Distributing Monte Carlo Errors as a Blue Noise in Screen Space by Permuting Pixel Seeds Between Frames

Eric Heitz, Laurent Belcour

Eurographics Symposium on Rendering 2019
T. Boubekeur and P. Sen
(Guest Editors)

Volume 38 (2019), Number 4

Distributing Monte Carlo Errors as a Blue Noise in Screen Space by Permuting Pixel Seeds Between Frames

E. Heitz and L. Belcour

Unity Technologies

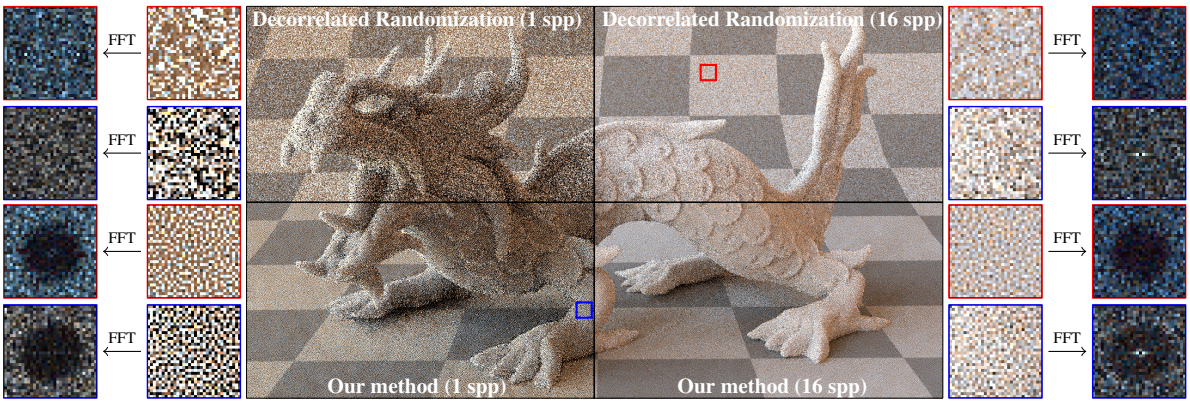

Figure 1: Distributing Monte Carlo errors as a blue noise in screen space. *Monte Carlo noise in raytraced renderings has usually a white spectrum because of the randomization used to decorrelate pixel estimates. Our temporal algorithm correlates pixel estimates to obtain a noise with a blue spectrum like dithered images. This makes the images appear less noisy despite the errors having statistically the same amplitudes. In this scene, the dragon is a participating medium rendered with up to* 20 *scattering events under coherent motion.*

Abstract

Recent work has shown that distributing Monte Carlo errors as a blue noise in screen space improves the perceptual quality of rendered images. However, obtaining such distributions remains an open problem with high sample counts and high-dimensional rendering integrals. In this paper, we introduce a temporal algorithm that aims at overcoming these limitations. Our algorithm is applicable whenever multiple frames are rendered, typically for animated sequences or interactive applications. Our algorithm locally permutes the pixel sequences (represented by their seeds) to improve the error distribution across frames. Our approach works regardless of the sample count or the dimensionality and significantly improves the images in low-varying screen-space regions under coherent motion. Furthermore, it adds negligible overhead compared to the rendering times. Note: our supplemental material provides more results with interactive comparisons against previous work.

CCS Concepts

• ***Computing methodologies*** → ***Rendering;***

1. Introduction

Rendering via Monte Carlo (MC) integration is subject to numerical errors. The *amplitude* of these integration errors is best attenuated via variance-reduction techniques such as importance sampling combined with high-convergence-rate sequences. Nevertheless, the errors remain present and their visual impact depends on their *screen-space distribution*. Classically, two options are considered: either *aliasing* (the pixels use the same sequence) or *white noise* (the pixels use decorrelated random sequences).

Inspired by halftoning algorithms, Georgiev and Fajardo [GF16] introduced another option that achieves superior results. They noticed that distributing the errors as a *blue noise* makes it less apparent and thus improves the perceptual quality of the images. This can be achieved by correlating pixel estimates (the pixels use different but correlated sequences). Figure 1 illustrates this effect by comparing blue-noise-error renderings (the spectrum has no low-frequencies) to classic white-noise-error renderings (the spectrum is flat). The former appear less noisy despite the errors having statistically the same amplitudes.

Blue-noise dithered sampling. Georgiev and Fajardo introduced *blue-noise dithered sampling* (BNDS), a technique that uses blue-noise dithering tiles [Uli93] to correlate the per-pixel sequences in screen-space. BNDS has quickly been adopted in the industry for games [GS16], virtual reality [LHR18], real-time raytracing [Sch19], or preview at low sample counts [GIF*18]. The reason for this adoption is because BNDS excels at one sample per pixel with simple rendering integrals, which is the typical use case of these applications. Unfortunately, the blue-noise properties of BNDS vanish rapidly as the number of samples or the dimensionality of the integrals increase. As a result, BNDS fails to provide significant improvements to production Monte Carlo rendering with many rays per pixel that bounce multiple times.

Contributions. Inspired by the high-quality results achieved by BNDS in the aforementioned applications, our motivation is to bring the benefits of distributing the errors as a blue noise in screen space to production rendering. Like BNDS, we consider a rendering context where each pixel uses a sequence of random numbers to estimate the Monte Carlo rendering integral. Our goal is to choose the sequence of each pixel such that the resulting errors will be distributed as a blue noise in screen space. In this context, BNDS can be categorized as an *a priori* method in the sense that it chooses the sequences regardless of the actual rendering integrand (regardless of the scene). We advocate that using *a posteriori* methods that optimize the choice of the sequence for a specific integrand (for the scene) is the key to scale to higher sample counts and dimensionalities. To support this claim, we make the following contributions:

- In Section 3, we introduce an *a posteriori* formulation of the rendering operation. Our formulation shows how to compute Monte Carlo renderings with high-quality blue-noise distribution of the errors regardless of the sample count and the dimensionality. The formulation is not practical but provides useful insights and motivates the exploration of *a posteriori* methods.
- In Section 4, we introduce a practical temporal algorithm that approximates this *a posteriori* formulation by applying local permutations to the pixel sequences (represented by their seeds) between two frames, as shown in Figure 2.
- In Section 5, we evaluate our temporal algorithm under varying rendering conditions. In screen-space regions that are locally constant and coherent under motion, our algorithm achieves high-quality blue-noise error distributions regardless of the sample count or the dimensionality. Otherwise, in the worst cases, it produces a white-noise error distribution equivalent to a classic decorrelated randomization.

2. Previous work

Temporal algorithms. Our temporal algorithm builds upon the fact that we have access to the previous frame to improve the current frame. This is reminescent of Temporal Anti-Aliasing (TAA) [MA06, SKW*17, SPD18]. Note however that the execution is fundamentally different. TAA algorithms merge the pixels of the previous frame with the pixels of the current frame, which bias the result. In contrast, our temporal algorithm improves the current frame by permuting the seeds using the information of the previous frame, which leads to unbiased results.

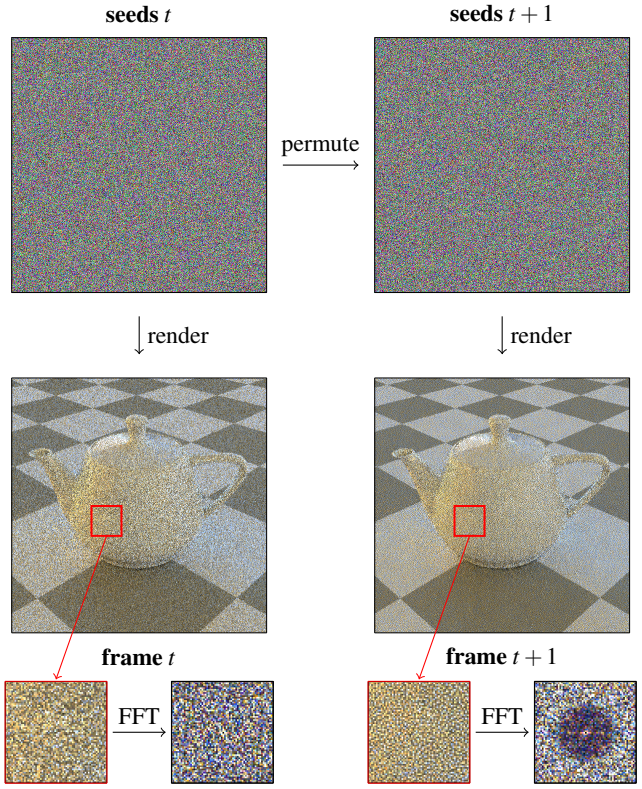

Figure 2: Permuting pixels between frames. *In our rendering architecture, each pixel uses a sequence of random numbers determined by a seed to estimate the Monte Carlo rendering integral. In this example, we start with frame t whose seeds are randomly distributed in screen-space. The resulting errors are distributed as a white noise (the spectrum is statistically flat in a small neighborhood). To obtain the seeds of frame $t+1$, we apply local permutations that correlate the pixel values. Thanks to this, the errors in the next frame are distributed as a blue noise (the spectrum has no low-frequencies in a small neighborhood). As a result, the frame appears less noisy despite the errors are statistically the same.*

Blue-noise properties. The problem addressed in this paper should not be confused with the problem of generating sample sequences following a blue-noise spectrum in sample space (see [Fat11, APC*16] as examples). Our objective is not to construct sample sequences but rather to find how to assign a sample sequence to each pixel such that the Monte Carlo errors that they produce on the rendering integrals have a blue spectrum in screen space. Note that our method does not make any assumption on the sequences used by the renderer as long as they can be represented by a seed.

Screen-space errors randomization. Our works builds upon the concept of randomizing or scrambling a sampling sequence to break structural artifacts that would appear if the same sequence was used across pixels. This technique is common in rendering [KK02, GRK12, Owe98]. However, using a pure random scram-

bling typically leads to a white-noise distribution of the errors that is perceptually unpleasing and produces low frequency artifacts after denoising [SPD18]. The goal of this paper is to control the spectral properties of the rendering errors in order to distribute them in screen-space as a blue noise rather than a white noise, which improves the visual fidelity.

2.1. Recap on Blue-Noise Dithered Sampling

We propose a more in-depth exposition of Blue-Noise Dithered Sampling (BNDS) [GF16] and gather several empirical observations practitioners made regarding its efficiency. These observations provide useful insights to understand the effectiveness of our *a posteriori* formulation of the rendering operation introduced in the next section.

Per-pixel Monte Carlo integration. We consider a rendering formulation where the value of each pixel (i,j) is the integral of a D-dimensional rendering integrand f_{ij}:

$$R_{ij} = \int_{[0,1]^D} f_{ij}(\mathbf{x})\, \mathrm{d}\mathbf{x}. \tag{1}$$

Each pixel (i,j) uses a sequence $\left(\mathbf{s}_{1,ij}, ..., \mathbf{s}_{N,ij}\right)$ of D-dimensional points in the unit hypercube to compute a Monte Carlo estimate of the integral:

$$I_{ij} = \frac{1}{N}\sum_{n=1}^{N} f_{ij}\left(\mathbf{s}_{n,ij}\right) \approx R_{ij}. \tag{2}$$

Blue-noise dithered sampling. Following the idea of using dithering tiles to produce halftone images, BNDS correlates the per-pixel sequences using a dithering tile [GF16]. All the pixels use the same sequence $(\mathbf{s}_1, .., \mathbf{s}_N)$ toroidally shifted by the D-dimensional vectors $\mathbf{d}_{ij}$ of a dithering tile, such that the sequence of pixel (i,j) is

$$\mathbf{s}_{n,ij} = \mathrm{mod}\left(\mathbf{s}_n + \mathbf{d}_{ij}, 1\right). \tag{3}$$

Since the vectors $\mathbf{d}_{ij}$ from the dithering tile are correlated in screen-space, the $\mathbf{s}_{n,ij}$ become correlated in screen-space as well. The assumption is that these correlations remain after the application of the integrands f_{ij}. This is why BNDS is an *a priori* method: the screen-space correlations are optimized regardless of the integrands and hopefully remain once the integrands are applied. This works only if certain conditions are met.

Optimal condition 1: single sample. The more samples from the sequence are used, the more the screen-space correlation vanishes [GF16]. Hence, BNDS works best when a single sample is used, which is equivalent to using the values stored in the dithering tile directly as a single sample value, i.e. when Equation (2) becomes

$$I_{ij} = f_{ij}\left(\mathbf{s}_{ij}\right). \tag{4}$$

Optimal condition 2: low dimensionality. Producing high-dimensional blue-noise dithering tiles of good quality remains an open problem [GF16, Pet17, Wol18]. The higher the dimension of the samples $\mathbf{s}_{ij}$, the less they can be correlated in each dimension. Hence, BNDS works better on low-dimensional integrands and optimally in 1D.

Optimal condition 3: correlation-preserving integrands. BNDS works better with simple integrands that preserve well the correlation of the samples [GF16]. Indeed, the blue-noise spectrum of a dithering tile reflects the screen-space correlation of its values. For instance, if two pixels (i,j) and (k,l) are in the same neighborhood in a dithering tile, then their respective values d_{ij} and d_{kl} are (anti-)correlated. As a consequence, the more $f_{ij}\left(\mathbf{s}_{ij}\right)$ correlates with $\mathbf{s}_{ij}$, and the more $f_{kl}\left(\mathbf{s}_{kl}\right)$ correlates with $\mathbf{s}_{kl}$, the more $f_{ij}\left(\mathbf{s}_{ij}\right)$ correlates with $f_{kl}\left(\mathbf{s}_{kl}\right)$:

$$\begin{array}{ccc} d_{ij} & \overset{\text{corr}}{\longleftrightarrow} & d_{kl} \\ \Big\updownarrow \text{corr} & & \Big\updownarrow \text{corr} \\ f_{ij}\left(d_{ij}\right) & \overset{\text{corr}}{\longleftrightarrow} & f_{kl}\left(d_{kl}\right) \end{array}$$

This is why the more integrands (anti-)correlate their values to their argument, the more the screen-space correlations and the blue shape of the spectrum are preserved. This is shown in the experiment of Figure 3.

Optimal condition 4: screen-space coherence. Finally, a classic condition of any dithering algorithm is that the integrands do not vary too much in screen-space. The closer pixels (i,j) and (k,l) are, the closer their respective integrands f_{ij} and f_{kl} should be. In the examples of Figure 3, we used integrands that are constant in screen space ($f_{ij} = f_{kl}$), which is the optimal case. In practice, this means that the blue-noise distribution of the errors works optimally in the screen-space neighborhoods where the scene does not vary too much.

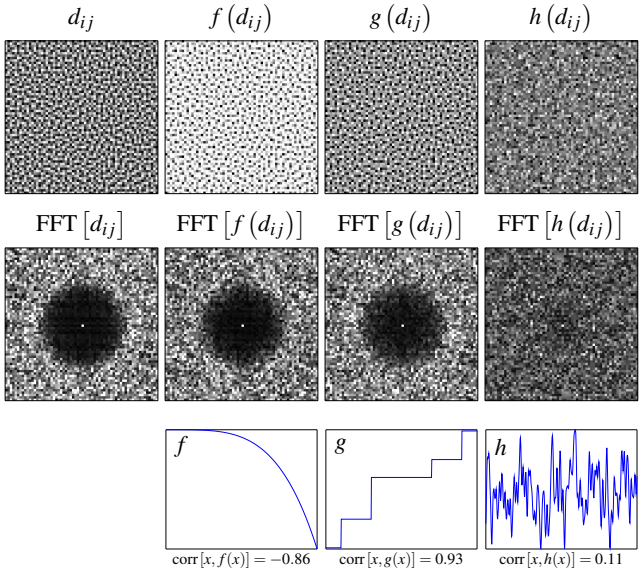

Figure 3: Correlation-preserving integrands. *In this experiment, we consider a 1D blue-noise dithering tile d_{ij} (typically used for halftoning) and we observe what happens when various integrands are applied. We notice that the blue spectrum of the dithering tile is preserved by integrands that (anti-)correlate their values to their arguments.*

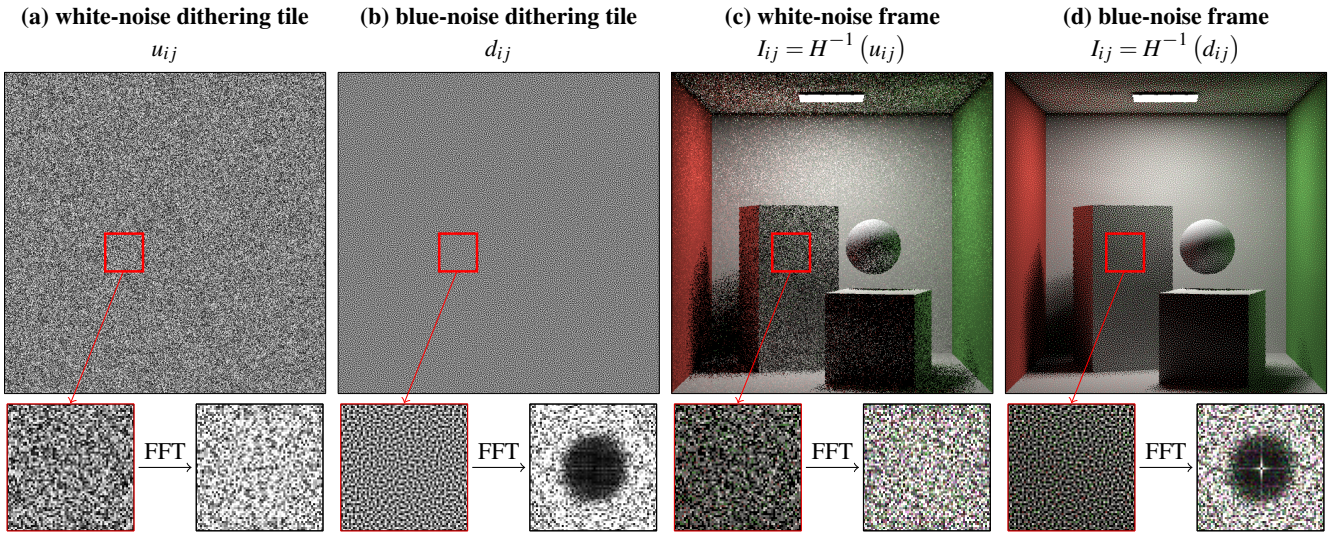

Figure 4: A posteriori formulation of the rendering operation. *To compute these frames, we start by rendering many frames with different random seeds in order to obtain an histogram of estimates for each pixel, as in Figure 5. Then, we sample the histogram of each pixel using random numbers provided by a dithering tile. This formulation efficiently transfers the screen-space correlations of the dithering tile to the frame regardless of the sample count used to compute the Monte Carlo estimates or the dimensionality of the rendering integral.*

3. Theory: A Posteriori Formulation of Rendering

In this section, we introduce an *a posteriori* formulation for distributing the errors as a blue-noise in screen space for a given scene, whatever the sample count and the dimensionality of the integral. This formulation cannot be used directly because it is prohibitively costly for practical usage. However, it provides several insights that the temporal algorithm introduced in the next section is based on.

The histogram of estimates. Remember that each pixel (i, j) uses a random seed to obtain the sequence used to compute a Monte Carlo estimate I_{ij} of the rendering integral inside this pixel. In Figure 5, we consider the histogram of all the Monte Carlo estimates I_{ij} given by all the possible seeds for this pixel. This *histogram of estimates* is a probability density function (PDF) h_{ij} over the space of the pixel estimates.

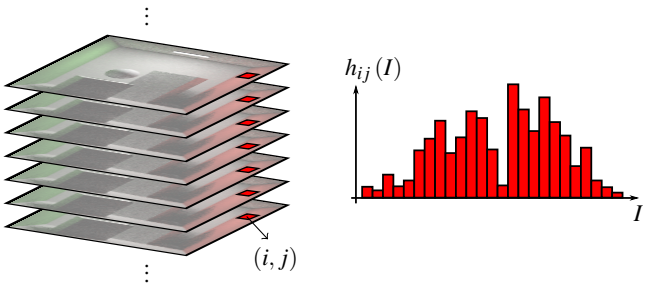

Figure 5: The histogram of estimates of a pixel. *In this experiment, we render the scene many times with different random seeds such that the Monte Carlo errors are different in each frame. Hence, the same pixel has a different value in each frame. The statistical representation of all the possible values of this pixel is what we call the histogram of estimates.*

Sampling the histogram of estimates. In each pixel, the rendering operation means choosing a random seed and computing an estimate using this seed. Hence, in pixel (i, j) the rendering operation can be seen as a generator of random variates I_{ij} following distribution h_{ij}. Reciprocally, sampling random variates I_{ij} from distribution h_{ij} is equivalent to rendering pixel (i, j). For this purpose, we use inverse-transform sampling, i.e. we map a uniform random number $u \in [0, 1]$ to a random I_{ij} from distribution h_{ij} with the inverse Cumulative Distribution Function (iCDF):

$$I_{ij} = H_{ij}^{-1}(u). \tag{5}$$

Classic (uncorrelated) rendering formulation. In Figure 4-(a), we show uncorrelated random numbers u_{ij} distributed as a white noise in screen space. We use them to sample the histogram of estimates

$$I_{ij} = H_{ij}^{-1}(u_{ij}), \tag{6}$$

and we obtain the frame shown in Figure 4-(c). The resulting errors are distributed as a white noise in screen-space. Note that this frame is mathematically equivalent to a classically rendered frame where the seeds are chosen randomly.

Dithered (correlated) rendering formulation. In Figure 4-(b), we show a dithering tile, i.e. correlated random numbers d_{ij} distributed as a blue noise in screen space. We use them to sample the histogram of estimates

$$I_{ij} = H_{ij}^{-1}(d_{ij}). \tag{7}$$

and we obtain the frame shown in Figure 4-(d). Note that the resulting errors are distributed as a high-quality blue noise in screen-space, which is expected from this formulation. Indeed, another interpretation of this equation is that it is using BNDS on the inverse CDF H_{ij}^{-1}, which satisfies three of the optimal conditions reviewed in Section 2.1:

Optimal condition 1: We use the dithering tile values d_{ij} directly as *single samples*. Equation (7) is similar to Equation (4) where function H_{ij}^{-1} plays the role of function f_{ij}.

Optimal condition 2: The values d_{ij} of the dithering tile are 1-*dimensional values*, which is the optimal dimensionality for blue-noise dithered sampling.

Optimal condition 3: Function H_{ij}^{-1} is a *monotonic function* because it is the inverse of a CDF. Monotonic functions usually preserves well the correlations of the samples d_{ij} as illustrated in the example of the third column of Figure 3.

A posteriori formulation of the rendering operation. Equation (7) is an *a posteriori* formulation of the rendering operation that produces a frame with a high-quality blue-noise distribution of the errors in the screen-space neighborhood where the scene does not vary too much (optimal condition 4). The strength of this formulation is that it uses BNDS in the space of the histogram where its optimal conditions are met regardless of the sample count and the dimensionality of the rendering integral.

Practical limitation. Unfortunately, the formulation is not practical as is because the computation of the histograms is prohibitively costly: we need to render many frames to obtain the per-pixel histograms. For the same rendering time one would rather render a single frame with more samples and get a noise-free result. The key issue to address to make this formulation practical is thus the evaluation of the histograms. This is the focus of the next section.

4. Practice: Temporal Algorithm

In this section, we introduce a temporal algorithm based on the *a posteriori* formulation introduced in the previous section. Our idea is to obtain a cheap approximation of the histograms by gathering the estimates of the neighboring pixels in the previous frame. We thus make the assumption that neighboring pixels have similar values in consecutive frames and the algorithm achieves high-quality blue-noise error distributions in low-varying screen-space regions under coherent motion. When this assumption is violated (at object edges, under incoherent motion, etc.) the errors are distributed as a white noise equivalent to a classic decorrelated randomization.

4.1. Overview of our Temporal Algorithm

In practice, our temporal algorithm applies permutations to the pixel seeds between two frames such that the errors become distributed as a target blue-noise dithering tile. Note that the target dithering tile changes after each frame, so that each pixel has a different error in each frame. This is important for temporal filtering algorithms that reduce the errors by averaging them over multiple frames. Our pipeline shown in Figure 6 is divided in two passes. First, the *sorting pass* (Sec. 4.2) approximates the *a posteriori* formulation, i.e. the computation of the histogram of estimates and its dithering, over blocks of pixels. Thanks to this, the distribution of the seeds improves with respect to the target dithering tile. However, the sorting pass remains approximate. Furthermore, the target changes after each frame and the sorting pass alone never converges towards the target. This is why we add a *retargeting pass* (Sec. 4.3) that permutes seeds distributed as the target dithering tile into seeds distributed as the target dithering tile of the next frame. Thanks to this, the improvements gathered by the sorting pass in the current frame are transferred to the next frame and they accumulate frame after frame. It is the combination of sorting (improving) and retargeting (accumulating) that achieves a high-quality blue-noise distribution after a few frames.

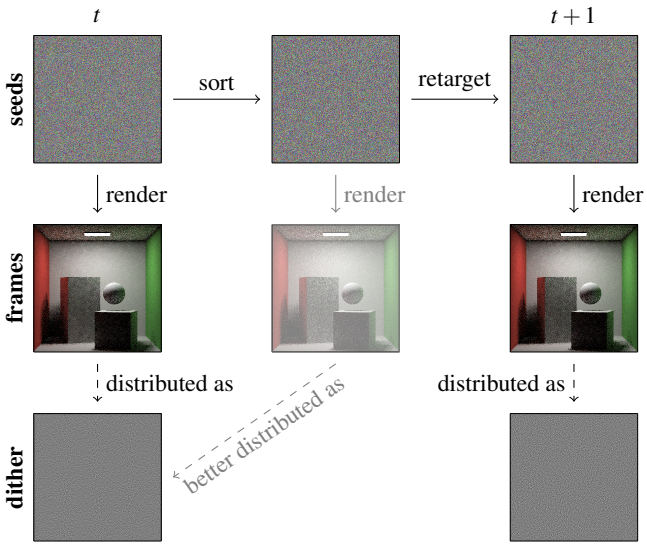

Figure 6: Our temporal algorithm. *Our algorithm permutes the seeds such that the errors of a frame are distributed as a target dithering tile that is different for each frame. The purpose of the* sorting pass *is to improve the seed distribution and the purpose of the* retargeting pass *is to accumulate these improvements frame after frame. Note that our algorithm operates only on the seeds used to compute the pixel values. The faded frame is represented only conceptually, it is not computed explicitely.*

4.2. Sorting Pass: Improving the Error Distribution

The sorting pass directly follows the dithering of the histogram of estimates in Equation (7). First, note that a classic sampling property is that using inverse-transform sampling on a set of discrete 1D data is equivalent to sampling a list of the sorted data. Indeed, inverse-transform sampling maps random numbers to quantiles of the distribution and the quantiles of a discrete distribution are its sorted elements. Hence, with a discrete set of estimates I_{ij} for pixel (i, j), inverse-transform sampling one estimate can be done by sorting the set and choosing a random index, as shown in Figure 7.

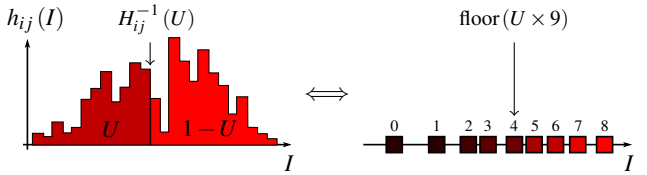

Figure 7: Inverse-transform sampling discrete data. *Applying inverse-transform sampling on a set of discrete data is equivalent to sorting the data and sampling an index in the sorted list.*

Processing blocks of pixels. Sampling the histogram of estimates of a pixel can thus be approximated by collecting estimates from the neighboring pixels, sorting them, and sampling the sorted list. Constructing a sorted list of neighboring pixels for each pixel is costly. The idea of Algorithm 1 is to amortize the cost by sharing the sorting computations over blocks of $B \times B$ pixels. Figure 8 illustrates how the sorting pass works over such a block of pixels. We construct two sorted list of B^2 elements: one for the pixels of the frame (sorted by intensities) and one for the grayscale pixel values of the dithering tile. The sorting order provides a mapping between the pixels of the frame and the pixels of the dithering tile. They are mapped with respect to the quantiles that they represent in their respective blocks, which is equivalent to inverse-transform sampling, as explained above. Hence, mapping by sorting orders effectively implements Equation (7) with the approximation that the histogram is computed using a finite number of neighboring pixels. This mapping yields the permutation that we apply on the pixel seeds. In Figure 8, we show that the mapping yields a permutation that would distribute the frame values as the dithering tile if we applied it on the frame. In practice, we apply it only on the seeds.

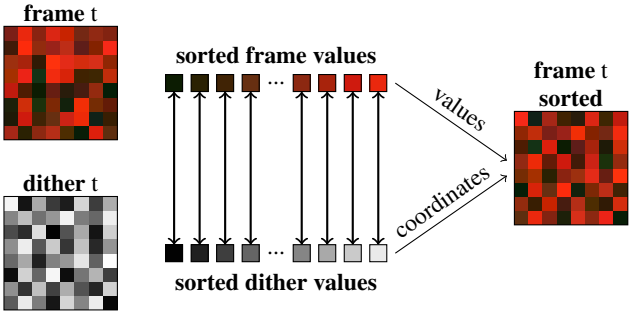

Figure 8: Illustration of the sorting pass. *In the sorting pass, we divide the frame in blocks of size $B \times B$ ($B = 8$ in this example). For each block, we sort the B^2 pixels from the frame by intensities and the B^2 pixels from the dithering tile by their grayscale values. The mapping provided by the sorted lists is a permutation. Virtually, if we applied this permutation on the frame, it would produce an image with the same values as the frame but the same screen-space correlations as the dithering tile. In Algorithm 1, we apply this permutation to the seeds.*

Algorithm 1 The sorting pass permutes pixel seeds by blocks.

▷ Create lists of Pixel(float value, int i, int j)
for each (i, j) in block **do**
  F.add(Pixel(intensity(**frame**(i, j)), i, j)) ▷ frame pixel
  D.add(Pixel(**dither**(i, j), i, j)) ▷ dither pixel
end for
▷ Sort lists by pixel values
sort(F)
sort(D)
▷ Permute seeds
for each n in 1..size(F) **do**
  seeds_sorted(D(n).i, D(n).j) = **seeds**(F(n).i, F(n).j)
end for

Advantages of processing by blocks. This algorithm has three good properties. First, the cost of the construction of the histogram (the sorting operation) is amortized over the block of pixels. Second, the mapping with the sorted dither values guarantees that each seed has a unique destination, i.e. we obtain a true (bijective) permutation and the seeds are never duplicated. Finally, processing by blocks makes the algorithm easily parallelizable.

Problem of processing by blocks. The size of the blocks in the sorting pass is a user-defined parameter B. In practice, we recommend using a block size B between 2 and 8 but there is no ideal value for B. The size of the blocks is a matter of tradeoff between two approximations: the spatio-temporal locality and the histogram discretization. If the blocks are too large, the locality assumption is violated and the permuted seeds do not produce similar estimates in the next frame, which results in a poor blue-noise distribution of the errors. On the other side, if the blocks are too small, the histogram does not have enough entries to be accurate and the blue-noise distribution does not improve much after a frame. In order to overcome this problem, we designed the retargeting pass (Section 4.3) that accumulates the improvements over time such that a small block size can be used and still reaching a decent quality after a few frames.

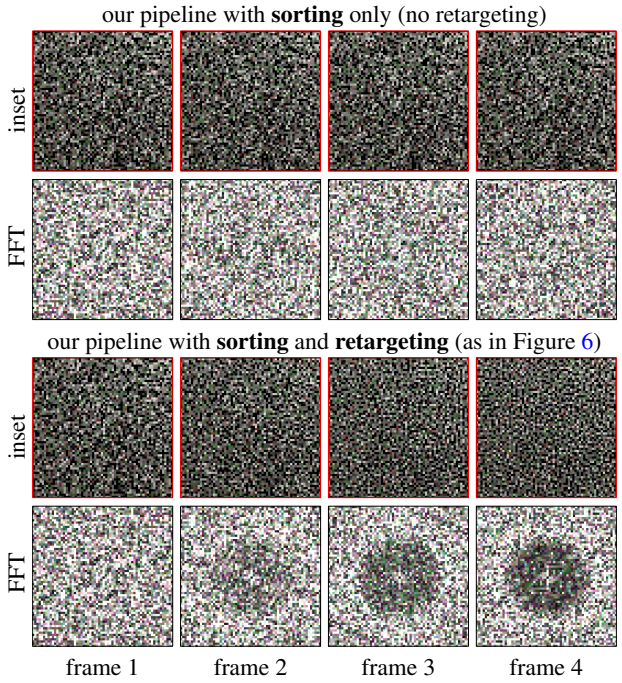

Figure 9: Accumulating improvements over frames thanks to the retargeting pass. *We study the evolution of the inset of Figure 4 through several frames. In this example, we use a block size $B = 2$ for the sorting pass, which is too small to provide significant improvements in a single frame. (Top) Without the retargeting pass, the improvements of the sorting pass cannot accumulate frame after frame. The errors remain distributed as a white noise. (Bottom) Thanks to the retargeting pass, the improvements accumulate and the errors become distributed as a blue noise after a few frames.*

4.3. Retargeting Pass: Accumulating Improvements

The motivation for the retargeting pass is illustrated in the experiments of Figure 9. In the first experiment (top rows), the sorting pass uses a block size $B = 2$, which means that the histogram of estimates is discretized over only 4 entries. This approximation of the histogram is too coarse and the blue-noise distribution does not improve much after a frame. Because the target dithering tile changes after each frame, the improvements do not accumulate and the errors remain distributed as a white noise. In the second experiment (bottom rows), we use the retargeting pass in addition to the sorting pass. The improvements accumulate and the errors become effectively distributed as a blue noise after a few frames. Thanks to the retargeting pass, we can thus use a small block size for the sorting pass (we need less screen-space coherence) and accumulate the improvements over multiple frame (the coarse histogram discretization is virtually amplified over multiple frames).

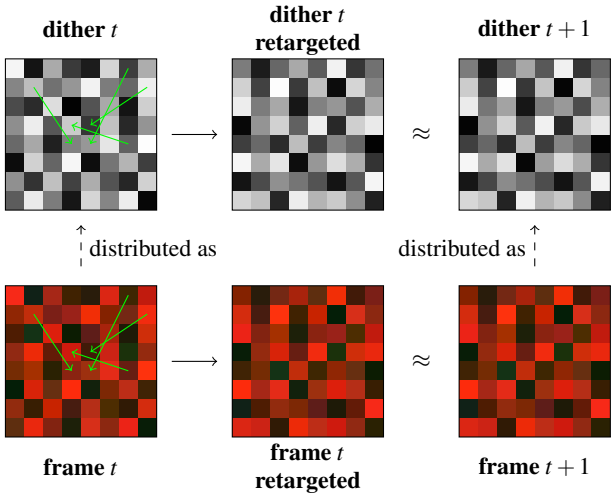

Figure 10: Illustration of the retargeting pass. *The retargeting pass uses a precomputed permutation represented by the green arrows. This permutation has been optimized to permute the dithering tile of frame t into the one of frame $t+1$. Virtually, if we applied this permutation on frame t that is distributed like dithering tile t, it would produce an image with the same values as frame t but the same screen-space distribution as dithering tile $t+1$. In Algorithm 2, we apply this permutation to the seeds.*

Computation of the retargeting pass. As shown in Figure 10, our idea is to use a precomputed permutation that transforms the dithering tile t into the dithering tile $t+1$ and apply this permutation to the seeds before rendering frame $t+1$. Hence, if the seeds used to compute frame t are well distributed as dithering tile t, then the retargeted seeds will already be distributed as dithering tile $t+1$. Thanks to this, we improve the seeds with respect to dithering tile $t+1$ twice: by retargeting the seeds optimized for frame t (before rendering frame $t+1$) and in the sorting pass of frame $t+1$ (after rendering frame $t+1$).

Precomputation of the retargeting permutation. We precompute the retargeting permutation in an offline process using an optimization similar to the algorithm of Georgiev and Fajardo [GF16]. We start with dithering tile t and apply random permutations via simulated annealing to minimize the difference with dithering tile $t+1$. In order to obtain local permutations, we only accept permutations within a small radius. In our implementation, we fix this radius to 6 pixels. Finally, we store this permutation in a 2-channel image in which each pixel (i,j) stores its retargeted coordinates (k,l). Algorithm 2 shows that implementing the retargeting pass is no more than fetching this precomputed permutation and applying it to the seeds.

Algorithm 2 The retargeting pass permutes pixel seeds.

▷ Fetch precomputed retargeting tile
1: $(k,l) = \textbf{retarget}(i,j)$
▷ Permute seeds
2: $\textbf{seeds\_retargeted}(k,l) = \textbf{seeds}(i,j)$

4.4. Representation of the precomputed data

Our algorithm uses two precomputed data for each frame: the dithering tile and the retargeting tile. In theory, these tiles are different for each frame. In practice, however, we always use the same tiles with a constant offset applied after each frame:

$$\textbf{dither}_t(i,j) = \textbf{dither}_0(i+at, j+bt), \tag{8}$$

$$\textbf{retarget}_t(i,j) = \textbf{retarget}_0(i+at, j+bt) + (at, bt), \tag{9}$$

where $\textbf{dither}_0$ is a dithering tile computed with the void-and-cluster method [Uli93] and $\textbf{retarget}_0$ is the permutation that transforms $\textbf{dither}_0$ into $\textbf{dither}_1$ computed as explained in Section 4.3. Note that the offsets are always applied toroidally (modulo the size of the tiles). In order to minimize the perception of the dithering pattern through the animation as tearing or repetitive structure, we used tiles that have the same resolution as the frame and we offset them with Robert's generalized golden ratio [Rob18].

5. Results

Our implementation runs as a loop involving the Mitsuba path tracer [Jak10] for the rendering pass and our algorithms for the sorting and the shuffling passes.

Memory. We store the seed tile as an image of 32-bit unsigned integers at the resolution of the rendering frame, the dither tile as a grayscale image, and the shuffle tile as two images of 32-bit signed integers (one image for the vertical offset and one image for the horizontal offset). Hence, for 1024×1024 renders, we store a 4.2MB seed tile, a 1.6MB dithering tile and a 8.4MB retargeting tile. Since we reuse the same dither and offset tiles across frames, only the seeds need to be recomputed per frame. Note that our storage is highly unoptimized and uncompressed.

Performance. In the following table, we aggregate the timings of the different passes of our algorithm to render an image of the animation for CORNELL BOX (Figure 13), HETVOL (Figure 14), and SSSDRAGON (Figure 12). Note that our CPU implementation is single-threaded but could be trivially parallelized.

scene	sorting + shuffling	rendering (Mitsuba)
CORNELL BOX (32spp, 1024×1024)	0.5s	8.6s
HETVOL (16spp, 1024×1024)	0.5s	13s
SSSDRAGON (32spp, 1280×720)	0.46s	84s

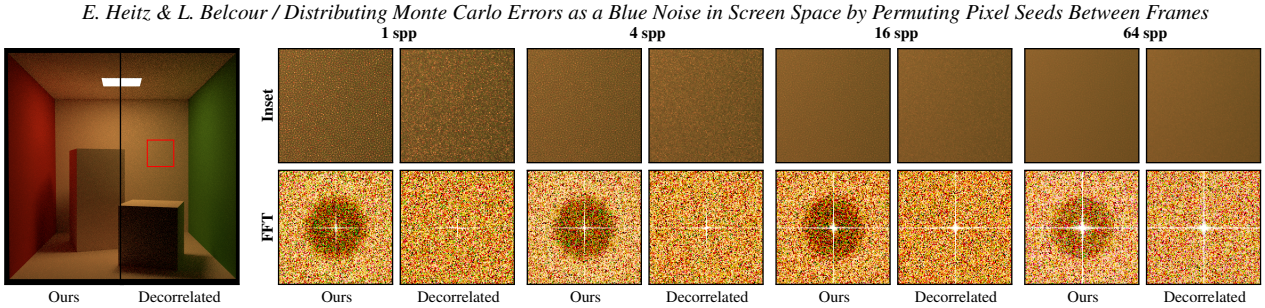

Figure 11: Scaling in sample count. *Our method correctly distributes the rendering noise whatever the number of samples per pixel. Despite the noise being less visible at 64 spp the improvement is still significant compared to a classic decorrelated randomization.*

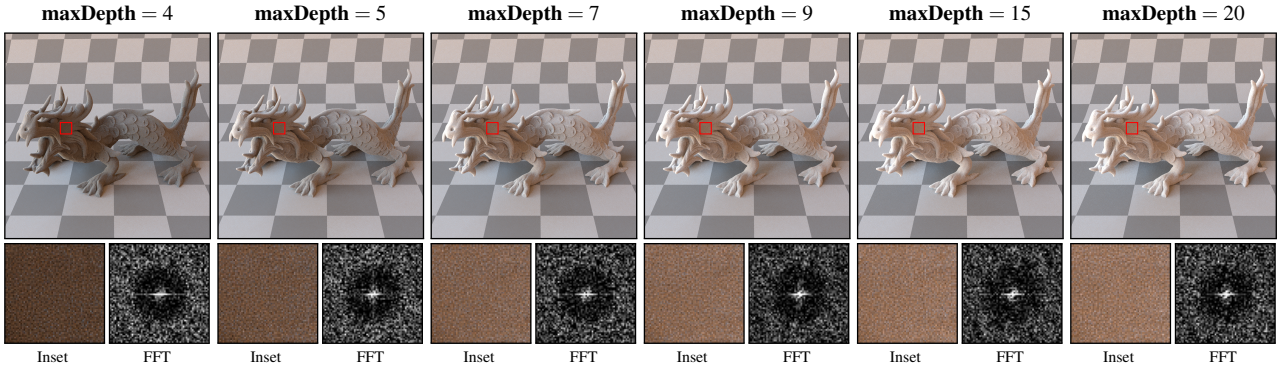

Figure 12: Scaling in dimensionality. *The dragon is a participating medium rendered with Mitsuba's* `volpath` *integrator. Our method still achieves a blue-noise distribution of the errors as we increase the maximum allowed path length.*

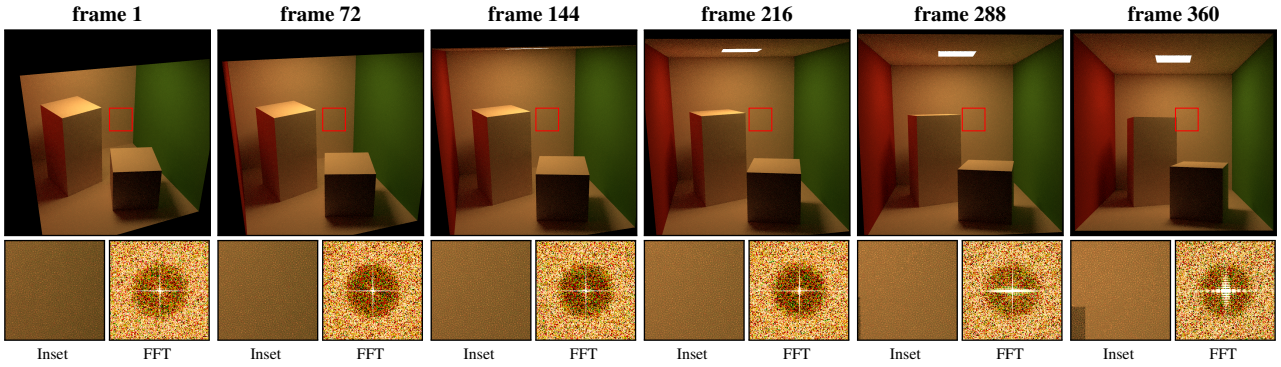

Figure 13: Viewpoint animation. *In this example, we run our method with a camera motion scene with a low sampling count of* 4 *spp and indirect illumination (see bottom insets). Despite the motion, our method still manages to distribute the rendering noise as a blue noise.*

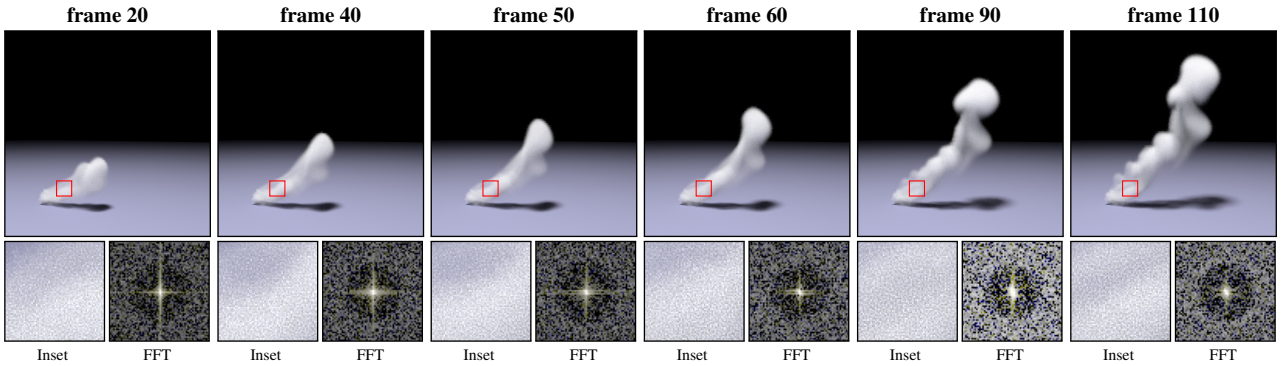

Figure 14: Object animation. *In this example, we run our method on a fluid animation at* 16 *spp. While the content of the smoke data changes and light transport is highly indirect (up to* 16 *bounces), our method still manages to distribute the rendering noise as a blue noise.*

5.1. Result Analysis

Objective 1: scaling in sample count. In Figure 11, we show that our method preserves a blue-noise distribution of the noise as the sampling count increases. This is expected as our method is independent of the sampling count and work *a posteriori* on the pixel intensities. We experienced that we could obtain better blue-noise distribution when the histogram of pixel values in local windows was sufficiently diverse. This is again expected since the ranking strategy will better match pixels to the dither tile values. A consequence is that our method can achieve a better blue-noise distribution with an increasing amount of samples per pixel.

Objective 2: scaling in dimensionality. In Figure 12, we show that our method achieves a blue-noise distribution of the errors with high-dimensional light transports such as path tracing in highly diffusing participating medium (the dragon). We test this property by increasing the maximum allowed path length for a volumetric path tracer (Mitsuba's `volpath`) from 4 to 20.

Robustness to animation. In Figure 13, we test our method with camera motion and in Figure 14 with animation. One of the assumption of our method is the spatio-temporal screen-space coherence of the rendering integrals (condition 4 of Section 2.1). We observe that our method is robust to smooth variations but moving edges are problematic. When the screen-space coherence assumption is violated, the sorting pass cannot correctly distribute the pixels with respect to the dither tile and our method produces a white-noise spectrum around the edge. However, note that even in this case, our method does not impede rendering convergence and the error becomes distributed as a white noise, i.e. it is equivalent to a decorrelated randomization (but not worse).

Limitation: high-frequency variations. In Figure 15, we test our method in a scene that has a texture with high-frequency variations. Because of the texture variations the condition 4 of Section 2.1 is violated and our method cannot correctly distribute the seeds to achieve a blue-noise spectrum. Again, note that in this case our method does not impede rendering convergence and the error becomes distributed as a white noise, i.e. it is equivalent to a decorrelated randomization (but not worse).

Side effect: improving denoising. As noted by Schied [Sch19], distributing rendering errors as a blue noise achieves better denoised results. Indeed, denoising typically reduces the high-frequency part of the errors and denoising a white-noise error usually leaves some low-frequency artifacts. In contrast, errors distributed as a blue noise have no low-frequency content and denoising removes most of their high-frequency spectral content. Our supplemental material provides denoised results that confirm this.

Comparisons against BNDS. Our supplemental material shows that BNDS is superior to our method when its optimal conditions are met: low-dimensional rendering (typically direct lighting) at 1 spp. With other sample counts or non-direct lighting, BNDS becomes equivalent to the classic decorrelated randomization and our method always achieves better results. BNDS and our algorithm provide complementary solutions to different rendering conditions.

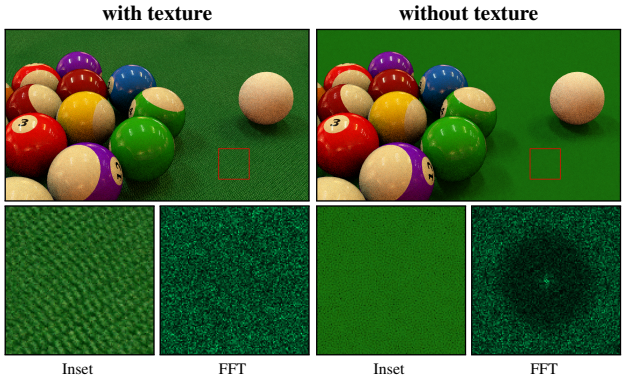

Figure 15: Limitation: high-frequency variations. *When high-frequency textures dominate the integrand, our algorithm fails to produce a blue-noise distribution of the errors. We obtain a white-noise distribution as with a classic decorrelated randomization.*

6. Conclusion and Future Work

We have shown that *a posteriori* methods can tackle the problem of distributing Monte Carlo errors as a blue noise in screen space without being subject to the limitations of *a priori* methods in terms of sample count and dimensionality. To support this claim, we introduced another formulation of the rendering operation and a temporal algorithm that approximates it. Our method produces promising results and we believe that it could already be considered for offline production rendering. In the best cases it significantly improves the results and in the worst cases it becomes equivalent to a classic decorrelated randomization. Furthermore, its execution time is negligible in comparison to the rendering time. Hence, it provides substantial potential benefits without actual drawbacks. Still, there is space for improvements.

Improvement: permuting similar pixels. One assumption of our method is that pixels that are in the same neighborhood have similar values (condition 4 of Section 2.1), which is not always true. Our permutations might move the seeds to pixels that have very different integrands, for instance across the edge of an object. One way to improve this would be to permute seeds only between similar pixels. The similarities between pixels could be obtained for instance by the filters computed by a denoiser.

Improvement: texturing as a post-process. In Figure 15, we have seen that high-frequency textures violate our spatial similarity assumption, which result in a white-noise distribution of the errors. To overcome this limitation, we could try to borrow a trick used by many real-time denoising techniques for diffuse objects: performing our algorithm on the incident irradiance and applying albedo textures as a post-process [SKW*17]. In Figure 16, we apply this trick on the example of Figure 15 and we successfully obtain a blue-noise distribution of the errors despite the high-frequency albedo texture.

Improvement: temporal similarity. Another assumption of our method is that pixels have similar values across frames. This assumption is true only if the screen-space positions of the objects do not vary too much between frames. To improve this approxi-

mation, TAA algorithms typically use temporal reprojection that better match the pixels of the previous frame to the pixels of the current frame. One possible improvement for our method would be a temporal reprojection pass of the seeds. Note, however, that this temporal reprojection should be implemented as a (bijective) permutation such that each seed remains unique.

Improvement: RGB dithering. One of the main limitation of our algorithm is that we do not handle color noise (typically the noise produced by a spectral renderer). This is because the sorting pass illustrated in Figure 8 uses the grayscale intensities of the frame. Note that the sorting pass is reminescent of *color-transfer* algorithms: its purpose is indeed to transfer the colors of the frame to the dithering tile. Hence, it is tempting to replace the sorting pass by a color-transfer algorithm that operates on 3D color values. Furthermore, note that the inverse-transform sampling solved by the sorting pass is equivalent to *optimal transport* in 1D and one classic color-transfer algorithms is precisely optimal transport in a 3D color space [MS03]. Hence, the natural generalization of our 1D sorting pass would be a 3D solver that computes the permutation that yields the optimal transport between the RGB pixels of the frame and the pixels of a 3D dithering tile.

Generalization: other a posteriori approaches. The crux of the problem to use our *a posteriori* formulation is the ability to evaluate the histogram of estimates, for which we use a temporal approximation. Is it possible to obtain a fast approximation of the histogram of estimates by other means?

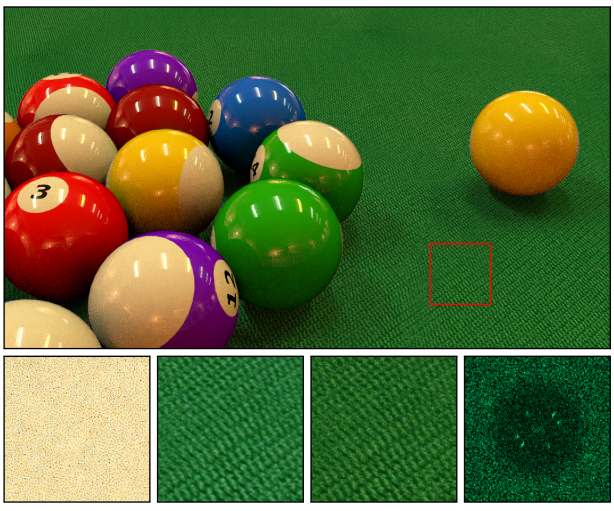

Figure 16: Improvement: texturing as a post-process. *To overcome the limitation of our method to work with textured assets shown in Figure 15, one can apply a classic trick of denoising methods. Here, we render an irradiance image without textures and another image containing the albedo textures and multiply them to obtain the final image (left). By applying our algorithm on the irradiance image without textures, our algorithm manages to distribute the errors as a blue-noise. (right).*

Acknowledgments We thank Kenneth Vanhoey, Jonathan Dupuy, Victor Ostromoukhov, David Coeurjolly and Jean-Claude Iehl for their constructive feedback.

References

[APC*16] AHMED A. G. M., PERRIER H., COEURJOLLY D., OSTROMOUKHOV V., GUO J., YAN D.-M., HUANG H., DEUSSEN O.: Low-discrepancy blue noise sampling. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 247:1–247:13. 2

[Fat11] FATTAL R.: Blue-noise point sampling using kernel density model. In *ACM Transactions on Graphics (TOG)* (2011), vol. 30, ACM, p. 48. 2

[GF16] GEORGIEV I., FAJARDO M.: Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks* (2016), ACM, p. 35. 1, 3, 7

[GIF*18] GEORGIEV I., IZE T., FARNSWORTH M., MONTOYA-VOZMEDIANO R., KING A., LOMMEL B. V., JIMENEZ A., ANSON O., OGAKI S., JOHNSTON E., HERUBEL A., RUSSELL D., SERVANT F., FAJARDO M.: Arnold: A brute-force production path tracer. *ACM Trans. Graph. 37*, 3 (Aug. 2018), 32:1–32:12. 2

[GRK12] GRÜNSCHLOSS L., RAAB M., KELLER A.: Enumerating quasi-monte carlo point sequences in elementary intervals. In *Monte Carlo and Quasi-Monte Carlo Methods 2010*. Springer, 2012, pp. 399–408. 2

[GS16] GJOEL M., SVENDSEN M.: Low complexity, high fidelity: The rendering of INSIDE. Game Developer Conference 2016. 2

[Jak10] JAKOB W.: Mitsuba renderer, 2010. http://www.mitsuba-renderer.org. 7

[KK02] KOLLIG T., KELLER A.: Efficient multidimensional sampling. In *Computer Graphics Forum* (2002), vol. 21, Wiley Online Library, pp. 557–563. 2

[LHR18] LATTA L., HILL S., RADEZTSKY R.: Powering up ILMxLAB's location-based VR experiences. Game Developer Conference 2018. 2

[MA06] MEYER M., ANDERSON J.: Statistical acceleration for animated global illumination. *ACM Trans. Graph. 25*, 3 (July 2006), 1075–1080. 2

[MS03] MOROVIC J., SUN P.-L.: Accurate 3d image colour histogram transformation. *Pattern Recogn. Lett. 24*, 11 (July 2003), 1725–1735. 10

[Owe98] OWEN A. B.: Scrambling sobol'and niederreiter–xing points. *Journal of complexity 14*, 4 (1998), 466–489. 2

[Pet17] PETERS C.: The problem with 3d blue noise, 2017. Blogpost. URL: http://momentsingraphics.de/?p=148. 3

[Rob18] ROBERTS M.: The unreasonable effectiveness of quasirandom sequences, 2018. Blogpost. URL: http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/. 7

[Sch19] SCHIED C.: Real-time path tracing and denoising in Quake 2. Game Developer Conference 2019. 2, 9

[SKW*17] SCHIED C., KAPLANYAN A., WYMAN C., PATNEY A., CHAITANYA C. R. A., BURGESS J., LIU S., DACHSBACHER C., LEFOHN A., SALVI M.: Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics* (2017), ACM, p. 2. 2, 9

[SPD18] SCHIED C., PETERS C., DACHSBACHER C.: Gradient estimation for real-time adaptive temporal filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2 (2018), 24. 2, 3

[Uli93] ULICHNEY R.: The void-and-cluster method for dither array generation. *Proceedings of SPIE - The International Society for Optical Engineering* (09 1993). 2, 7

[Wol18] WOLFE A.: Not all blue noise is created equal, 2018. Blogpost. URL: https://blog.demofox.org/2018/08/12/not-all-blue-noise-is-created-equal/. 3