels may tend to smooth and slightly flatten textures that are not highly-contrasted. This results from the averaging nature of the filter and the difficulty to differentiate pixels with similar color histograms. Nevertheless, as we have illustrated in Figure 0**??**, the introduced smoothing is moderate, and lower than the smoothing caused by a classic self-similarity filter such as NL-means.

Fig. 10. Comparison of ASR and the proposed method for the `san miguel` scene. The original 1357×986 Monte Carlo image was rendered with 256 spp. The PSNR values are given for the whole image. This image presents a very complex fine geometry and complex corridors where it is very difficult to capture all the details with few samples. Nonetheless, the proposed algorithm produces acceptable quality and PSNR. The reference image was generated with 65536 samples per pixel. For RHF, the indicated time is the the total time *rendering* + *filtering*.

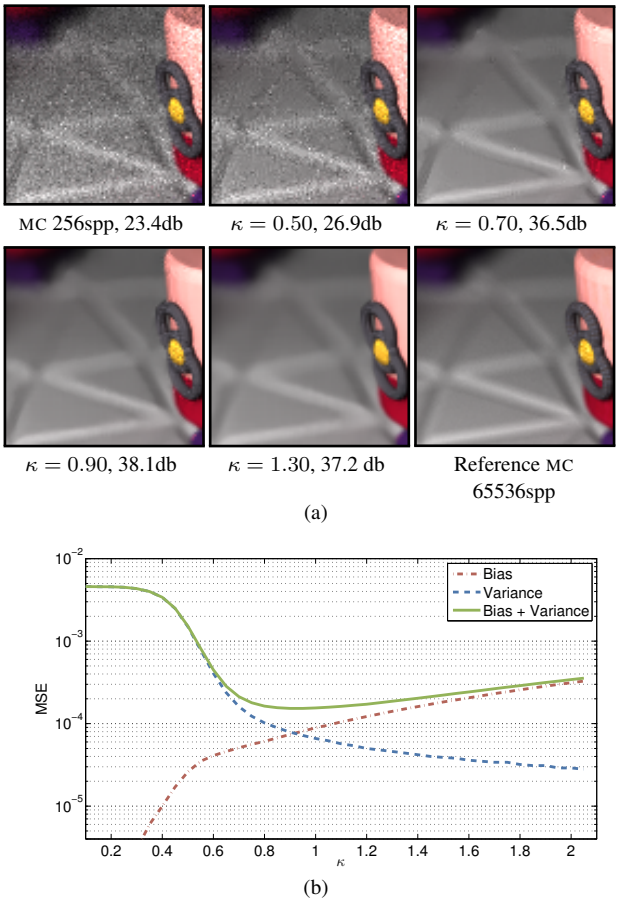

Fig. 11. Changing the distance threshold κ: it fixes the maximum distance that two color distributions can differ. (a): a small detail in the `toasters` image filtered with the RHF algorithm with growing κ values. The MSE presents a minimum for $\kappa = 0.7 - 1.0$ (b): If κ is too small the test on 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 spp and the reference image with 65536 spp.

The acceleration factor with respect to pure Monte Carlo rendering depends on the degree of self similarity of the scene, which fortunately is usually high [**?**]. Besides, in order to capture details, pixels need a large enough number of color samples to be well characterized. This is actually a design decision: we wanted our method to produce unbiased high quality images for any kind of scenes and complex effects, and this naturally requires a proper sampling of the light field. If this requirement is not met, the algorithm may not properly cluster similar pixels and details may be removed due to over-blur, as it happens with some details in the `plants` image. In the case of very low sample numbers, if the path-space is regular enough to be well described by sparse sampling, methods based on strong scene hypothesis that use fat samples [**?**; **?**] are certainly much more adapted. This will be more clear from the comparison to [**?**] that follows.

Comparison to RPF *[**?**].* In RPF "fat samples" namely a Monte Carlo sample with its geometric features, texture, color, and random generation parameters are being considered as input while RHF uses only "color samples", namely the color of each final sample. The fat samples in a given pixel permit to establish a (rough) statistics for each sample feature (mean and variance). These will be used to define the set of similar samples in a spatial neighborhood to the fat samples of the pixel. In other terms, in RPF individual samples are compared individually to the sample distribution inside a pixel. This is the set of samples whose color will be averaged to define the new color of the original sample. In RHF, the histograms of any two pixels samples color (below some maximal distance) are compared by the χ^2 distance. Thus the distance is be-

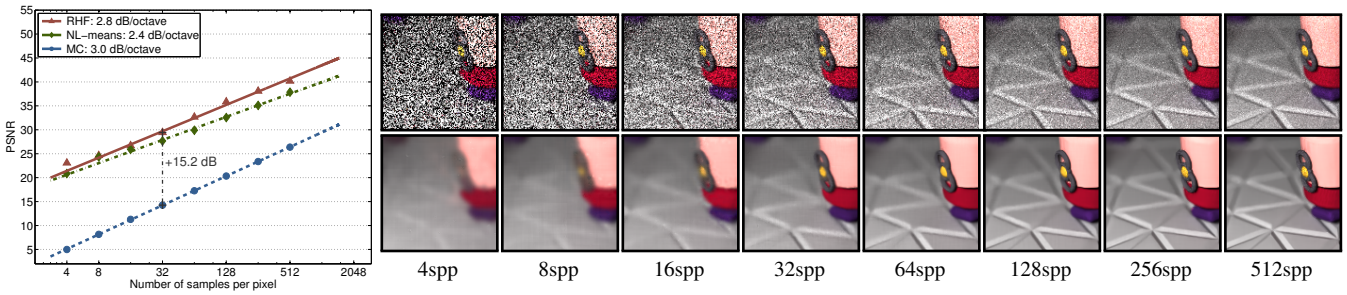

Fig. 12. Left: A comparison of the PSNR for RHF, NL-means, and the pure MC pathtracing algorithm on the `toasters` scene. As the number of samples per pixel increases the PSNR increases. In a pure MC scenario the square error decreases linearly with the number of samples (which is a trivial consequence of averaging independent samples). Thus, duplicating the number of samples produces a 3db gain. Although for the proposed algorithm the slope gain is a little smaller (2.8db/octave), RHF reduces the error significantly in comparison to MC. +15.5dB is a huge difference; it permits to reach the same image quality with $35\times$ fewer samples. On the right we show a close up, generated with a varying number of samples through pure MC (top) and filtered with RHF (bottom).

tween two pixels and not an asymmetric distance between a sample and a group of samples in the pixel.

In RPF the average of "similar" fat samples will be weighted to take into account the similarities of samples and their dependency on the generation parameters. To do so a mutual information based coefficient is computed between sample features and generation parameters; this permits to fix the weights in the final color averaging procedure. However, in RHF, the distance between two color histograms of two pixels fixes, by a binary decision, if a pixel will share its samples with another or not. In case where the decision is to fuse these pixels, both pixels share their rays. The method is made more selective by comparing not pixels, but patches of 3x3 pixels. In addition, RHF is multiscale, to ensure that low frequency noise is also removed.

Hence, note that while both methods propose to share similar samples between pixels, the decisions that are taken and the information that is used to reach this goal are, according to the above comparison, significantly different. In short, both methods group samples, but they do not use the same information to group them. We can now explain why RHF needs more samples per pixel than RPF: since we are only comparing colors, obviously very similar pixels may have very different color histograms when only very few samples are available. Thus, RHF might become efficient at low sample rate by using, as RPF, more sample features acting like weak classifiers to discriminate the right similar pixels.

7. CONCLUSION

In this paper we have introduced RHF, an adaptive filtering scheme that accelerates Monte Carlo renderers. In the proposed approach, each pixel in the image is characterized by the collection of rays that reach its surface. The proposed filter uses a distance based on the sample color distribution of each pixel, to decide whether two pixels can share their samples. This permits to boost the performance of a Monte-Carlo render by reusing samples without introducing significant bias.

We have presented several experiments showing that RHF achieves artifact-free high quality noise reduction on a variety of scenes, and is able to cope with multiple simultaneous effects. The method is not only capable of removing high frequency noise: thanks to its natural multi-scale design, it can also successfully remove low-frequency noise. The proposed method can be easily extended to process animated sequences.

The method is independent of the rendering system and can be applied to samples generated by different methods, such as pure Monte Carlo path-tracing or photon-mapping with final gathering. It could also be potentially applied to post-process other methods that re-synthesize samples using information from the scene, like the one recently proposed by **?**]. An advantage of the proposed filter is that its time and memory complexities do not depend on the number of input samples, and scale linearly with the image size.

Finally, since a direct output of our method is the number of similar pixels for each given pixel, a decision on where to distribute new samples can be adopted. This may lead to an adaptive rendering version of the proposed filtering approach.

Acknowledgments

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, PLANTS-DUSK scene by **?**], SAN MIGUEL cathedral courtesy of Guillermo M. Leal Llaguno, SIBENIK cathedral by Marko Dabrovic and Mihovil Odak, TOASTERS scene by Andrew Kensler via the Utah 3D Animation Repository, and YEAHRIGHT model courtesy of Keenan Crane. This work was partially funded by: Région Ile de France (CAP DIGITAL, DIG1-AAP FEDER 4), Office of Naval Research under grant N00014-97-1-0839, and the European Research Council, advanced grant "Twelve labours".