Ray Histogram Fusion

Mauricio Delbracio

July 24, 2014

1 Ray Distribution Similarity

1.1 Monte Carlo Path Tracing

The global illumination light transport problem can be stated in the space of light paths, as shown by Veach in [10]. 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 $p^1_\mathbf{x}, \dots, p^{n_s(\mathbf{x})}_\mathbf{x}$, generated by an appropriate Monte Carlo sampling procedure. That is, 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 (1) is computed as

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

Figure 1 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

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 $(s_x, s_y) \in \mathbb{R}^2$ 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 $P_{\mathbf{x}}$.
G_σ	Digital Gaussian convolution of standard deviation σ.
D_s	Gaussian subsampling operator by a factor s, $(D_s\mathbf{u})(\mathbf{x}) = G_{\sigma\sqrt{4^s-1}}\mathbf{u}(2^s\mathbf{x})$, $s \geq 1$.
U_s	Digital bicubic interpolator by a factor s.

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

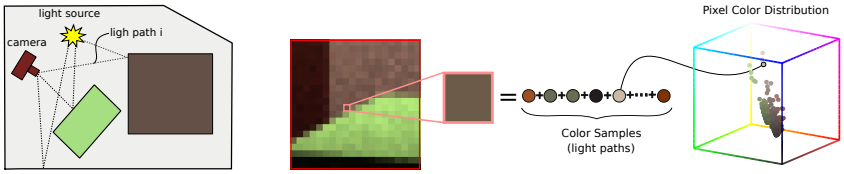

Figure 1: Pixel color computed as *average* of values along light paths cast from the image pixel, going through the camera aperture, bouncing in the scene and reaching a light source. During rendering a lot of information is computed. In particular, the color of each ray hitting a given pixel. The color distribution of the rays cast from every pixel can be generated and used to cluster similar pixels.

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 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.

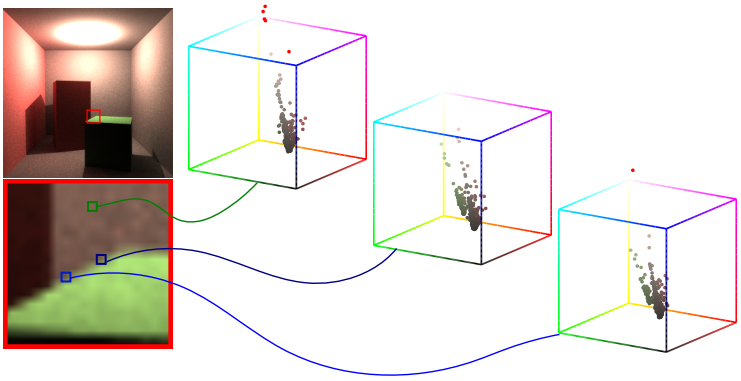

Figure 2: This figure singles out three pixels in Cornell Box scene and their color sample distributions. (The samples with color values falling out of the $[0,1]^3-$box are by convention colored in red.) The first pixel, situated on the brown wall, has a unimodal sample color distribution. The other two pixels belong to an occlusion boundary showing a bimodal green-brown distribution.

1.2 Color distribution pixel similarity

Consider the empirical distribution of the color samples at a given pixel. In Figure 2 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 color distribution can help discriminate pixels of different nature, even when their pixels color are similar.

Let us denote by $\mathcal{C}_{\mathbf{x}} = \{c_{\mathbf{x}}^1, \ldots, c_{\mathbf{x}}^{n_s(\mathbf{x})}\}$ 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 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 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) $h(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_{n_b}(\mathbf{x}))$ and $h(\mathbf{y}) = (h_1(\mathbf{y}), h_2(\mathbf{y}), \ldots, h_{n_b}(\mathbf{y}))$, 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 [9, 6] or the Kantorovich-Mallows-Monge-Wasserstein distances (also known as the Earth Mover's Distance [8]) 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 2. 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.

1.3 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

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

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, 4, 7]. 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:

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

where we denoted by $\tilde{u}(P_\mathbf{y})$ the evaluation of 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:

$$\tilde{u}(\mathbf{x}) = \frac{1}{(2w+1)^2} \sum_{|\mathbf{y}-\mathbf{x}| \leq w} V_\mathbf{y}(\mathbf{y}-\mathbf{x}). \tag{6}$$

Taking the mean as done in the preceding formula is the simplest possible aggregation method as proposed in other denoising algorithms [3, 4].

1.4 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. Indeed, long wavelength noise cannot be eliminated by this procedure, because large structures cannot be captured by small patches. Removing noise at lower frequencies thus requires a multi-scale extension of the method.

Let

$$D_s u(\mathbf{x}) := (G_{\sigma\sqrt{4^s-1}} * u)(2^s \mathbf{x}) \tag{7}$$

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 $h^s(\mathbf{x})$ 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 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 $\tilde{u}(\mathbf{x})$ and its respective pixel color distribution $h(\mathbf{x})$ the multi-scale histogram fusion proceeds as follows:

1. Generate the Gaussian multi-scale sequence: $\tilde{u}^0 = \tilde{u}$, $\tilde{u}^s = D_s\tilde{u}$, $s = 1, \dots, N$, and their respective sample color distributions h^s.

2. Apply the RHF denoising algorithm separately to each scale to recover $\bar{u}^0, \bar{u}^1, \dots, \bar{u}^N$.

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 $\hat{u}_N = \bar{u}_N$.

2 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 [5]. 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 [5] 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.

2.1 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 have 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-diminensional 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 channel. 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[1], 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

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

where n_{bins} is the number of bins, $M = 7.5$ and $s = 2$ 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: giving 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.

2.2 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

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

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 $\mathbf{s} = (s_x, s_y)$.
- Pixel $\mathbf{x}$ within the neighbor of $\mathbf{s}$, 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 s, M, n_{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 $n_s(\mathbf{x})$.

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

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

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

- $v = w_{\mathbf{s}}^{\mathbf{x}} \cdot \mathbf{c}_i$;
- **if** $v < 0$ **then** $v = 0$; *Truncate negative values*
- $v = v^{\frac{1}{\gamma}}/M$; *Compress dynamical range and renormalize*
- **if** $v > s$ **then** $v = s$; *Truncate to s*
- $\mathsf{fbin} = v \cdot (n_{\mathsf{bins}} - 2)$;
- $\mathsf{ibinL} = \texttt{floor}(\mathsf{fbin})$;

Check out of bounds

- **if** $\mathsf{ibinL} < n_{\mathsf{bins}} - 2$ **then**
  - `//inbounds`;
  - $\mathsf{wH} = \mathsf{fbin} - \mathsf{ibinL}$;
  - $\mathsf{ibL} = \mathsf{ibinL}$; *Low bin*
  - $\mathsf{wbL} = 1 - \mathsf{wH}$; *Low bin weight*
  - $\mathsf{ibH} = \mathsf{ibinL} + 1$; *High bin*
  - $\mathsf{wbH} = \mathsf{wH}$; *High bin weight*
- **else**
  - `//out of bounds,` $v >= 1$;
  - $\mathsf{wH} = (v - 1)/(s - 1)$;
  - $\mathsf{ibL} = n_{\mathsf{bins}} - 2$; *Low bin*
  - $\mathsf{wbL} = 1 - \mathsf{wH}$; *Low bin weight*
  - $\mathsf{ibH} = n_{\mathsf{bins}} - 1$; *High bin*
  - $\mathsf{wbH} = \mathsf{wH}$; *High bin weight*
- $h_i(\mathbf{x})[\mathsf{ibL}]$ += wbL;
- $h_i(\mathbf{x})[\mathsf{ibH}]$ += 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*[8] *Other popular interpolation filters are the Gaussian filter or the Mitchell filter (see [5, Chapter 7]).*

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. 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.

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

Algorithm 2: Single-Scale Ray Histogram Fusion

input : MC image $\tilde{u}$, corresponding histograms h (computed by Algorithm 1), patch size w, 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 $\quad P_{\mathbf{x}} \leftarrow$ patch centered at pixel $\mathbf{x}$;
5 $\quad W_{\mathbf{x}} \leftarrow$ search window with size b for pixel $\mathbf{x}$;
6 $\quad c \leftarrow 0$ and $V \leftarrow 0$;
7 $\quad$ **for** *every* $\mathbf{y} \in W_{\mathbf{x}}$ **do**
8 $\quad\quad Q_{\mathbf{y}} \leftarrow$ patch centered at pixel $\mathbf{y}$;
9 $\quad\quad d_{\mathbf{y}} \leftarrow \texttt{ChiSquareDistance}(h(P_{\mathbf{x}}), h(Q_{\mathbf{y}}))$; *Given by* (4)
10 $\quad S_{\mathbf{x}} \leftarrow \texttt{compute\_knn}(k_{NN}, \{d_{\mathbf{y}}\})$; $S_{\mathbf{x}}$: *set of indices of* k_{NN} *most similar patches (smallest* $d_{\mathbf{y}}$ *values)*

Llines $11-18$ *implement Equation* (5)
11 $\quad$ **for** *every pixel* $\mathbf{y} \in S_{\mathbf{x}}$ **do**
12 $\quad\quad V \leftarrow V + \tilde{u}(Q_{\mathbf{y}})$;
13 $\quad\quad c \leftarrow c + 1$;
14 $\quad$ **for** *every* $\mathbf{y} \in W_{\mathbf{x}} \cap S_{\mathbf{x}}^{c}$ **do**
15 $\quad\quad$ **if** $d_{\mathbf{y}} < \kappa$ **then**
16 $\quad\quad\quad V \leftarrow V + \tilde{u}(Q_{\mathbf{y}})$;
17 $\quad\quad\quad c \leftarrow c + 1$;
18 $\quad V \leftarrow V/c$;

Aggregation given by Equation (6)
19 $\quad n(P_{\mathbf{x}}) \leftarrow n(P_{\mathbf{x}}) + 1$; *+1 estimator for each pixel in* $P_{\mathbf{x}}$
20 $\quad \bar{u}(P_{\mathbf{x}}) \leftarrow \bar{u}(P_{\mathbf{x}}) + \big(V - \bar{u}(P_{\mathbf{x}})\big)./n(P_{\mathbf{x}})$;

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.*

Algorithm 3: Ray Histogram Fusion

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

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

$s \leftarrow n_s - 1$ $n_T \leftarrow \sum_{\mathbf{x},k} h(\mathbf{x})[k]$ *total number of samples*
while $s \geq 0$ **do**
  $u^s \leftarrow D_s(\tilde{u})$;
  $h^s \leftarrow D_s(h), \quad n_T^s \leftarrow \sum_{\mathbf{x},k} h_k^s(\mathbf{x}), \quad h^s \leftarrow \frac{n_T}{n_T^s} h^s$;
  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 $s < n_s - 1$ **then**
    $\bar{u}_s \leftarrow \bar{u}_s - U_1 D_1 \bar{u}_s + U_1 \bar{u}_{\text{old}}$
  $\bar{u}_{\text{old}} \leftarrow \bar{u}_s$ $s \leftarrow s - 1$

References

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

[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] 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.

[5] M. Pharr and G. Humphreys, *Physically Based Rendering, Second Edition: From Theory To Implementation*, Morgan Kaufmann, 2010.

[6] 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.

[7] F. Rousselle, C. Knaus, and M. Zwicker, *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.

[8] 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.

[9] 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.

[10] E. Veach, *Robust Monte Carlo methods for light transport simulation*, PhD thesis, Stanford University, Stanford, CA, USA, 1997.