Eurographics Symposium on Rendering 2017
P. Sander and M. Zwicker
(Guest Editors)

Volume 36 (2017), Number 4

Bayesian Collaborative Denoising for Monte Carlo Rendering

Malik Boughida   Tamy Boubekeur

LTCI, Telecom ParisTech, Paris-Saclay University

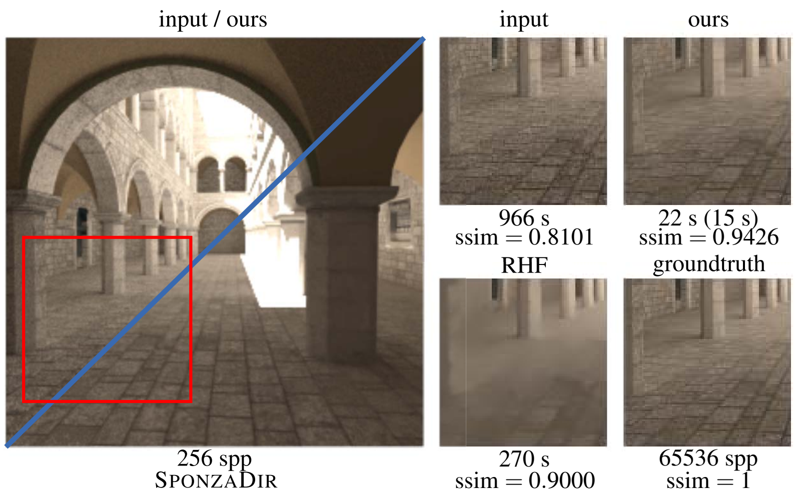

Figure 1: *A scene path-traced at 256 samples per pixel (spp), and denoised with our method. Compared to Ray Histogram Fusion (RHF), our bayesian filtering significantly improves the denoising quality – as shown by the structural similarity (ssim) measures – especially in dark areas, and accelerates by one order of magnitude the computations thanks to its collaborative nature. GPU timing between brackets.*

Abstract

The stochastic nature of Monte Carlo rendering algorithms inherently produces noisy images. Essentially, three approaches have been developed to solve this issue: improving the ray-tracing strategies to reduce pixel variance, providing adaptive sampling by increasing the number of rays in regions needing so, and filtering the noisy image as a post-process. Although the algorithms from the latter category introduce bias, they remain highly attractive as they quickly improve the visual quality of the images, are compatible with all sorts of rendering effects, have a low computational cost and, for some of them, avoid deep modifications of the rendering engine. In this paper, we build upon recent advances in both non-local and collaborative filtering methods to propose a new efficient denoising operator for Monte Carlo rendering. Starting from the local statistics which emanate from the pixels sample distribution, we enrich the image with local covariance measures and introduce a non-local bayesian filter which is specifically designed to address the noise stemming from Monte Carlo rendering. The resulting algorithm only requires the rendering engine to provide for each pixel a histogram and a covariance matrix of its color samples. Compared to state-of-the-art sample-based methods, we obtain improved denoising results, especially in dark areas, with a large increase in speed and more robustness with respect to the main parameter of the algorithm. We provide a detailed mathematical exposition of our bayesian approach, discuss extensions to multiscale execution, adaptive sampling and animated scenes, and experimentally validate it on a collection of scenes.

CCS Concepts

•***Computing methodologies*** $\rightarrow$ ***Ray tracing; Image processing;***

1. Introduction

Synthesizing realistic images from a virtual scene model means simulating the light transport from the light sources, through this scene, to the sensor (e.g., a camera). The physics laws that describe how light interacts with and bounces on materials are now quite well known, and have been modeled especially through the *rendering equation* [Kaj86]. But translating these laws into a proper simulation to achieve realistic rendering implies solving this equation numerically. Unfortunately, this is a very difficult task, since this involves computing a huge amount of integrals of unknown functions. Consequently, rendering algorithms build upon approximation models which trade accuracy for efficient computation.

Monte Carlo ray-tracing provides an elegant and generic solution to this approximation problem. The main idea is to randomly sample the integrand of the rendering equation by tracing random light paths in the scene between the camera and the light sources. From each path, a color response, called a *sample*, is computed. The final color of a pixel is defined as the average of all the samples that correspond to paths going through that pixel. Thus, the color response we get for one pixel is a random variable, or more precisely a statistical estimator of the true value of the pixel. For most of the Monte Carlo rendering algorithms, its expectation is actually the groundtruth value: the result is *unbiased*. Therefore, although the algorithm generates a random, noisy rendering, if we increase the number of samples per pixel, we are guaranteed to converge to the true solution of the rendering equation, taking into account all lighting effects without any algorithmic trick.

Since converging to the correct solution can be extremely slow, a great amount of research has been devoted to accelerate the convergence, either using importance sampling, or changing the way light paths are generated, leading to algorithms such as *bidirectional path-tracing* [LW93], *photon mapping* [Jen96], *Metropolis Light Transport* [VG97] or *Vertex Connection and Merging* [GKDS12]. However, even if a relatively fast convergence is achieved for a large part of the image, some difficult zones (such as caustics) often remain noisy and need much more time to converge.

Beside accelerating convergence, one can also *filter* the resulting image, as a post-processing step. This allows to get a fast nice-looking result, at the cost of adding some bias, which is still fair for a number of visual applications. Thus, numerous methods have been designed to improve synthetic image filtering (see Section 2) by using two alternative strategies: while some methods are *feature-based* and exploit the typical rich per-pixel extra information (e.g., normals from the G-Buffer), some others are purely *sample-based* and rely solely on the radiance distribution ending up in each pixel to perform the filtering. Among these methods, the *Ray Histogram Fusion* [DMB*14] (or RHF) adopts a very generic approach, agnostic to the particulars of the scene and simulated effects (see Section 2.2). Concurrrently, in image processing, a collaborative filter named *Non-Local Bayes* [LBM13] (or NL-Bayes) has recently emerged as one of the state-of-the-art image denoising algorithms.

In this paper, we propose a new sample-based denoising algorithm for synthetic images which combines the benefits of both RHF and NL-Bayes (Section 4). In particular, we design a specific Bayesian framework which accounts for local statistics extracted from the per-pixel sample distribution (Section 3), introduce a specific multiscale execution algorithm (Section 5), propose an adaptive sampling scheme to cope with the non uniform convergence rate across the image by triggering more samples in noisier areas (Section 6) and extend our method to animated sequences (Section 7). Last, we evaluate our approach and compare it to RHF and other methods on a number of examples (Section 8) before discussing its limitations (Section 9).

By improving significantly the denoising quality compared to RHF, we aim at demonstrating that we can achieve a very high denoising power while relying only on sample statistics, without using extra information such as the G-Buffer. We do get particularly interesting results for scenes where the G-Buffer is not informative, like highly specular ones. Indeed, as shown recently by Bitterli et al. [BRM*16], both feature-based and sample-based methods have their own strengths depending on the underlying nature of the region to be denoised and can be combined to achieve the best practical denoising result. We think that the quality, robustness, generality, speed, ease of implementation and lack of parameter tweaking of our method may find its use in a wide spectrum of application scenarios, including production, where numerous per-pixel effects make the G-Buffer often irrelevant. From a more theoretical point of view, we believe that our introduction of the Bayesian framework into the context of rendering denoising may breed new ideas and algorithms in the field.

Contributions. The main contributions of our approach are threefold. First, we propose a bayesian model for per-pixel noise in Monte Carlo Rendering. Second, we develop a collaborative filter that exploits this model for all-effects rendering and with a low degree of invasiveness in the engine. Third, we extend our denoiser to multi-scale filtering, adaptive sampling and animated scenes. In particular, the dichotomous approach that we use to robustly address sampling constraints can be generalized to any other kinds of adaptive sampling scheme. Although our algorithm may appear at first sight as a combination of NL-Bayes and RHF, we will see that NL-Bayes is fundamentally not usable on Monte-Carlo rendering, and that we do need the novel mathematical derivations at the heart of our method to make it work.

2. Related work

2.1. Image denoising

The key idea of a good filtering for denoising purpose is to update the value of a pixel using only pixels of the same "nature", i.e., with a close groundtruth value. Otherwise, the filter tends to overblur the image. However, being too strict when gathering pixels of the same nature leads to poor denoising. Consequently, a tradeoff is usually made using a weighting scheme or a filtering threshold.

Bilateral filters [TM98] compute weights of neighbor pixels by comparing both their locations and their values. Going a step further, *cross* (or *joint*) bilateral filters [ED04, PSA*04] integrate more dimensions in the weighting kernels to account for additional pixel information (e.g., depth).

To further characterize pixels, the use of patches instead of pointwise values has been proposed to improve the denoising quality and robustness. One of the first patch-based denoising algorithm was the *Non-Local Means* algorithm [BCM05] (or NL-means): in a nutshell, this is a bilateral filter which uses patches instead of pixels and removes the spatial component in the weights (hence the name "non-local"). Since NL-means, various denoising algorithms have been proposed, beyond simple weighted average of neighbors, such as *BM3D* [DFKE06] and all its many variants. The recent Non-Local Bayes method [LBM13] (or NL-Bayes) is known to be currently among the best ones, which motivates us to build our method upon it.

NL-Bayes uses the Maximum A Posteriori (MAP) Estimator to denoise patches in the image. To compute this estimator, it relies on a bayesian approach, which requires a prior on the local distribution of patches. In practice, NL-Bayes locally builds this prior model using a selection of similar neighbor patches. Finally, not only the main patch but also all its selected neighbor ones are denoised at the same time: consequently, NL-Bayes is a *collaborative* filter.

The interested reader may refer to the work of Lebrun et al. [LCBM12] for a review of other denoising algorithms.

2.2. Filtering Monte Carlo rendering

In the following, we focus on methods which, just like our approach, work as post-processing steps, requiring relatively few changes in the rendering algorithm.

Post-process filtering is often followed by an error analysis that is used to determine a required per-pixel number of samples for the next step of a progressive rendering [RKZ11], with the purpose of minimizing the relative Mean Square Error. Iterating non-uniform rendering, filtering and analysis is called *Adaptive sampling and reconstruction*, a method later enriched to use the non-local means filter [RKZ12]. Recently, Moon et al. [MMMG16] also used this process, but with a more robust error estimator to drive the adaptive rendering, and local polynomial models to perform reconstruction.

When filtering Monte Carlo rendering, one can rely on the additional information (or *features*) the rendering engine can provide beyond just a single color value. For instance, while cross bilateral filters have been used in Monte Carlo rendering for a decade [XP05], Rousselle et al. [RMZ13] used not only the G-buffer but also per-pixel information about caustics and shadows to apply a cross-bilateral filter on complex Monte Carlo renderings.

However, these additional per-pixel features may be very noisy as well, especially when dealing with effects such as motion blur or depth of field. Consequently, Rousselle et al. [RMZ13] apply a pre-filtering on these additional inputs before using them. As for Moon et al. [MCY14], their weighted regression scheme is specifically designed to robustly address noisy high-dimensional features. Later, the same authors used a simpler truncated SVD-based pre-filtering on their features [MIGYM15], focusing mainly on efficiency as they target interactive GPU-accelerated ray-tracing engines. Indeed, their strategy to share local linear models across several pixels is at the essence of their computational speed, and can be seen as a form of collaborative denoising.

Alternatively, other filtering methods rely purely on the per-pixel color response distribution (or *samples*), such as the Ray Histogram Fusion [DMB*14] (or RHF) which does not use the G-Buffer at all: instead, it makes use of per-pixel histograms of samples and applies a form of “joint” non-local means filter, using the histograms as the joint data. Moreover, a multiscale scheme is proposed to remove low-frequency noise: a pyramid of the input buffers is generated, and the algorithm is applied at each level.

Lastly, several methods were proposed to choose locally the filter to use or its parameters. Sen and Darabi [SD12] estimate per-pixel bandwidth for their cross-bilateral filtering, while Kalantari et al. [KS13] take the filtering algorithm itself as additional input and ajust locally its parameters to achieve better results, adding a multi-scale dimension to their filtering process. Rousselle et al. [RMZ13] used the Stein’s Unbiased Risk Estimator – or SURE, introduced for denoising Monte Carlo rendering by Li et al. [LWC12] – to combine various filtering schemes with different parameters, and used a pair of half buffers (i.e., that both contain only half of the samples) to estimate the per-pixel variance used to guide an adaptive sampling strategy. Bauszatz et al. [BEEM15] performed a local filter selection based on graph cuts. As for Kalantari et al. [KBS15], they rely on machine learning techniques to estimate local filtering parameters: a long pre-process is performed to train a neural network, but once it is done, they are able to produce fast and impressive denoising, even with a very low number of samples per pixel. Last, Bitterli et al. [BRM*16] employ a first-order model that exploits per-pixel features when relevant and gracefully fade to sample-based filtering when no correlation can be established between the auxiliary features and the pixel color.

We refer the reader to the recent survey by Zwicker et al. [ZJL*15] for more details, references and a wider view on the topic of Monte Carlo rendering filtering.

3. Adaptation of Bayesian Collaborative Denoising

3.1. Context and notations

We use the following notations:

- (r_w, r_h) resolution (width/height) of all the buffers (in pixels),
- $p_i = (u_i, v_i) \in I = [0; r_w - 1] \times [0; r_h - 1]$ a pixel (ith pixel),
- $s_i^k = ((s_i^k)^R, (s_i^k)^G, (s_i^k)^B)^T$ kth color sample of pixel p_i,
- n_i number of samples of pixel p_i,
- h_i histogram of the samples of pixel p_i,
- $\tilde{x}_i = \frac{1}{n_i} \sum_{k=1}^{n_i} s_i^k$ renderer’s noisy output color for pixel p_i,
- $\ddot{\mathbf{c}}_i = \frac{1}{n_i - 1} \sum_{k=1}^{n_i} (s_i^k - \tilde{x}_i)(s_i^k - \tilde{x}_i)^T$ 3×3 empirical covariance matrix of the samples of pixel p_i,
- P_i $p \times p$ patch around p_i (set of pixels),
- $\mathcal{J}_i$ $q \times q$ search window around p_i (set of pixels).

Capital letters are used to denote the patch version of each quantity: for instance, $\tilde{X}_i$ denotes the vector of dimension $3p^2$ that concatenates the color values $\tilde{x}_j$ of the patch P_i.

A standard Monte-Carlo ray-tracer produces the $\tilde{x}$ buffer. To be able to denoise this buffer with our method, we require the renderer to additionally output the h and $\ddot{\mathbf{c}}$ buffers. We use the h buffer to measure similarity between patches while the $\ddot{\mathbf{c}}$ buffer gives us an estimate of the local pixel covariance which is a major component of our bayesian denoising scheme. Alternatively, our implementation can also extract h and $\ddot{\mathbf{c}}$ from *full sampling* images (i.e., record of all per-pixel individual samples) before proceeding.

3.2. Bayesian denoising framework

The denoising problem can be formulated as follows:

$$\tilde{X}_i = \mathring{X}_i + \nu_i$$

where $\tilde{X}_i$ is the observed noisy value, $\mathring{X}_i$ is the unknown groundtruth value that we want to recover and ν_i is a random noise. Our goal is to build an estimator $\hat{X}_i$ of $\mathring{X}_i$, using the observed $\tilde{X}_i$

and our knowledge about the noise. A good way to estimate $\mathring{X}_i$ is to choose the most probable one, given $\tilde{X}_i$ (MAP estimator):

$$\hat{X}_i = \underset{X}{\operatorname{argmax}}\, \mathbb{P}(X \,|\, \tilde{X}_i)$$

which gives us, by the Bayes rule:

$$\hat{X}_i = \underset{X}{\operatorname{argmax}} \frac{\mathbb{P}(\tilde{X}_i \,|\, X)\mathbb{P}(X)}{\mathbb{P}(\tilde{X}_i)} = \underset{X}{\operatorname{argmax}}\, \mathbb{P}(\tilde{X}_i \,|\, X)\mathbb{P}(X)$$

$\mathbb{P}(\tilde{X}_i \,|\, X)$ is the noise model (see section 3.3) while $\mathbb{P}(X)$is a *prior* on the probability distribution of the groundtruth (see section 3.4).

Algorithm 1 Overview of our algorithm

```
Require: input buffers: color $\tilde{x}$, histogram $h$, pixel covariance $\mathbf{c}$
Require: parameters: patch size $p$, search window size $q$, histogram distance threshold $d_{\max}$
  function BAYESIANCOLLABORATIVEDENOISING($\tilde{x}$, $h$, $\mathbf{c}$)
    Unmark pixels and initialize to zero buffers $\hat{x}$ and $m$
    for all $p_i \in I$ do
      if ISMARKED($p_i$) then
        continue
      $\mathcal{K} \leftarrow$ SELECTSIMILARPATCHCENTERS($h$, $p_i$)
      MARKPIXELS($\mathcal{K}$)
      $\tilde{\mathcal{X}} \leftarrow \{\tilde{X}_k, p_k \in \mathcal{K}\}, \mathcal{C} \leftarrow \{\mathbf{C}_k, p_k \in \mathcal{K}\}$
      $\hat{\mathcal{X}} \leftarrow$ DENOISEPATCHES($\tilde{\mathcal{X}}$, $\mathcal{C}$)          ▷ (see Algorithm 2)
      AGGREGATE($\hat{x}$, $m$, $\hat{\mathcal{X}}$)
    FINALAGGREGATION($\hat{x}$, $m$)
    return $\hat{x}$
  function SELECTSIMILARPATCHCENTERS($h$, $p_i$)
    $\mathcal{K} \leftarrow \emptyset$
    for all $p_j \in \mathcal{J}_i$ do
      if $\mathcal{D}(H_i, H_j) < d_{max}$ then
        $\mathcal{K} \leftarrow \mathcal{K} \cup \{p_j\}$
    return $\mathcal{K}$
  function AGGREGATE($\hat{x}$, $m$, $\hat{\mathcal{X}}$)
    for all $p_k \in \mathcal{K}$ do
      for all $p_l \in P_k$ do
        $m_l \leftarrow m_l + 1$
        $\hat{x}_l \leftarrow \hat{x}_l + \hat{\mathcal{X}}_k[p_l]$          ▷ part of patch $\hat{\mathcal{X}}_k$ that corresponds to pixel $p_l$
  function FINALAGGREGATION($\hat{x}$, $m$)
    for all $p_i \in I$ do
      $\hat{x}_i \leftarrow \hat{x}_i / m_i$
```

3.3. Noise model

While the NL-Bayes algorithm is meant to denoise an image with a uniform (i.e., the same for all pixels) isotropic (in RGB space) gaussian noise, to address Monte Carlo denoising, we must first understand its own noise model.

Let's focus on one pixel p_i, and assume that all the samples s_i^k are independent realizations of the same random variable s_i (independent identically distributed, or i.i.d.). The probability distribution of s_i is potentially extremely complex e.g., it can be multi-modal (if the pixel covers a non-homogeneous area for instance), or it can output mainly very dark values and suddenly a very bright one (for a caustic in standard path-tracing for example). In other words, this distribution is far from being a gaussian. Still, we know that its expectation is the groundtruth value we want to estimate ($\mathbb{E}[s_i] = \mathring{x}_i$). And, indeed, we are not interested in the distribution of s_i, but in the one of $\tilde{x}_i$, the average of the samples. Its expectation is the same:

$$\mathbb{E}[\tilde{x}_i] = \mathbb{E}\left[\frac{1}{n_i}\sum_{k=1}^{n_i} s_i^k\right] = \frac{1}{n_i}\sum_{k=1}^{n_i}\mathbb{E}[s_i^k] = \frac{1}{n_i}\sum_{k=1}^{n_i}\mathring{x}_i = \mathring{x}_i$$

Moreover, the independence of the samples induces that the covariance of the sum is the sum of the covariances:

$$\mathbf{Cov}[\tilde{x}_i] = \frac{1}{n_i^2}\mathbf{Cov}\left[\sum_{k=1}^{n_i} s_i^k\right] = \frac{1}{n_i^2}\sum_{k=1}^{n_i}\mathbf{Cov}[s_i^k] = \frac{1}{n_i}\mathbf{Cov}[s_i]$$

Thus, even if we have only one observation of the random variable $\tilde{x}_i$, we can still estimate its covariance matrix by estimating the one of s_i. That is why we require the rendering engine to output the empirical covariance matrix of the samples $\ddot{\mathbf{c}}_i$: if we simply divide it by n_i, we get an estimate of the covariance of $\tilde{x}_i$, i.e., an estimation of the amount of noise in the form of a matrix, without any isotropic assumption.

Interestingly, the central limit theorem also tells us that when the number of samples n_i increases, the probability distribution of $\tilde{x}_i$ converges to a gaussian: hence, we can argue that even if s_i has a very complex distribution, $\tilde{x}_i$ can still be seen as a gaussian noise around the groundtruth value, provided we have a sufficient number of samples.

From now on, we assume that for each pixel p_i, we have an anisotropic gaussian noise of covariance matrix $\mathbf{c}_i = \frac{1}{n_i}\ddot{\mathbf{c}}_i$. In Algorithm 1 (see below), we directly take $\mathbf{c}$ as input buffer instead of $\ddot{\mathbf{c}}$. To deal with patches, we assume that the noise of a pixel is not correlated to the noise of another pixel of the patch. Notice that this assumption can be perfectly valid for two pixels that have the same groundtruth values; on the other hand, sharing samples across pixels (as some renderers may do) introduce correlation between these. So, with $P_i = \{p_{i_1}, p_{i_2}, \ldots p_{i_{p^2}}\}$, the covariance of the patch $\tilde{X}_i$ is the $3p^2 \times 3p^2$ block-diagonal matrix:

$$\mathbf{C}_i = \begin{bmatrix} \mathbf{c}_{i_1} & 0 & \ldots & 0 \\ 0 & \mathbf{c}_{i_2} & \ldots & 0 \\ & & \ddots & \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \ldots & \mathbf{c}_{i_{p^2}} \end{bmatrix} = \mathbf{Diag}\left(\mathbf{c}_j\right)_{p_j \in P_i}$$

3.4. Local prior model

$\mathbb{P}(X)$ is a prior on the distribution of the groundtruth, locally around $\tilde{X}_i$ in the space of patch color values. By selecting neighbor patches, we get a point cloud in this space, from which we can compute the empirical mean $\overline{\tilde{X}}$ and covariance matrix $\tilde{\mathfrak{S}}$: these are the parameters of a gaussian model of the distribution of noisy values.

Then, we can infer an approximation of the distribution of groundtruth values. While in NL-Bayes, all pixels have the same isotropic gaussian noise of variance σ^2 – and one can keep the same mean and substract $\sigma^2 I$ to the covariance matrix– for Monte Carlo rendering, we propose to use the average $\overline{\mathbf{C}}$ of the covariance matrices $\mathbf{C}_j$ of all patches instead of $\sigma^2 I$ (see Appendix A for the mathematical justification).

4. Algorithm

4.1. Overview

Our method is illustrated in Figure 2 and listed in Algorithm 1. At initialization, we unmark all pixels. Then, for each pixel p_i, if not *marked*, we compare the patch of histograms H_i with the neighbor patches of histograms H_j in the $q \times q$ neighborhood ($p_j \in \mathcal{J}_i$), using

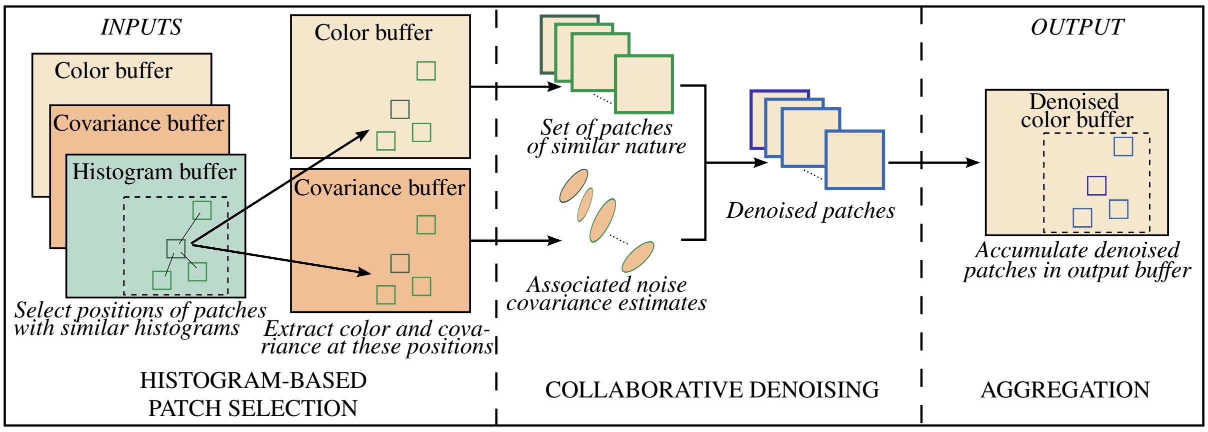
INPUTS
Color buffer
Covariance buffer
Histogram buffer
Color buffer
Covariance buffer
Select positions of patches with similar histograms
Extract color and covariance at these positions
HISTOGRAM-BASED PATCH SELECTION
Set of patches of similar nature
Associated noise covariance estimates
Denoised patches
COLLABORATIVE DENOISING
OUTPUT
Denoised color buffer
Accumulate denoised patches in output buffer
AGGREGATION

Figure 2: ***Overview***. *Starting from our 3 input buffers, the histogram of the current patch (central dark green one) is compared to the ones of neighboring patches to select patches of similar nature (light green); then, we extract the noisy color and the covariance values and use them to compute denoised versions of the patches (purple), that we aggregate in an output buffer (blue).*

a distance that we denote $\mathcal{D}$. Following Delbracio et al. [DMB*14] we use the χ^2 distance (see Appendix B for the formula).

The resulting patch distances are compared to a threshold, which is the main parameter of our method. It controls the tradeoff between denoising and overblur. Thus we have selected a set of patches $\mathcal{K}$ (including p_i itself) of similar nature; from now on, our goal is to filter them all together, *collaboratively*.

If we have a sufficient number of selected neighbors $|\mathcal{K}|$, we can perform the actual Bayesian collaborative denoising computations, (see Section 4.2). The resulting denoised patches $\{\hat{X}_k\}$ are then accumulated in an $\hat{x}$ buffer, and their center pixels are *marked*. As a color patch $\tilde{X}_k$ has dimension $3p^2$, we need $|\mathcal{K}| >= 3p^2$ to be able to inverse the empirical covariance matrix $\tilde{\mathfrak{S}}$ of these patches. Otherwise, we just average the color patches and use the result as the filtering output for the patch P_i only: $\hat{X}_i = \frac{1}{|\mathcal{K}|}\sum_{p_k\in\mathcal{K}}\tilde{X}_k$.

In an additional buffer m, we count the number of estimates received by each pixel, so that we can compute the average in the end (see Algorithm 1).

4.2. First step

As stated in Section 2.1, the bayesian formulation of the denoising problem can be written as:

$$\hat{X}_i = \underset{X}{\operatorname{argmax}}\, \mathbb{P}(\tilde{X}_i\,|\,X)\mathbb{P}(X)$$

where the first factor is the non-uniform anisotropic gaussian noise model (see Section 3.3):

$$\mathbb{P}(\tilde{X}_i\,|\,X) = \alpha \exp\left(-\frac{1}{2}(X-\tilde{X}_i)^T \mathbf{C}_i^{-1}(X-\tilde{X}_i)\right)$$

and the second one a prior on the distribution of the groundtruth, estimated from the neighbor patches (see Section 3.4):

$$\mathbb{P}(X) = \beta \exp\left(-\frac{1}{2}(X-\overline{\tilde{X}})^T(\tilde{\mathfrak{S}}-\overline{\mathbf{C}})^{-1}(X-\overline{\tilde{X}})\right)$$

Solving this optimization problem yields (see Appendix C for the derivations):

$$\hat{X}_i = \tilde{X}_i - \mathbf{C}_i\tilde{\mathfrak{S}}_i'^{-1}(\tilde{X}_i - \overline{\tilde{X}}) \qquad (1)$$

where $\tilde{\mathfrak{S}}_i' = \tilde{\mathfrak{S}} - \overline{\mathbf{C}} + \mathbf{C}_i$

Similarly to Rousselle et al. [RMZ13], we filter the noisy measure $\mathbf{C}_i$ of the covariance of the noise by setting $\mathbf{C}_i = \overline{\mathbf{C}}$. As we are considering patches of same nature, we can argue that they should have similar sample distributions and then close covariance matrices (provided they have a close number of samples). This assumption simplifies Formula 1, we get $\tilde{\mathfrak{S}}_i' = \tilde{\mathfrak{S}}$ and thus:

$$\boxed{\hat{X}_i = \tilde{X}_i - \overline{\mathbf{C}}\tilde{\mathfrak{S}}^{-1}(\tilde{X}_i - \overline{\tilde{X}})} \qquad (2)$$

Thanks to this simplification, we can compute the matrix inversion and multiplication $\overline{\mathbf{C}}\tilde{\mathfrak{S}}^{-1}$ once for all the neighbor patches. Figure 3 illustrates this collaborative filtering process.

4.3. Eigen values clamping

During the first step, given the empirical covariance matrix $\tilde{\mathfrak{S}}$ of the noisy patch cloud, we estimated the covariance matrix of the noiseless patch cloud by subtracting the mean noise covariance matrix $\overline{\mathbf{C}}$. This difference of matrices is symmetric as both terms are symmetric. However, we have no guarantee that it is positive. Yet, it represents the covariance matrix parameter of a gaussian distribution and therefore should be positive . Unfortunately, we experimentally found out that getting a non-positive matrix actually happens and causes artifacts. Indeed, with our complex case of non-uniform anisotropic noise, it is very likely that we get negative eigen values, even in some inhomogeneous areas. We solve this problem by clamping the negative eigen values to 0. As $\tilde{\mathfrak{S}} - \overline{\mathbf{C}}$ is symmetric, it can be diagonalized: $\tilde{\mathfrak{S}} - \overline{\mathbf{C}} = \mathbf{V}\mathbf{D}\mathbf{V}^T$, where $\mathbf{D}$ is the diagonal matrix with all the eigen values, and each column of $\mathbf{V}$ is a corresponding eigen vector. Then we can define the "clamped" matrix:

$$\tilde{\mathfrak{S}}_+ = (\tilde{\mathfrak{S}} - \overline{\mathbf{C}})_+ + \overline{\mathbf{C}} = \mathbf{V}\mathbf{D}_+\mathbf{V}^T + \overline{\mathbf{C}}$$

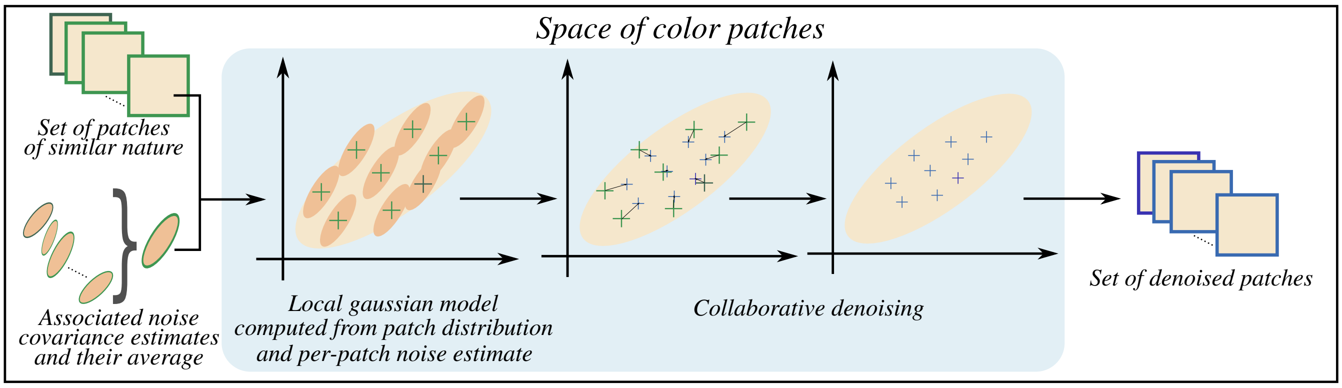
Space of color patches
Set of patches of similar nature
Associated noise covariance estimates and their average
Local gaussian model computed from patch distribution and per-patch noise estimate
Collaborative denoising
Set of denoised patches

Figure 3: *First step of the collaborative denoising. From noisy color and covariance values, we compute a local gaussian model (big ellipse). The product of this gaussian (big ellipse) with the one that represents the covariance of a patch (little ellipse) has a maximum (blue cross), which is the denoised version of that patch.*

Algorithm 2 Denoising computations in our algorithm

```
Require: $\tilde{\mathcal{X}} = \{\tilde{X}_k, p_k \in \mathcal{K}\}$
Require: $\mathcal{C} = \{\mathbf{C}_k, p_k \in \mathcal{K}\}$
  function DENOISEPATCHES($\tilde{\mathcal{X}}, \mathcal{C}$)
      $\hat{\mathcal{X}}^{\text{step1}} \leftarrow$ DENOISESTEP1($\tilde{\mathcal{X}}, \mathcal{C}$)
      $\hat{\mathcal{X}} \leftarrow$ DENOISESTEP2($\tilde{\mathcal{X}}, \mathcal{C}, \hat{\mathcal{X}}^{\text{step1}}$)
      return $\hat{\mathcal{X}}$
  function EMPIRICALMEAN($\mathcal{Y}$)
      return $\frac{1}{|\mathcal{Y}|} \sum_{\mathcal{Y}_k \in \mathcal{Y}} \mathcal{Y}_k$
  function EMPIRICALCOVARIANCEMATRIX($\mathcal{Y}$)
      $\overline{\mathcal{Y}} \leftarrow$ EMPIRICALMEAN($\mathcal{Y}$)
      return $\frac{1}{|\mathcal{Y}| \quad 1} \sum_{\mathcal{Y}_k \in \mathcal{Y}} (\mathcal{Y}_k \quad \overline{\mathcal{Y}})(\mathcal{Y}_k \quad \overline{\mathcal{Y}})^T$
  function DENOISESTEP1($\tilde{\mathcal{X}}, \mathcal{C}$)
      $\overline{\mathbf{C}} \leftarrow$ EMPIRICALMEAN($\mathcal{C}$)
      $\overline{\tilde{X}} \leftarrow$ EMPIRICALMEAN($\tilde{\mathcal{X}}$)
      $\tilde{\mathfrak{S}} \leftarrow$ EMPIRICALCOVARIANCEMATRIX($\tilde{\mathcal{X}}$)
      $\tilde{\mathfrak{S}}_+ \leftarrow \overline{\mathbf{C}}+$ CLAMPNEGATIVEEIGENVALUES($\tilde{\mathfrak{S}} \quad \overline{\mathbf{C}}$)
      for all $\tilde{X}_k \in \tilde{\mathcal{X}}$ do
          $\hat{\mathcal{X}}_k^{\text{step1}} \leftarrow \tilde{X}_k \quad \overline{\mathbf{C}}\tilde{\mathfrak{S}}_+^{\ 1}(\tilde{X}_k \quad \overline{\tilde{X}})$
      return $\hat{\mathcal{X}}^{\text{step1}}$
  function DENOISESTEP2($\tilde{\mathcal{X}}, \mathcal{C}, \hat{\mathcal{X}}^{\text{step1}}$)
      $\overline{\mathbf{C}} \leftarrow$ EMPIRICALMEAN($\mathcal{C}$)
      $\overline{\hat{X}} \leftarrow$ EMPIRICALMEAN($\hat{\mathcal{X}}^{\text{step1}}$)
      $\hat{\mathfrak{S}}^{\text{step1}} \leftarrow$ EMPIRICALCOVARIANCEMATRIX($\hat{\mathcal{X}}^{\text{step1}}$)
      for all $\tilde{X}_k \in \tilde{\mathcal{X}}$ do
          $\hat{\mathcal{X}}_k \leftarrow \tilde{X}_k \quad \overline{\mathbf{C}}(\hat{\mathfrak{S}}^{\text{step1}} + \overline{\mathbf{C}}) \quad ^1(\tilde{X}_k \quad \overline{\hat{X}})$
      return $\hat{\mathcal{X}}$
```

where $\mathbf{D}_+$ is the clamped diagonal matrix: $\forall l, (\mathbf{D}_+)_{l,l} = \max(0, (\mathbf{D})_{l,l})$. Thus, in our practical implementation, we replace $\tilde{\mathfrak{S}}$ by $\tilde{\mathfrak{S}}_+$ in Equation 2.

4.4. Second step

After the first step, we have denoised the set of selected noisy patches $\{\tilde{X}_j\}$ into the set of patches $\{\hat{X}_j^{\text{step1}}\}$ where $\hat{X}_j^{\text{step1}} = \tilde{X}_j \quad \overline{\mathbf{C}}\tilde{\mathfrak{S}}_+^{\ 1}(\tilde{X}_j \quad \overline{\tilde{X}})$. We can then compute the empirical covariance matrix $\hat{\mathfrak{S}}^{\text{step1}}$ of this point cloud and use it as our model of denoised patch distribution, instead of $(\tilde{\mathfrak{S}} \quad \overline{\mathbf{C}})_+$ (which is also equal to $\tilde{\mathfrak{S}}_+ \quad \overline{\mathbf{C}}$). Besides, our prior model is now centered around $\overline{\hat{X}} = \frac{1}{|\mathcal{K}|}\sum_{p_j \in \mathcal{K}} \hat{X}_j^{\text{step1}}$ instead of $\overline{\tilde{X}}$. So in the formulas, we just have to replace $\tilde{\mathfrak{S}}_+$ by $\hat{\mathfrak{S}}^{\text{step1}} + \overline{\mathbf{C}}$, and $\overline{\tilde{X}}$ by $\overline{\hat{X}}$:

$$\boxed{\hat{X}_j^{\text{step2}} = \tilde{X}_j \quad \overline{\mathbf{C}}(\hat{\mathfrak{S}}^{\text{step1}} + \overline{\mathbf{C}}) \quad ^1(\tilde{X}_j \quad \overline{\hat{X}})}$$

As we select the neighbors exclusively from the histograms, which will not be changed after the first denoising, the role of the second step is only to improve the prior distribution. Consequently, it can be performed right after the first step, for each group of patches, without the need to generate a full first-step denoised image.

Algorithm 2 summarizes our two-step denoising method.

5. Multiscale filtering

Applied at a single scale, low-frequency noise may resist to our algorithm. Consequently, we propose a multiscale execution scheme.

We call S the number of scales. Scale 0 is the full-resolution scale while scale $S \quad 1$ is the scale with lowest resolution and scale s is a $2^s \times 2^s$ downsampling of scale 0. We call $\mathcal{U}$ the 2×2 upsampling operator (with an interpolation scheme), and $\mathcal{D}_s$ the $2^s \times 2^s$ downsampling operator (with pre-filtering to avoid aliasing). The single-scale collaborative denoising process that we have described so far will be written as the $\mathcal{F}$ operator. The set of input buffers is denoted by $I = I^{(0)}$: it includes the color values x, the histograms h and the covariance matrices $\ddot{\mathbf{c}}$.

First we take care of the lowest resolution scale $s = S \quad 1$ by computing $I^{(s)} = \mathcal{D}_s(I^{(0)})$ and its output $O^{(s)} = \mathcal{F}(I^{(s)})$. Then, for each scale from $s = S \quad 2$ to $s = 0$, we compute:

- the downsampled input buffers $I^{(s)} = \mathcal{D}_s(I^{(0)})$,
- a temporary single-scale output $O^{(s)\prime} = \mathcal{F}(I^{(s)})$,
- $O_{\text{hf}}^{(s)} = O^{(s)\prime} \quad \mathcal{U}(\mathcal{D}_1(O^{(s)\prime}))$ to retain only the high frequencies of this previous result,
- the low-frequency part $O_{\text{lf}}^{(s)} = \mathcal{U}(O^{(s+1)})$, using the result from higher scales,
- the final output for scales s to $S \quad 1$: $O^{(s)} = O_{\text{lf}}^{(s)} + O_{\text{hf}}^{(s)}$.

The final result of this multiscale process is $O^{(0)}$.

The pre-filtering and simple downsampling on $h^{(0)}$ gives only an intermediate result that we call $h^{(s)\prime}$: the histogram buffer needs an additional normalization step. By summing up all the bins of all the pixels, we get the total numbers of samples $n_{\text{tot}}^{(0)}$ and $n_{\text{tot}}^{(s)}$. Finally, we multiply $h^{(s)\prime}$ by $n_{\text{tot}}^{(0)}/n_{\text{tot}}^{(s)}$ to get the "normalized" input histogram buffer $h^{(s)}$ for scale s, that have the same total number of samples as $h^{(0)}$.

While this multiscale algorithm and the histogram normalization resemble the RHF approach, we have one additional critical stage: the correct covariance matrix buffer downsampling. The color x_i of pixel p_i in the full-resolution input buffer is a random variable, which is the result of an average of n_i i.i.d. variables s_i^1, ... $s_i^{n_i}$. As stated in Section 3.3, from the empirical covariance matrix of the samples $\ddot{\mathbf{c}}_i$, we build an estimate $\mathbf{c}_i = \frac{\ddot{\mathbf{c}}_i}{n_i}$ of $\mathbf{Cov}[\tilde{x}_i]$ (the covariance matrix of $\tilde{x}_i$). A pixel value in the pre-filtered downsampled input color buffer at scale s can be written: $\tilde{x}_i^{(s)} = \sum_j g_j \tilde{x}_j$ where the g_j are the normalized pre-filtering weights ($\sum_j g_j = 1$). When we downsample the covariance buffer, we want to get an estimate of the covariance of $\tilde{x}_i^{(s)}$. So let us compute:

$$\mathbf{Cov}[\tilde{x}_i^{(s)}] = \mathbf{Cov}\left[\sum_j g_j \tilde{x}_j\right] = \sum_j \mathbf{Cov}[g_j \tilde{x}_j] = \sum_j g_j^2 \mathbf{Cov}[\tilde{x}_j]$$

Consequently, to downsample the covariance buffer correctly, we pre-filter using the same mask as for the color buffer but with squared coefficients:

$$\mathbf{c}_i^{(s)} = \sum_j g_j^2 \mathbf{c}_j$$

6. Adaptive sampling

Monte Carlo rendering suffers from *spikes* i.e., samples with extremely high values, that completely alter the average of the samples for a given pixel. A practical solution can be to apply a spike-removal prefiltering (as described in Section 7). But, for high quality results, we do not want to consider these spikes as outliers, because they carry information and their contribution may even be essential to the final pixel value. A typical example is a caustic in standard (non-bidirectional) path tracing.

Our basic algorithm does not cope well with spikes: their presence reveals that the probability distribution of the samples s_i has an extremely high variance, with at least two components that are very far away from each other in color space, and then the distribution of the pixel color $\tilde{x}_i$ may still be far from having converged to a gaussian (central limit theorem). Such a violation of our base gaussian hypothesis breeds artifacts.

Adaptive Sampling helps dealing with this issue, generating more samples for difficult pixels, and fewer samples for those which converge quickly. In this section, we explain how we can make our algorithm and an adaptive sampling scheme collaborate and both benefit from each other: the denoising drives the sampling, and the improved sampling produces better inputs for the denoising.

More precisely, we propose an adaptive sampling process which can be described by the following steps:

1. **Uniform rendering**: perform a first rendering with a uniform number of samples per pixel.
2. **Collaborative denoising**: denoise the noisy color buffer with our filtering algorithm.
3. **Adaptive sampling map computation**: use the difference between the noisy and the denoised values along with the known per-pixel covariances to compute a map of required samples per pixel.
4. **Adaptive rendering**: use this map to compute a new adaptive rendering (with a variable number of samples per pixel).
5. **Merging**: merge the buffers produced by this rendering with the previous (noisy) ones.
6. Go back to step 2.

6.1. Merging step

Let $s_i^1, \dots s_i^{n_i^{(1)}}$ be the $n_i^{(1)}$ samples generated by a first rendering and $s_i^{n_i^{(1)}+1}, \dots s_i^{n_i^{(1)}+n_i^{(2)}}$ the ones from a second step in an adaptive rendering framework. If we call $n_i^{(r)}$, $\tilde{x}_i^{(r)}$, $h_i^{(r)}$ and $\ddot{\mathbf{c}}_i^{(r)}$ respectively the number of samples, the average value, the histogram and the covariance matrix for the pixel p_i of the rth rendering, the merged values taking into account all the samples are simply (see Appendix D for the justification of the covariance formula):

- $n_i = n_i^{(1)} + n_i^{(2)}$
- $\tilde{x}_i = \frac{1}{n_i}\left(n_i^{(1)}\tilde{x}_i^{(1)} + n_i^{(2)}\tilde{x}_i^{(2)}\right)$
- $h_i = h_i^{(1)} + h_i^{(2)}$
- $\ddot{\mathbf{c}}_i = \frac{1}{n_i - 1}\Big[(n_i^{(1)} - 1)\ddot{\mathbf{c}}_i^{(1)} + n_i^{(1)}(\tilde{x}_i - \tilde{x}_i^{(1)})(\tilde{x}_i - \tilde{x}_i^{(1)})^T + (n_i^{(2)} - 1)\ddot{\mathbf{c}}_i^{(2)} + n_i^{(2)}(\tilde{x}_i - \tilde{x}_i^{(2)})(\tilde{x}_i - \tilde{x}_i^{(2)})^T\Big]$

These formulas show that we do not need to access each previous sample to compute the merged values required as input of our filtering algorithm. At each merging step, we can simply aggregate the rendered buffers with the previously aggregated buffers.

6.2. Adaptive sampling map

Given a total budget of n_{budget} samples and an interval $[n_{\min}, n_{\max}]$ of allowed number of samples per pixel, we are seeking for the best strategy to distribute the samples among the pixels. $n_{\max}$ is meant to avoid spending too much samples on one pixel while $n_{\min}$ aims to ensure that we keep exploring no matter how the pixel seems to have converged.

Properly handling these constraints (the total budget and the bounds) is not as straightforward as it may seem at first sight. We believe that our approach for that specific issue is a practical contribution that may be reused for any kind of adaptive sampling scheme. Hereafter we provide only the final formulas and algorithm (see Appendix E for the whole justification).

The total number of additional samples needed to reach a relative error as close as possible to a targeted value e, while handling the bounds $n_{\min}$ and $n_{\max}$, is:

$$n'_{\text{tot}}(e) = \sum_i n'_i(e)$$

$$\text{with} \quad n'_i(e) = \text{clamp}\left(\frac{\hat{v}_i}{\max(\varepsilon^2, \hat{x}_i^2)e^2} - n_i, n_{\min}, n_{\max}\right)$$

and $\mathrm{clamp}(x,a,b) = \min(\max(x,a),b)$. The estimator of the variance of the samples $\hat{\hat{v}}_i$ is computed by:

$$\hat{\hat{v}}_i = \max(\hat{\hat{v}}_i^{(1)}, \hat{\hat{v}}_i^{(2)})$$

$$\text{where} \quad \hat{\hat{v}}_i^{(1)} = \frac{n_i - 1}{n_i}\ddot{v}_i + (\tilde{x}_i - \hat{x}_i)^T(\tilde{x}_i - \hat{x}_i)$$

$$\text{and} \quad \hat{\hat{v}}_i^{(2)} = n_i(\tilde{x}_i - \hat{x}_i)^T(\tilde{x}_i - \hat{x}_i)$$

The function $n'_{\mathrm{tot}}(e)$ is not easy to invert analytically, because of the clamping, but it is monotonic and very quick to evaluate. So we can easily invert it iteratively, in order to find the optimal error e_{opt} such that $n'_{\mathrm{tot}}(e_{\mathrm{opt}}) = n_{\mathrm{budget}}$. For simplicity, we used a dichotomous approach to get closer and closer estimates of e_{opt}, with a multiplication/division by 2 if the upper/lower bound is not known yet. The initial guessed error $e^{(0)}$ is computed as the mean error we would get if the samples were distributed uniformly:

$$e^{(0)} = \frac{1}{wh}\sum_i \frac{\sqrt{\hat{\hat{v}}_i}}{\max(\varepsilon, \hat{x}_i)\sqrt{n_i + \frac{n_{\mathrm{budget}}}{wh}}}$$

In practice, reaching the budget with 1% tolerance takes less than 10 iterations and less than one second without parallelization. One could notice that these computations could be very easily parallelized on the GPU: this is useless in our case (timings are already negligible), but could be useful for other applications like real-time progressive Monte Carlo rendering.

6.3. Adaptive rendering

The previous computations give a floating number of samples for each pixel. Similarly to Rousselle et al. [RKZ12], we round this number randomly to one of its closest integers, depending on its decimal part: 42.77 is rounded to 43 with probability 0.77, else to 42.

Thus, we modified the source code of the rendering engine (PBRT in our case) to take a map of number of samples as additional input.

7. Filtering animated sequences

When filtering a sequence of frames, our algorithm can be easily extended to exploit the information contained in neighbor frames, improving both denoising quality and temporal coherency. To do so, we just search for neighbor patches in additional frames too, with the same neighbor window, leading to more patches to build a better prior model. In the end, only the patches that belong to the current frame are filtered and aggregated in the denoised buffer.

As mentionned earlier, our adaptive sampling scheme is primarly motivated by the disturbance stemming from strong outliers. Such artifacts are even more disturbing for animated sequences, because they suddenly appear and disappear from a frame to another. Therefore, even if, as stated in Section 6, outliers do carry useful information, one may prefer to ensure an artifact-free result, depending on the application scenario. For this purpose, a simple spike-removal pre-filtering can, indeed, be very effective.

In practice we adopt a similar approach to Kalantari et al. [KBS15]. To pre-filter a pixel, we consider its 3×3 neighborhood. Then, for each color channel, we compute the average and standard deviation of these 9 pixels. If, in any channel, the central pixel is distant from the average by at least γ times the standard deviation, its color value, histogram and covariance matrix are replaced by the median pixel i.e., the one which color value has the lowest summed L1-distance to the others. The threshold γ is typically set within the range $[1.5, 2]$. The use of this spike-removal pre-filtering is illustrated in the Additional Materials (SPONZAKILLEROO scene) and in the supplementary video (SANMIGUELDOF scene).

The case of animated scene can trigger a number of additional variations to our approach. One could for instance use motion vectors (if available) to shift the neighbor window position for additional frames, which would increase the chance to find similar patches if the camera or the objects are moving too fast. Also, the marking strategy could be extended to a full 2D+time setup, denoising collaboratively pixels in neighbor frames as well. The careful analysis of these variations is left as future work.

8. Results and discussion

The parameters of our algorithms are: (i) the histogram patch distance threshold κ, (ii) the patch size p, (iii) the search window size q and (iv) the number of scales S. In this section, we compare our results to those of its closest competitor, RHF [DMB*14], which has exactly the same parameters. The default values will be $\kappa = 1$, $p = 3$, $q = 13$ and $S = 3$ for our algorithm, and $\kappa = 0.7$, $p = 3$, $q = 13$ and $S = 3$ for RHF. Note that we chose this value of κ for RHF because it was empirically giving us the best denoising results (in terms of structural similarity). Moreover, a number of comparisons to other techniques are provided in the supplemental materials. As we focus on the idea of a simple post-processing algorithm with low invasiveness, we use adaptive sampling only in the dedicated Figure 11, as a proof of concept.

The scenes are rendered with Pharr and Humphreys PBRT engine [PH10], and are taken from the corresponding website (`http://www.pbrt.org/scenes.php`), with some slight changes. Delbracio et al. slightly adapted the PBRT source code to make it output the histograms of samples. In addition, we modified it to output covariance matrices. We used 6 scenes, all at a 1024×1024 resolution:

- SANMIGUELUP: complex geometry, with a diffuse lighting.
- SANMIGUELDOF: strong depth of field effect
- SPONZAENV: dark diffuse environment map.
- TEAPOTMETAL: specular materials, with interreflections.
- SIBENIKCRASH: motion blur and perfectly specular reflections.
- CORNELLBOX: simple scene, with uniform, flat materials.

We rendered the scenes with either 256 or 64 samples per pixel (spp). Figure 4 shows a set of results we obtained with our algorithm, compared to those of Ray Histogram Fusion.

8.1. Failure of the Non-local Bayes image denoising

NL-Bayes [LBM13] assumes that its input has a uniform isotropic gaussian noise of known variance. These assumptions are not met

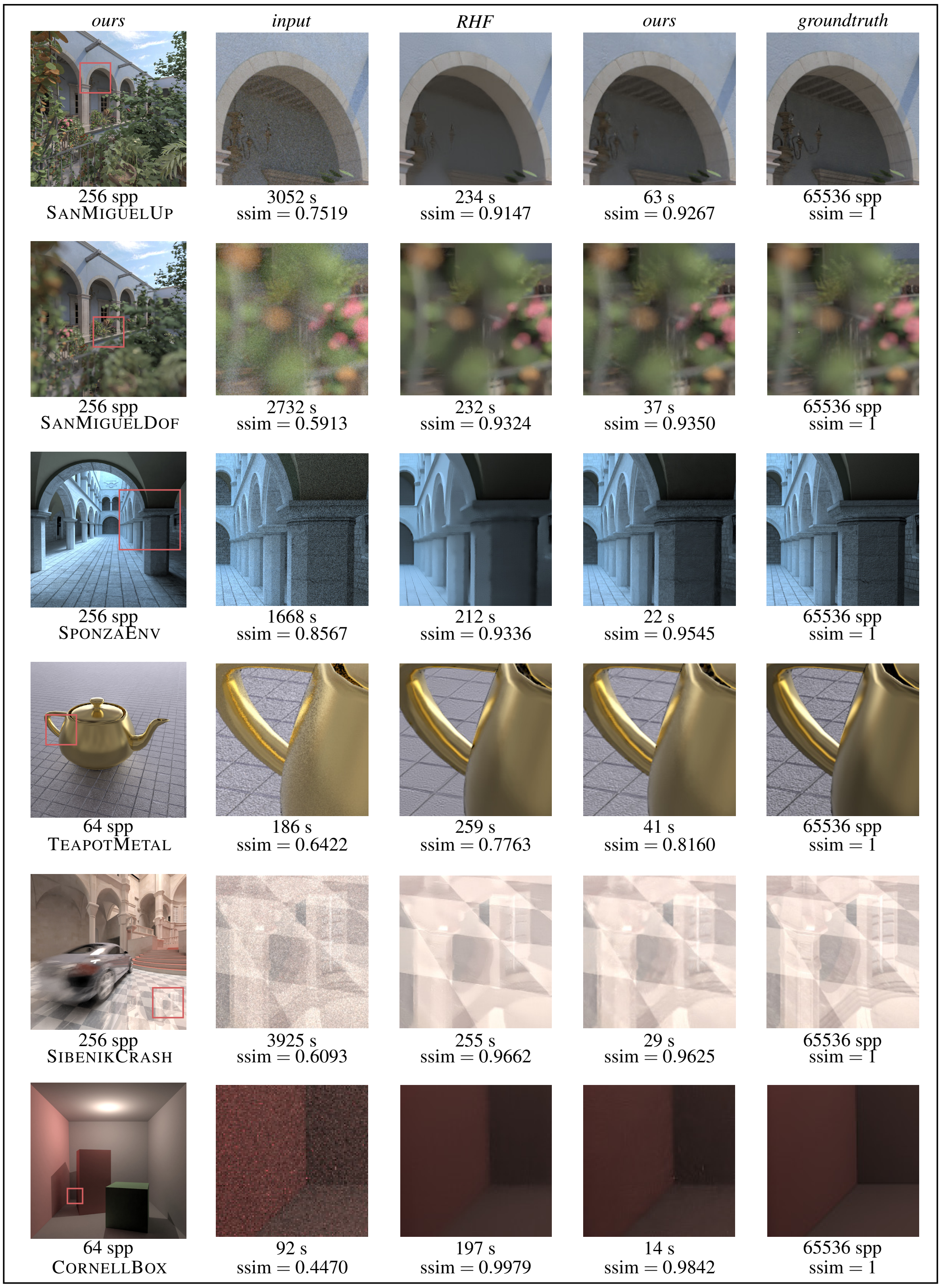

Figure 4: *Results and comparison to Ray Histogram Fusion.*

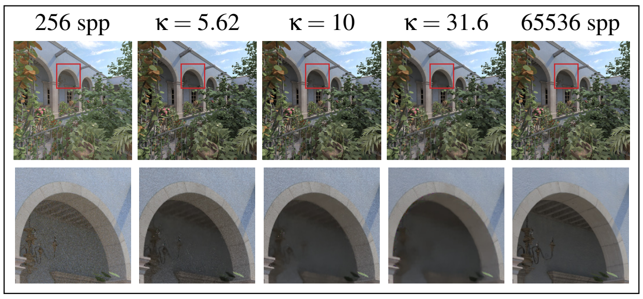

Figure 5: ***Failure of the Non-local Bayes** denoising applied directly to Monte Carlo rendering: it cannot denoise dark and bright zones at the same time, because they do not exhibit the same variance.*

by the noisy images we get from Monte Carlo rendering, and then we cannot denoise them correctly with this algorithm. Figure 5 illustrates this experimentally: no matter how we try to tune the variance parameter, we cannot get a nice tradeoff between denoising power and overblur for the whole image at the same time.

8.2. Timings

We measured timings on a 4-cores Intel Xeon processor at 2.80 GHz, and an NVIDIA GTX 680 GPU. Unless otherwise stated, all timings reported in this paper are performed by taking advantage of the 4 cores, but without GPU parallelization, to allow fair comparisons with RHF (which implementation does not use the GPU). We implemented our algorithm in C++, using OpenMP for CPU parallelization, and CUDA for GPU parallelization. Table 1 shows speed improvements we get thanks to parallelization. The Additional Materials report supplementary measurements.

In Figure 4, we can see that our algorithm is 4 to 13 times faster than RHF. The computations are more complex than the simple average of patches that is performed in RHF, but the bottleneck still remains the neighbor selection (with the comparison of patches of histograms), which is the same for both algorithms and occupies from 70 to 95% of the total computation time (besides, our GPU implementation focuses on speeding up this bottleneck only). The reason of the major increase in speed is the marking strategy, enabled by the collaborative nature of our denoising. In the main loop of our algorithm, over all the pixels, we directly skip those which have been previously marked as the centers of patches already used in a bayesian denoising computation. Thus, if we are too strict with our histogram distance threshold κ, fewer neighbor patches are selected for the collaborative denoising, fewer pixels are marked, and less time is saved by our marking strategy. Table 2 shows the strong influence of parameter κ on the computation time of our algorithm, while it does not affect at all the speed of RHF.

8.3. Denoising quality

To measure the quality of denoising, we compare our results to a reference image without noise. Therefore, we have computed for each scene a rendering with 65536 samples per pixel, that we take as our noiseless groundtruth. PSNR (Peak Signal Noise Ratio) or RelMSE (Relative Mean Squared Error) are commonly used to measure noise by comparing the noisy image with a reference. However, we chose a third measure called *Structural Similarity index* [WBSS04] (that we will denote “ssim”) , which uses local statistics of luminance values to produce results that are more reliable than PSNR and possibly more robust than RelMSE. We still report RelMSE measurements in the Additional Materials.

Figure 4 shows a set of structural similarity measures. Our algorithm globally outperforms RHF thanks to its finer filtering scheme, except for SIBENIKCRASH and the very flat, uniform scene CORNELLBOX, where the structural similarity is very high but not as good as RHF (see Section 9 for a discussion). Notice that, in SIBENIKCRASH, despite its slightly lower score, our algorithm better recovers some details in textures and reflections that are overblurred by RHF. Indeed, even the SSIM cannot properly reflect the visual assessment one can easily do looking at both results.

Compared to RHF, our algorithm especially improves the result in dark zones of the image. Indeed, in these areas, during the histogram computations, almost all the samples tend to fall into the same lowest bin. Therefore, all neighbor patches are considered similar to the central one, leading to a huge overblur, that we can observe in the SPONZADIR scene for instance (see Figure 1). Delbracio et al. already suggested using larger bins for brighter values, but adapting the bins to the arbitrarily high dynamic range of the image is not trivial, as we want to store the samples in the histograms on the fly (before knowing the full image), without increasing the number of bins, because of the cost in memory for the histogram storage and in speed for the histogram distance computation — which turns to be the bottleneck.

Fortunately, our results show that, even with a poorly discriminative selection of neighbor patches, the bayesian approach is still able to denoise very well while keeping faithful texturing (in Figure 4, see the ceiling of SANMIGUELUP, the floor of SPONZADIR and the columns of SPONZAENV).

This observation is reinforced by the graph of Figure 6, which displays how structural similarity relates to the histogram distance threshold κ. A low value of κ means a very discriminative threshold: there is not a sufficient number of patches of similar nature to denoise correctly. When we increase κ, both algorithms reach an optimum structural similarity, but then the structural similarity of Ray Histogram Fusion decreases much quicker because of overblur. Our algorithm keeps managing to get interesting denoising results despite the bad neighbor patch selection (see Figure 6).

This appreciated behavior makes perfect sense: in RHF, patches must have very close natures for the averaging to work without

CPU cores used	1	1	4	4
GPU used	no	yes	no	yes
Timing RHF (s)	834	-	270	-
Timing ours (s)	46	22	16	12

Table 1: *Effect of CPU and GPU parallelization on total computation time in our implementation (scene:* SPONZADIR*)*

κ	0.55	0.7	0.85	1	1.15	1.3
Timing RHF (s)	235	234	234	235	239	235
Timing ours (s)	96	66	46	38	33	30

Table 2: *Effect of the* κ *parameter on total computation time (scene:* SanMiguelUp*)*

overblurring the image, while our algorithm relaxes this constrain: even if the natures of patches are less close, we can still use common information to denoise each patch separately.

Delbracio et al. argued that the optimal κ should be quite stable if we keep the same rendering algorithm, and that it is possible to tweak a little bit their algorithm to search for the best κ at little cost (as a lot of computations can be put in common for all the tested κ values). These observations apply to our algorithm as well, but we add on top of them a high robustness on the choice of κ. Moreover, as mentionned in section 8.2, the higher the value of κ, the faster our algorithm, so the user interested in a fast filtering should not fear too much of increasing κ.

8.4. Highly specular materials

In the Additional Materials, we compare our method with RHF and four state-of-the-art denoising algorithms:

- ARNLMF: Adaptive Rendering with Non-Local Means Filtering [RKZ12]
- RDFCI: Robust Denoising using Feature and Color Information [RMZ13]
- WLR: Weighted Local Regression [MCY14]
- LBF: Learning-Based Filtering [KBS15]

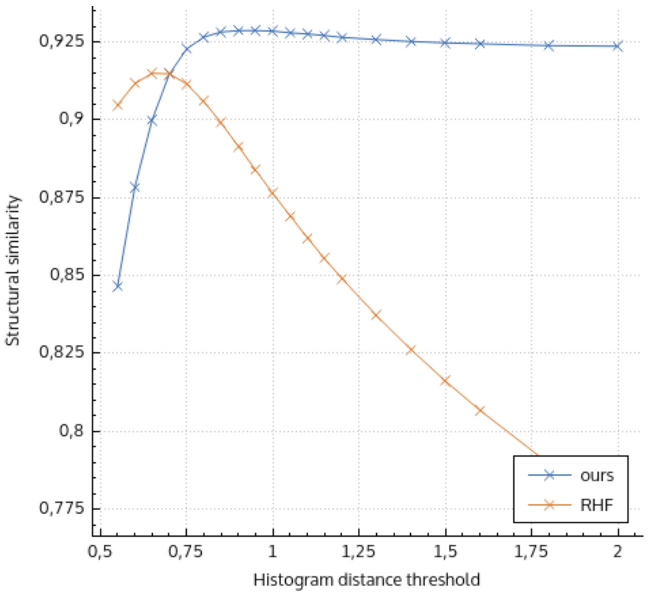

Figure 6: *Influence of the main parameter* κ *on denoising quality in the* SanMiguelUp *scene*

The last three ones are feature-based denoisers (gathered from the GBuffer and even additional feature buffers for RDFCI). The last one, LBF, is based on machine learning and needs a heavy preprocessing step on a database of noisy and groundtruth examples. Note that ARNLMF and WLR use adaptive sampling (as these techniques are inherently meant to be used with it) whereas we do not. While we significantly improve state-of-the-art sample-based denoising (i.e., RHF), we also approach very closely the performance of these feature-based and learning-based methods, exploiting only the per-pixel sample set.

Moreover, feature-based methods can be misled in some cases. For instance, motion blur and depth-of-field introduce noise in the GBuffer that must be taken care of carefully. In the presence of participating media, computing a GBuffer may not even really make sense. Finally, with highly specular materials, two neighbor pixels may have almost identical feature values but very different groundtruth values.

Figure 7 shows that all these state-of-the-art methods have overblurring artifacts in at least one of the 3 zoomed parts with highly specular reflections (first three lines). Even LBF, despite its long learning pre-process, gets fairly blurry reflections on the ground. RHF and our method do not have these artifacts on reflections, but RHF still overblurs the dark diffuse wall while we are still able to recover some texture (last line). RDFCI manages to recover well the reflections on the ground, but at the cost of overblurring its texture; moreover, the reflections on the back of the car are overblurred as well. The full images for that scene can be found in the Additional Materials. In Figure 8, we provide an even more compelling example of heavy overblurring artifacts that may occur with feature-based methods when dealing with materials which are both transparent and reflective (like the very common and widespread glass material, in this scene). Figure 9 illustrates how our sample-based filter can handle other complex effects, such as participating media, for which features from the GBuffer (e.g., depth, normal) are usually not relevant.

8.5. Multiscale filtering

As explained in Section 5, to implement our multiscale approach, we need only to define two operators: *downscaling* and *upscaling*. For the former, a gaussian pre-filtering followed by downsampling is a common choice, while for the latter, a bicubic interpolation is considered good candidate. Surprisingly, we found out that a straightforward 4-pixel average for downscaling, and a simple 4-pixel weighted average (with coefficients $\frac{9}{16}/\frac{3}{16}/\frac{3}{16}/\frac{1}{16}$ as interpolation weights) for upscaling, were sufficient to reduce low-frequency noise, as shown in Figure 10.

8.6. Adaptive sampling

As detailed in Section 6, we can use our denoising results to drive an adaptive sampling scheme. We also argued that a good adaptive sampling helps struggling against spikes artifacts. Figure 11 shows an example of result obtained by adaptive sampling. The scene is quite difficult: in the foreground, a lot of pixels are very dark, as they require several indirect light bounces to be lit. Moreover, the glossy material of the golden statue projects caustics, that are very

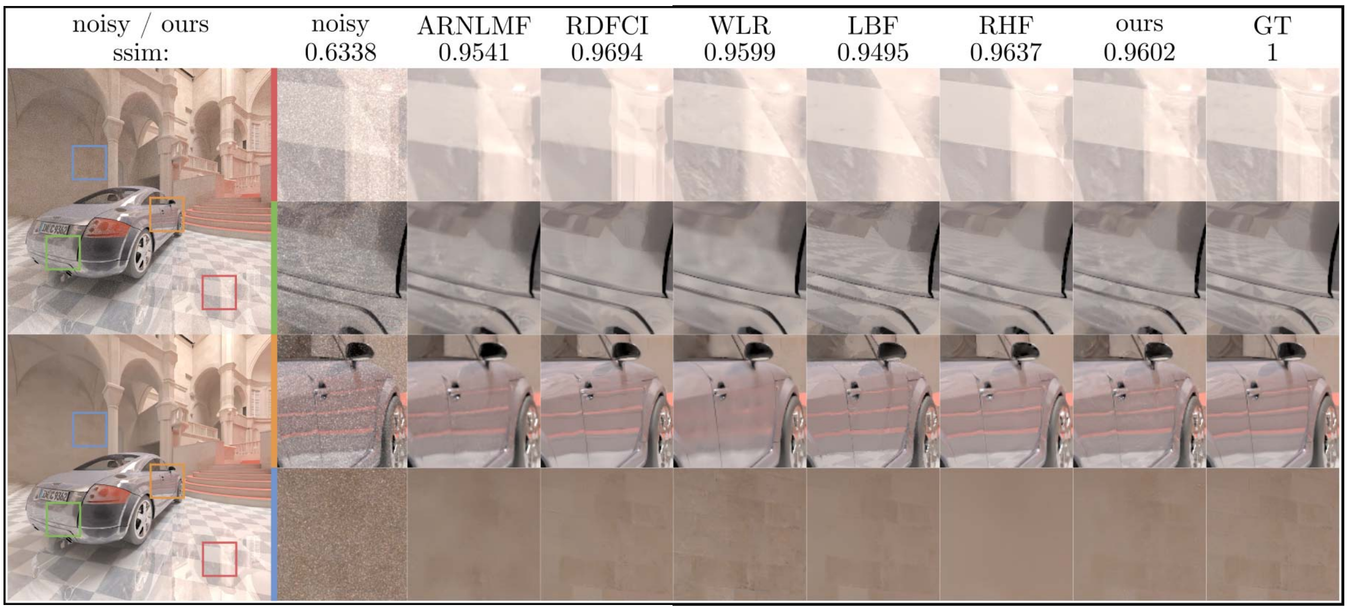

Figure 7: *Comparison of our pure sample-based denoiser with feature- and learning-based methods on a scene at 256spp with highly specular materials. Our approach provides a convincing output, while the other ones cause (sometimes severe) artifacts in at least one region of the image. Timings can be found in the Additional Materials.*

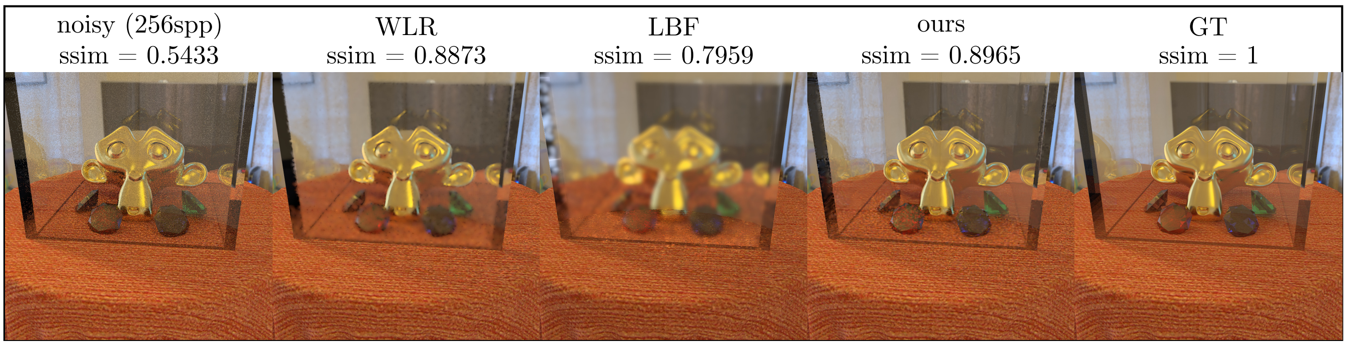

Figure 8: ***Overblurring artifacts in feature-based approaches**. When the features do not capture relevant information, heavy overblurring artifacts appear with feature-based approaches, such as in this scene with Weighted Local Regression and Learning-Based Filtering. The sample-based nature of our method makes it robust to such artifacts. Timings (in seconds) for rendering + filtering from left (noisy) to right (ours): 485, 695, 635, 543.*

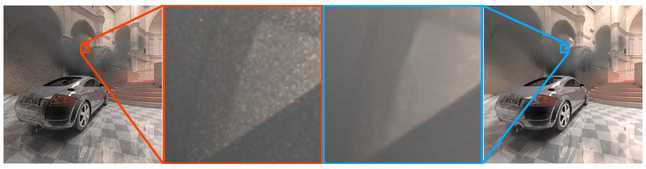

Figure 9: *The sample-based nature of our denoiser makes it able to handle complex effects in the image, such as participating media.*

difficult to capture by the simple path-tracer we are using. Worse, specularity breeds spikes even in the foreground area.

The parameters we chose for the adaptive sampling were:

- number of iterations: 8, including the uniform rendering as first step,

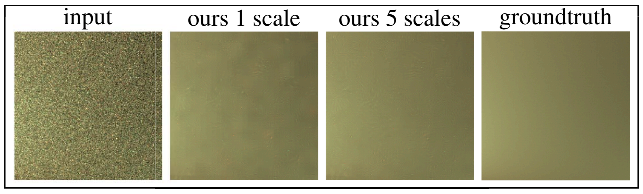

Figure 10: *Multiscale filtering to remove low-frequency noise (close-up on the dark face of the green cube in the* CORNELLBOX*).*

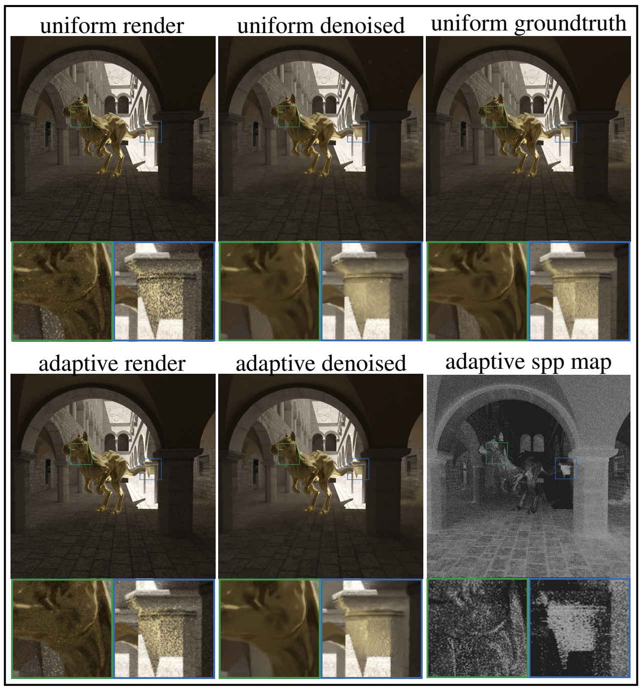

Figure 11: *Effects of adaptive sampling on a scene with caustics*

- number of samples per pixel for the first rendering: 64,
- number of samples per pixel for each adaptive rendering step: restricted to $[16, 128]$, 64 in average.

In the end we get an average of $8 \times 64 = 512$ samples per pixel. So we compare it with a uniform rendering at 512 samples per pixel.

First, we observe that adaptive sampling tends to reduce the number and the intensity of the spikes, leading to less artifacts in the final denoising. The per-pixel sample count map looks exactly as expected: brighter, directly-lit regions are given few samples (even the back of the kangaroo, that reflects the directly lit wall) whereas darker, indirectly-lit ones are given many samples. There is one exception: a lot of light paths are launched in the region of the caustic, which is exactly what we want.

Indeed, the caustic region is very interesting to analyse. We can see that even the final adaptive render did not catch caustic values in all the pixels. Still, more samples have been computed in the whole region, not only for the brighter pixels. This is because intermediate filterings have spread the caustic values around neighbors (i.e., a few very high sample values will not affect too much the histogram distance computations), causing a gap between the noisy and filtered values that will end up with triggering the evaluation of more samples even for darker pixels.

8.7. Animated sequences

We provide a supplementary video to show the results of our denoising framework on animated sequences. A part of the video compares the simple per-frame denoising with what we get when we take as additional inputs two frames before and two frames after the current one (see Section 7). The simple filtering does not have any popping or flickering artifact next to (dis)occlusion edges. However, there still remains some slight low-frequency noise, that is not temporally coherent and can be disturbing.

Our simple multi-frame approach solves this issue, improving the quality of denoising. Moreover, looking into the dark face in the foreground of the CORNELLBOX sequence, we can observe that the residual low-frequency noise becomes more temporally coherent, causing less noise flickering and therefore less visual disturbance. Note, however, that we do not activate our multiscale execution scheme in this example, which is the last component for completely getting rid of the low-frequency.

9. Limitations

In very homogenous regions, the best estimator of the groundtruth is nothing else than the simple average, which is exactly what the NL-means employed by Delbracio et al. does. When zooming on flat zones in Figure 12, we see that our algorithm does not denoise enough, as we can recognize some (attenuated) patterns that were present in the noisy input. After all, the core idea of our Bayesian approach is to share information between patches that have close but not identical natures, in order to denoise each of them individually, by keeping a bit of their specificities. Then, when we have a relatively high noise, and when all patches have almost identical natures, we are in a best case scenario for RHF, while our denoiser tends to preserve a bit of the patterns of the input. On the other hand, this is exactly what makes us preserve texture better.

We assumed that all the samples of one pixel were i.i.d. random variables. This is a crucial hypothesis, especially for all our covariance estimators. This makes our algorithm *a priori* incompatible with some rendering strategies like Metropolis Light Transport (where samples are highly correlated) or even low-discrepancy anti-aliasing (which also introduces a correlation between samples). We believe it would be interesting to study how much sample correlation actually affects the quality of denoising if we apply directly our algorithm on such a rendering anyway, and even more interesting to investigate how we can adapt our algorithm to correctly handle such cases.

Besides, if we do not have enough samples per pixel, the histograms and covariances are meaningless, and the gaussian noise hypothesis is invalidated. Our algorithm is simply not designed for low sampling rates and more adapted to production-like sampling rates. Appendix F provides quantitative results for various number of samples per pixel.

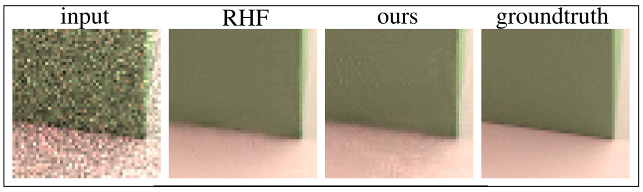

Figure 12: *Under-denoising phenomenon, particularly visible in dark homogeneous regions*

10. Conclusion and future work

We have introduced a new fast denoising method for Monte Carlo rendering. Our bayesian approach exploits rich deep images which comes with first and second order statistics on the per-pixel sample distributions, and which are easy to generate during any Monte Carlo rendering. As a result, our method combines the genericity of sample-based filters, with the details and texture preservation of the state-of-the-art bayesian denoising methods. We get globally better denoising quality than the former, especially in dark areas that are not overblurred anymore, along with about one order of magnitude speed-up. Our algorithm gives a good tradeoff between denoising quality, computation time and ease of implementation and integration. Furthermore, no parameter tweaking are needed as our results are very robust with respect to our main parameter. The collaborative nature of our approach makes it significantly faster than alternatives, while remaining compatible with a parallel computing environment. Moreover, we showed that our denoiser can drive an adaptive sampling process, trivially extends to animated sequences and can be enclosed in a multiscale execution scheme.

Beyond the mathematical justification of our approach, our experimental studies indicate that we achieve state-of-the-art denoising results. Our filtering scheme is meant to be used in the context of offline Monte Carlo rendering, where parts of the image take a very long time to converge. While we need a sufficiently large number of samples per pixel, so that histograms are populated enough, we can, in exchange, handle all-effects denoising, including highly specular materials, depth of field, motion blur and even participating media, and do not rely on complex preprocessing like machine learning (training) nor even additional, engine-dependent per-pixel features. Our denoising is also compatible with alternative adaptive sampling strategies, that can be driven by our denoising results.

With this paper, we wish to highlight two high-level ideas in particular. The first one is that our specific Bayesian framework can be a useful tool that could breed new algorithms to our community. The second one is that the field of pure sample-based methods is worth investigating further in rendering research. Although we focused on exploring how far we can push sample-based denoisers, we do not deny that feature-based ones can successfully exploit additional G-Buffer information in many scenes, with the combination of both approaches [BRM*16] being, in practice, a good strategy. However, the genericity of sample-based denoising makes it a good candidate for robust processing of high quality renderings, independently of the effects in the scene, which on the long run, may be the right direction for Monte Carlo rendering for production.

Aknowledgements. This work is partially supported by the ANR 16-LCV2-0009-01 ALLEGORI grant and the BPI France PAPAYA grant.

References

[BCM05] Buades A., Coll B., Morel J.-M.: A non-local algorithm for image denoising. In *Proc CVPR* (2005), pp. 60–65. 2

[BEEM15] Bauszat P., Eisemann M., Eisemann E., Magnor M.: General and robust error estimation and reconstruction for monte carlo rendering. *Comput. Graph. Forum 34*, 2 (2015), 597–608. 3

[BRM*16] Bitterli B., Rousselle F., Moon B., Iglesias-Guitián J. A., Adler D., Mitchell K., Jarosz W., Novák J.: Nonlinearly weighted first-order regression for denoising monte carlo renderings. *Computer Graphics Forum 35*, 4 (2016). 2, 3, 14

[DFKE06] Dabov K., Foi A., Katkovnik V., Egiazarian K.: Image denoising with block-matching and 3d filtering. In *Proc. SPIE 6064, NO. 6064A-30* (2006). 2

[DMB*14] Delbracio M., Musé P., Buades A., Chauvier J., Phelps N., Morel J.-M.: Boosting monte carlo rendering by ray histogram fusion. *ACM Trans. Graph. 33*, 1 (2014), 8:1–8:15. 2, 3, 5, 8

[ED04] Eisemann E., Durand F.: Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph. 23*, 3 (2004), 673–678. 2

[GKDS12] Georgiev I., Křivánek J., Davidovič T., Slusallek P.: Light transport simulation with vertex connection and merging. *ACM Trans. Graph. 31*, 6 (2012), 192:1–192:10. 2

[Jen96] Jensen H. W.: Global illumination using photon maps. In *Proc. EGSR* (1996), pp. 21–30. 2

[Kaj86] Kajiya J. T.: The rendering equation. *Proc. SIGGRAPH 20*, 4 (1986), 143–150. 1

[KBS15] Kalantari N. K., Bako S., Sen P.: A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph. 34*, 4 (2015), 122:1–122:12. 3, 8, 11, 17

[KS13] Kalantari N. K., Sen P.: Removing the noise in Monte Carlo rendering with general image denoising algorithms. *Computer Graphics Forum 32*, 2 (2013). 3

[LBM13] Lebrun M., Buades A., Morel J. M.: A nonlocal bayesian image denoising algorithm. *SIAM Journal on Imaging Sciences 6*, 3 (2013), 1665–1688. 2, 8

[LCBM12] Lebrun M., Colom M., Buades A., Morel J. M.: Secrets of image denoising cuisine. *Acta Numerica 21* (2012), 475–576. 3

[LW93] Lafortune E. P., Willems Y. D.: Bi-directional path tracing. In *Proc. SIGGRAPH* (1993), pp. 145–153. 2

[LWC12] Li T.-M., Wu Y.-T., Chuang Y.-Y.: Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph. 31*, 6 (2012), 194:1–194:9. 3

[MCY14] Moon B., Carr N., Yoon S.-E.: Adaptive rendering based on weighted local regression. *ACM Trans. Graph. 33*, 5 (2014), 170:1–170:14. 3, 11

[MIGYM15] Moon B., Iglesias-Guitian J. A., Yoon S.-E., Mitchell K.: Adaptive rendering with linear predictions. *ACM Trans. Graph. 34*, 4 (2015), 121:1–121:11. 3

[MMMG16] Moon B., McDonagh S., Mitchell K., Gross M.: Adaptive polynomial rendering. *ACM Trans. Graph. 35*, 4 (2016). 3

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering, 2nd Ed.*, 2nd ed. Morgan Kaufmann Publishers Inc., 2010. 8

[PSA*04] Petschnigg G., Szeliski R., Agrawala M., Cohen M., Hoppe H., Toyama K.: Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. 23*, 3 (2004), 664–672. 2

[RKZ11] Rousselle F., Knaus C., Zwicker M.: Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph. 30*, 6 (2011), 159:1–159:12. 3

[RKZ12] Rousselle F., Knaus C., Zwicker M.: Adaptive rendering with non-local means filtering. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 195:1–195:11. 3, 8, 11

[RMZ13] Rousselle F., Manzi M., Zwicker M.: Robust denoising using feature and color information. *Computer Graphics Forum 32*, 7 (2013), 121–130. 3, 5, 11

[SD12] Sen P., Darabi S.: On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph. 31*, 3 (2012). 3

[TM98] Tomasi C., Manduchi R.: Bilateral filtering for gray and color images. In *Proc. ICCV* (1998), pp. 839–846. 2

[VG97] Veach E., Guibas L. J.: Metropolis light transport. In *Proc. SIGGRAPH* (1997), pp. 65–76. 2

[WBSS04] Wang Z., Bovik A., Sheikh H., Simoncelli E.: Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13*, 4 (2004), 600–612. 10

[XP05] Xu R., Pattanaik S. N.: A novel monte carlo noise reduction operator. *IEEE Comput. Graph. Appl. 25*, 2 (2005), 31–35. 3

[ZJL*15] Zwicker M., Jarosz W., Lehtinen J., Moon B., Ramamoorthi R., Rousselle F., Sen P., Soler C., Yoon S.-E.: Recent advances in adaptive sampling and reconstruction for monte carlo rendering. *Comput. Graph. Forum 34*, 2 (2015), 667–681. 3

Appendix A: Expectation of the empirical covariance matrix of a noisy point set

Having the following set up:

- $\{\mathring{X}_1, \mathring{X}_2, ...\mathring{X}_N\}$ a fixed point cloud,
- $\overline{\mathring{X}} = \frac{1}{N}\sum_{i=1}^{N}\mathring{X}_i$ its empirical mean,
- $\mathring{\mathfrak{S}} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathring{X}_i - \overline{\mathring{X}})(\mathring{X}_i - \overline{\mathring{X}})^T$ its empirical covariance matrix,
- $\{N_1, N_2, ...N_N\}$ a set of N independant gaussian random variables. They are all centered but each one has its own covariance matrix $\mathbf{C}_i$ ($N_i \sim \mathcal{N}(0, \mathbf{C}_i)$),
- $\{\tilde{X}_1, \tilde{X}_2, ...\tilde{X}_N\}$ the random noisy version of the point cloud: $\forall i, \tilde{X}_i = \mathring{X}_i + N_i$,
- $\overline{\tilde{X}} = \frac{1}{N}\sum_{i=1}^{N}\tilde{X}_i$ its empirical mean,
- $\tilde{\mathfrak{S}} = \frac{1}{N-1}\sum_{i=1}^{N}(\tilde{X}_i - \overline{\tilde{X}})(\tilde{X}_i - \overline{\tilde{X}})^T$ its empirical covariance matrix,

we want to compute the expectation of the random variable $\tilde{\mathfrak{S}}$:

$$\mathbb{E}[\tilde{\mathfrak{S}}] = \frac{1}{N-1}\sum_{i=1}^{N}\left[\mathbb{E}\left[\tilde{X}_i\tilde{X}_i^{T}\right] + \mathbb{E}\left[\overline{\tilde{X}}\,\overline{\tilde{X}}^{T}\right] - \mathbb{E}\left[\tilde{X}_i\overline{\tilde{X}}^{T}\right] - \mathbb{E}\left[\overline{\tilde{X}}\tilde{X}_i^{T}\right]\right]$$

Let us compute some intermediate results:

$$\begin{aligned}
\mathbb{E}\left[\tilde{X}_i\right] &= \mathring{X}_i \\
\mathbb{E}\left[\tilde{X}_i\tilde{X}_i^{T}\right] &= \mathring{X}_i\mathring{X}_i^{T} + \mathring{X}_i\mathbb{E}[N_i]^T + \mathbb{E}[N_i]\mathring{X}_i^{T} + \mathbb{E}[N_iN_i^T] \\
&= \mathring{X}_i\mathring{X}_i^{T} + \mathbf{C}_i \\
\mathbb{E}\left[\tilde{X}_i\tilde{X}_j^{T}\right] &= \mathbb{E}\left[\tilde{X}_i\right]\mathbb{E}\left[\tilde{X}_j\right]^T = \mathring{X}_i\mathring{X}_j^{T} \quad \forall j \neq i \\
\mathbb{E}\left[\overline{\tilde{X}}\tilde{X}_i^{T}\right] &= \frac{1}{N}\mathbb{E}\left[\tilde{X}_i\tilde{X}_i^{T}\right] + \frac{1}{N}\sum_{j\neq i}\mathbb{E}\left[\tilde{X}_j\tilde{X}_i^{T}\right] \\
&= \frac{1}{N}\mathring{X}_i\mathring{X}_i^{T} + \frac{1}{N}\mathbf{C}_i + \frac{1}{N}\left(\sum_{j\neq i}\mathring{X}_j\right)\mathring{X}_i^{T} \\
&= \frac{1}{N}\mathbf{C}_i + \overline{\mathring{X}}\mathring{X}_i^{T} \\
\mathbb{E}\left[\tilde{X}_i\overline{\tilde{X}}^{T}\right] &= \frac{1}{N}\mathbf{C}_i + \mathring{X}_i\overline{\mathring{X}}^{T} \\
\mathbb{E}\left[\overline{\tilde{X}}\,\overline{\tilde{X}}^{T}\right] &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\left[\overline{\tilde{X}}\tilde{X}_i^{T}\right] \\
&= \frac{1}{N}\overline{\mathbf{C}} + \overline{\mathring{X}}\,\overline{\mathring{X}}^{T}
\end{aligned}$$

with $\overline{\mathbf{C}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{C}_i$. We can thus come back to the expectation of the empirical covariance:

$$\mathbb{E}[\tilde{\mathfrak{S}}] = \frac{1}{N-1}\sum_{i=1}^{N}\left(\mathring{X}_i\mathring{X}_i^{T} + \mathbf{C}_i + \frac{1}{N}\overline{\mathbf{C}} + \overline{\mathring{X}}\,\overline{\mathring{X}}^{T} - \frac{1}{N}\mathbf{C}_i - \mathring{X}_i\overline{\mathring{X}}^{T} - \frac{1}{N}\mathbf{C}_i - \overline{\mathring{X}}\mathring{X}_i^{T}\right)$$

By regrouping the terms, and using that $\sum_{i=1}^{N}\mathbf{C}_i = N\overline{\mathbf{C}}$:

$$\begin{aligned}
\mathbb{E}[\tilde{\mathfrak{S}}] &= \frac{1}{N-1}\left(\sum_{i=1}^{N}(\mathring{X}_i - \overline{\mathring{X}})(\mathring{X}_i - \overline{\mathring{X}})^T\right) \\
&\quad + \frac{1}{N-1}\left(N\overline{\mathbf{C}} + \overline{\mathbf{C}} - \overline{\mathbf{C}} - \overline{\mathbf{C}}\right) \\
&= \boxed{\mathring{\mathfrak{S}} + \overline{\mathbf{C}}}
\end{aligned}$$

Appendix B: Histogram Distance

Comparing two histograms can be done in many ways, from simple L^1 or L^2 distance to the more complex *Earth Mover* one. We use the χ^2 distance:

$$\begin{aligned}
\mathcal{D}(h_i, h_j) &= \frac{1}{|\mathcal{B}_{i,j}|}\sum_{b\in\mathcal{B}_{i,j}}\frac{\left(\sqrt{\frac{n_j}{n_i}}h_i^b - \sqrt{\frac{n_i}{n_j}}h_j^b\right)^2}{h_i^b + h_j^b} \\
&= \frac{1}{|\mathcal{B}_{i,j}|}\sum_{b\in\mathcal{B}_{i,j}}\frac{\left(n_jh_i^b - n_ih_j^b\right)^2}{n_in_j(h_i^b + h_j^b)}
\end{aligned}$$

with n_{bins} the number of bins of the histograms, h_i^b the number stored in the bth bin of histogram h_i, $\mathcal{B}_{i,j} = \{b \in [1, n_{\text{bins}}]\,,\, h_i^b + h_j^b > 0\}$ the set of bins that are non-empty for h_i or h_j, and $|\mathcal{B}_{i,j}|$ its cardinal. To deal with patches, we concatenate the histograms bins and take the χ^2 distance between the extended histograms:

$$\mathcal{D}(H_i, H_j) = \frac{1}{\sum_{p_k\in P_i}|\mathcal{B}_{k,l}|}\sum_{p_k\in P_i}\sum_{b\in\mathcal{B}_{k,l}}\frac{\left(n_lh_k^b - n_kh_l^b\right)^2}{n_kn_l(h_k^b + h_l^b)}$$

with $p_l = p_j + (p_k - p_i)$, such that p_l is the pixel in patch P_j that matches with pixel p_k in patch P_i.

Appendix C: First step: mathematical derivations

In the first step of our algorithm (Section 4.2), we want to solve:

$$\hat{X}_i = \operatorname*{argmax}_X \mathbb{P}(\tilde{X}_i\,|\,X)\mathbb{P}(X) \tag{3}$$

with:

$$\mathbb{P}(\tilde{X}_i\,|\,X) = \alpha\exp\left(-\frac{1}{2}(X - \tilde{X}_i)^T\mathbf{C}_i^{-1}(X - \tilde{X}_i)\right)$$

$$\mathbb{P}(X) = \beta\exp\left(-\frac{1}{2}(X - \overline{\tilde{X}})^T(\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}(X - \overline{\tilde{X}})\right)$$

Back to Equation 3, by removing the constants and the exponentials, we get:

$$\begin{aligned}
\hat{X}_j = \operatorname*{argmin}_X \Big[&(X - \tilde{X}_j)^T\mathbf{C}_j^{-1}(X - \tilde{X}_j) \\
&+ (X - \overline{\tilde{X}})^T(\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}(X - \overline{\tilde{X}})\Big]
\end{aligned}$$

Then, to find the minimum, we differentiate this expression with respect to X, and put it equal to zero:

$$\mathbf{C}_j^{-1}(\hat{X}_j - \tilde{X}_j) + (\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}(\hat{X}_j - \overline{\tilde{X}}) = 0$$

$$[\mathbf{C}_j^{-1} + (\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}]\hat{X}_j = \mathbf{C}_j^{-1}\tilde{X}_j + (\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}\overline{\tilde{X}}$$

Multiplying by $\mathbf{C}_j$ gives:

$$[\mathbf{I} + \mathbf{C}_j(\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}]\hat{X}_j = \tilde{X}_j + \mathbf{C}_j(\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}\overline{\tilde{X}}$$

We then compute the following simplification:

$$\begin{aligned}\mathbf{I} + \mathbf{C}_j(\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1} &= (\tilde{\mathfrak{S}} - \overline{\mathbf{C}})(\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1} + \mathbf{C}_j(\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}\\ &= (\tilde{\mathfrak{S}} - \overline{\mathbf{C}} + \mathbf{C}_j)(\tilde{\mathfrak{S}} - \overline{\mathbf{C}})^{-1}\\ &= \tilde{\mathfrak{S}}'_j(\tilde{\mathfrak{S}}'_j - \mathbf{C}_j)^{-1}\end{aligned}$$

where we set $\tilde{\mathfrak{S}}'_j = \tilde{\mathfrak{S}} - \overline{\mathbf{C}} + \mathbf{C}_j$. Finally:

$$\begin{aligned}\hat{X}_j &= (\tilde{\mathfrak{S}}'_j - \mathbf{C}_j)\tilde{\mathfrak{S}}'^{-1}_j[\tilde{X}_j + \mathbf{C}_j(\tilde{\mathfrak{S}}'_j - \mathbf{C}_j)^{-1}\overline{\tilde{X}}]\\ &= (\mathbf{I} - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j)\tilde{X}_j + (\mathbf{I} - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j)\mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j(\mathbf{I} - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j)^{-1}\overline{\tilde{X}}\\ &= (\mathbf{I} - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j)\tilde{X}_j + \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j(\mathbf{I} - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j)(\mathbf{I} - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j)^{-1}\overline{\tilde{X}}\\ &= (\mathbf{I} - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j)\tilde{X}_j + \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j\overline{\tilde{X}}\\ &= \boxed{\tilde{X}_j - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j(\tilde{X}_j - \overline{\tilde{X}})} \qquad (4)\\ &= \overline{\tilde{X}} - \overline{\tilde{X}} + \tilde{X}_j - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j(\tilde{X}_j - \overline{\tilde{X}})\\ &= \overline{\tilde{X}} + (\mathbf{I} - \mathbf{C}_j\tilde{\mathfrak{S}}'^{-1}_j)(\tilde{X}_j - \overline{\tilde{X}})\\ &= \boxed{\overline{\tilde{X}} + (\tilde{\mathfrak{S}}'_j - \mathbf{C}_j)\tilde{\mathfrak{S}}'^{-1}_j(\tilde{X}_j - \overline{\tilde{X}})} \qquad (5)\end{aligned}$$

While we use Formula 4 in practice, Formula 5 may be more explanatory about what we really compute: a form of "shrinkage" around the mean patch (see Figure 3).

(6)

Appendix D: Merging covariance matrices

Let $s_i^1, \dots s_i^{n_i^{(1)}}$ be the $n_i^{(1)}$ samples generated by a first rendering and $s_i^{n_i^{(1)}+1}, \dots s_i^{n_i^{(1)}+n_i^{(2)}}$ the ones from a second rendering. To shorten formulas, we will define the $\mathcal{Q}$ operator that turns a vector into a matrix: $\mathcal{Q}(X) = XX^T$. We denote:

$$n_i = n_i^{(1)} + n_i^{(2)},\ \tilde{x}_i^{(1)} = \frac{1}{n_i^{(1)}}\sum_{k=1}^{n_i^{(1)}} s_i^k,\ \tilde{x}_i^{(2)} = \frac{1}{n_i^{(2)}}\sum_{k=n_i^{(1)}+1}^{n_i} s_i^k$$

$$\ddot{\mathbf{c}}_i^{(1)} = \frac{1}{n_i^{(1)} - 1}\sum_{k=1}^{n_i^{(1)}} \mathcal{Q}(s_i^k - \tilde{x}_i^{(1)})$$

$$\ddot{\mathbf{c}}_i^{(2)} = \frac{1}{n_i^{(2)} - 1}\sum_{k=n_i^{(1)}+1}^{n_i} \mathcal{Q}(s_i^k - \tilde{x}_i^{(2)})$$

The merged mean value is:

$$\tilde{x}_i = \frac{1}{n_i}\sum_{k=1}^{n_i} s_i^k = \frac{1}{n_i}(n_i^{(1)}\tilde{x}_i^{(1)} + n_i^{(2)}\tilde{x}_i^{(2)})$$

And the merged covariance matrix:

$$\begin{aligned}\ddot{\mathbf{c}}_i &= \frac{1}{n_i - 1}\sum_{k=1}^{n_i}\mathcal{Q}(s_i^k - \tilde{x}_i)\\ &= \frac{1}{n_i - 1}\left[\sum_{k=1}^{n_i^{(1)}}\mathcal{Q}\left((s_i^k - \tilde{x}_i^{(1)}) + (\tilde{x}_i^{(1)} - \tilde{x}_i)\right)\right.\\ &\qquad \left. + \sum_{k=n_i^{(1)}+1}^{n_i}\mathcal{Q}\left((s_i^k - \tilde{x}_i^{(2)}) + (\tilde{x}_i^{(2)} - \tilde{x}_i)\right)\right]\end{aligned}$$

Using the fact that $\sum_{k=1}^{n_i^{(1)}}(s_i^k - \tilde{x}_i^{(1)}) = 0$ and $\sum_{k=n_i^{(1)}+1}^{n_i}(s_i^k - \tilde{x}_i^{(2)}) = 0$, we get, after developing and removing zero terms:

$$\begin{aligned}\ddot{\mathbf{c}}_i = \frac{1}{n_i - 1}&\left[\sum_{k=1}^{n_i^{(1)}}\left(\mathcal{Q}(s_i^k - \tilde{x}_i^{(1)}) + \mathcal{Q}(\tilde{x}_i^{(1)} - \tilde{x}_i)\right)\right.\\ &\left. + \sum_{k=n_i^{(1)}+1}^{n_i}\left(\mathcal{Q}(s_i^k - \tilde{x}_i^{(2)}) + \mathcal{Q}(\tilde{x}_i^{(2)} - \tilde{x}_i)\right)\right]\end{aligned}$$

Hence the final formula:

$$\begin{aligned}\ddot{\mathbf{c}}_i = \frac{1}{n_i - 1}&\left[(n_i^{(1)} - 1)\ddot{\mathbf{c}}_i^{(1)} + n_i^{(1)}(\tilde{x}_i - \tilde{x}_i^{(1)})(\tilde{x}_i - \tilde{x}_i^{(1)})^T\right.\\ &\left. + (n_i^{(2)} - 1)\ddot{\mathbf{c}}_i^{(2)} + n_i^{(2)}(\tilde{x}_i - \tilde{x}_i^{(2)})(\tilde{x}_i - \tilde{x}_i^{(2)})^T\right]\end{aligned}$$

Appendix E: Adaptive sampling map computation details

In our adaptive sampling strategy, we seek the best way to determine for each pixel p_i its number of new samples to compute n'_i (with the constraints $\sum_i n'_i \approx n_{\text{budget}}$ and $n'_i \in [n_{\min}, n_{\max}]$).

To generate more samples in difficult pixels, a first solution is to use a measure of the pixel variance. Just as we have done for covariance, pixel variance $v_i = \text{Tr}(\mathbf{c}_i)$ (the trace of the pixel covariance matrix) can be estimated from the empirical variance of samples $\ddot{v}_i = \text{Tr}(\ddot{\mathbf{c}}_i)$ and the number of samples n_i: $v_i = \frac{\ddot{v}_i}{n_i}$. Doubling the number of samples of a pixel will basically halve its variance. So we could compute the number of samples to generate in order to get the same standard deviation for every pixel.

However, a standard deviation of $\sqrt{v_i} = 0.1$ around a mean value of $\tilde{x}_i = 0.2$ is visually much more disturbing than around $\tilde{x}_i = 10$. Hence, we consider *relative* standard deviation $e_i = \frac{\sqrt{v_i}}{\max(\varepsilon, \tilde{x}_i)}$ as the error criterion that we want to minimize, with ε a small constant meant to avoid division by zero.

Another issue is that these measures are not sufficient to determine whether a pixel has converged. Maybe all the rays have missed the main source of energy (a caustic for instance), leading to a pixel that looks to have converged in terms of pixel relative standard deviation, but which is actually far from its groundtruth. A possible way to detect these pixels is to rely on the denoising: neighbors may have captured the missing energy and shared it through the filtering. Therefore we take into account the filtered value in our adaptive sampling scheme by modifying the estimated variance.

The variance of a random (vector) variable z is $\mathbf{Var}[z] =$

$\mathbb{E}[(z \quad \mathbb{E}[z])^T(z \quad \mathbb{E}[z])]$. If we have n independent realizations of this variable $z_1 ... z_n$, an unbiased estimator of this variance is $\frac{1}{n \quad 1}\sum_{k=1}^{n}(z_k \quad \bar{z})^T(z_k \quad \bar{z})$ where $\bar{z} = \frac{1}{n}\sum_{k=1}^{n} z_k$. However, if we knew the actual value of $\mathbb{E}[z]$, an unbiased estimator would be $\frac{1}{n}\sum_{k=1}^{n}(z_k \quad \mathbb{E}[z])^T(z_k \quad \mathbb{E}[z])$ (notice the n instead of $n \quad 1$).

Consequently, we update the estimated sample variance by pretending that the filtered value $\hat{x}_i$ is the actual groundtruth value, yielding:

$$\begin{aligned}\hat{\ddot{v}}_i^{(1)} &= \frac{1}{n_i}\sum_{k=1}^{n_i}(s_i^k \quad \hat{x}_i)^T(s_i^k \quad \hat{x}_i)\\ &= \frac{1}{n_i}\sum_{k=1}^{n_i}((s_i^k \quad \tilde{x}_i)+(\tilde{x}_i \quad \hat{x}_i))^T((s_i^k \quad \tilde{x}_i)+(\tilde{x}_i \quad \hat{x}_i))\end{aligned}$$

By developing, and using $\sum_{k=1}^{n_i}(s_i^k \quad \tilde{x}_i) = 0$, we get:

$$\begin{aligned}\hat{\ddot{v}}_i^{(1)} &= \frac{1}{n_i}\sum_{k=1}^{n_i}(s_i^k \quad \tilde{x}_i)^T(s_i^k \quad \tilde{x}_i) + \frac{1}{n_i}\sum_{k=1}^{n_i}(\tilde{x}_i \quad \hat{x}_i)^T(\tilde{x}_i \quad \hat{x}_i)\\ &= \frac{n_i \quad 1}{n_i}\ddot{v}_i + (\tilde{x}_i \quad \hat{x}_i)^T(\tilde{x}_i \quad \hat{x}_i)\end{aligned}$$

With this formula, we take into account both the measured sample variance and the gap between noisy and denoised values, without any arbitrary weighting scheme. Moreover, with the hypothesis of $\hat{x}_i$ being the groundtruth, $\hat{\ddot{v}}_i^{(1)}$ is unbiased.

However, we observed that this estimator was not behaving as expected for a special case that happened too often to be ignored. In dark areas, the probability distribution of s_i may generate completely black samples with a quite high probability i.e., when it is difficult to find a light path that carries energy.

If all the samples are black, we get $\ddot{v}_i = 0$, $\tilde{x}_i = 0$, and then $\hat{\ddot{v}}_i^{(1)} = \hat{x}_i^T\hat{x}_i$. This formula does not depend on n_i. However, when increasing n_i, if we keep getting only black samples and the same expected value $\hat{x}_i$, we should deduce that the non-black samples that we are missing should have higher and higher values to counterbalance the black ones and reach an average close to $\hat{x}_i$. This yields that the estimated variance of samples $\hat{\ddot{v}}_i^{(1)}$ should increase with n_i.

Hence, we propose to use a second estimator. $(\tilde{x}_i \quad \hat{x}_i)^T(\tilde{x}_i \quad \hat{x}_i)$ is a noisy but unbiased estimator of the *pixel* variance. So we can easily deduce a second unbiased estimator of the variance *of the samples*:

$$\hat{\ddot{v}}_i^{(2)} = n_i(\tilde{x}_i \quad \hat{x}_i)^T(\tilde{x}_i \quad \hat{x}_i)$$

This estimator should be noisier than the first one, but it behaves better for the special case we have just described. To mix the benefits of both estimators, and because we prefer overestimating variance of uncertain pixels rather than keeping an unbiased estimator, we simply take the maximum as our final estimate:

$$\hat{\ddot{v}}_i = \max(\hat{\ddot{v}}_i^{(1)}, \hat{\ddot{v}}_i^{(2)})$$

We then define the updated estimate of pixel variance $\hat{v}_i = \frac{\hat{\ddot{v}}_i}{n_i}$ and the updated estimate of pixel relative standard deviation:

$$\hat{e}_i = \frac{\sqrt{\hat{v}_i}}{\max(\varepsilon, \hat{x}_i)} = \frac{\sqrt{\hat{\ddot{v}}_i}}{\max(\varepsilon, \hat{x}_i)\sqrt{n_i}}$$

Now, if we add n_i' samples, assuming that $\hat{\ddot{v}}_i$ and $\hat{x}_i$ remain stable, the error becomes:

$$\hat{e}_i'(n_i') = \frac{\sqrt{\hat{\ddot{v}}_i}}{\max(\varepsilon, \hat{x}_i)\sqrt{n_i + n_i'}}$$

We can inverse this relationship: we can compute the number of samples that is needed to achieve a certain error goal. This value must be clamped between our bounds $n_{\min}$ and $n_{\max}$. By summing over all the pixels, we get the total number of samples n_{tot}'.

$$n_i'(e) = \text{clamp}\left(\frac{\hat{\ddot{v}}_i}{\max(\varepsilon^2, \hat{x}_i{}^2)e^2} \quad n_i, n_{\min}, n_{\max}\right)$$

$$n_{\text{tot}}'(e) = \sum_i n_i'(e)$$

We then use the dichotomous approach described in Section 6.2 to find the error value that makes us fulfill the sample budget n_{budget}.

Appendix F: Comparison for various numbers of samples per pixels

Our sample-based algorithm is not meant do be used with a very low number of samples per pixel, as this is the sole information it exploits. In this context, we cannot assume the noise of a pixel to be gaussian, and the histograms and covariance matrices do not represent well the underlying distributions. However, for the sake of completeness, we provide a comparison with other algorithms for 4 to 1024 samples per pixel, in Figure 13. The Learning-Based Filtering method [KBS15] gets impressive results at low sampling rate, at the cost of a heavy pre-process learning phase.

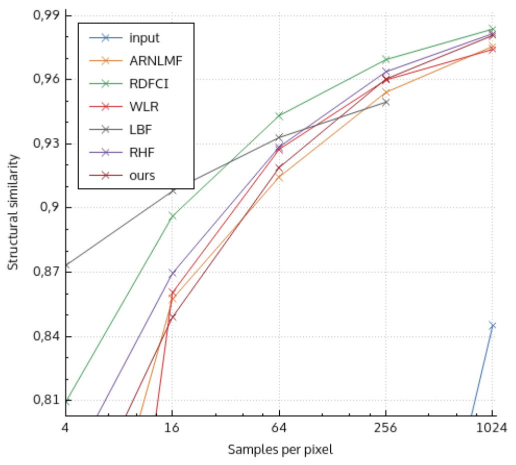

Figure 13: *Comparison with other algorithms with respect to the number of samples per pixel, for the* SIBENIKCAR *scene. Our method needs at least around 64 samples per pixel to be competitive.*