A Low-Discrepancy Sampler that Distributes Monte Carlo Errors as a Blue Noise in Screen Space

Eric Heitz, Laurent Belcour, Victor Ostromoukhov, David Coeurjolly, Jean-Claude Iehl

A Low-Discrepancy Sampler that Distributes Monte Carlo Errors as a Blue Noise in Screen Space

Eric Heitz
Unity Technologies

Laurent Belcour
Unity Technologies

V. Ostromoukhov
Univ. Lyon, CNRS

David Coeurjolly
Univ. Lyon, CNRS

Jean-Claude Iehl
Univ. Lyon, CNRS

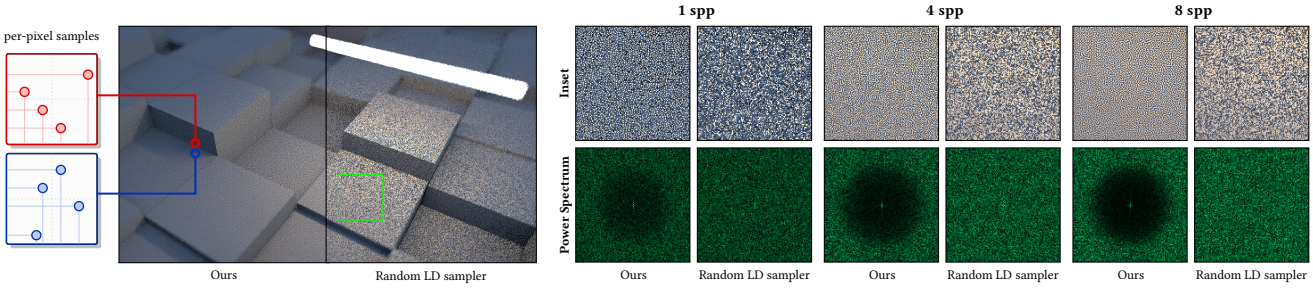

Figure 1: Our sampler distributes per-pixel sample sets such that their Monte Carlo errors is a blue noise in screen space. This increases the visual quality of the renders in contrast to randomly distributing the sample sets.

ABSTRACT

We introduce a sampler that generates per-pixel samples achieving high visual quality thanks to two key properties related to the Monte Carlo errors that it produces. First, the sequence of each pixel is an Owen-scrambled Sobol sequence that has state-of-the-art convergence properties. The Monte Carlo errors have thus low magnitudes. Second, these errors are distributed as a blue noise in screen space. This makes them visually even more acceptable. Our sampler is lightweight and fast. We implement it with a small texture and two xor operations. **Our supplemental material provides comparisons against previous work for different scenes and sample counts.**

ACM Reference Format:
Eric Heitz, Laurent Belcour, Victor Ostromoukhov, David Coeurjolly and Jean-Claude Iehl. 2019. A Low-Discrepancy Sampler that Distributes Monte Carlo Errors as a Blue Noise in Screen Space. In *Proceedings of SIGGRAPH '19 Talks.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3306307.3328191

1 INTRODUCTION

Georgiev and Fajardo pioneered the concept of distributing the error of Monte Carlo rendering as a blue noise in screen space with *dithered sampling* [2016]. Inspired by halftoning algorithms, they optimize a tile whose pixels contain Cranley-Patterson rotations (toroidal shifts) applied on an arbitrary sequence so that it becomes different in each pixel. Equivalently, each pixel of their tile can be seen as the first sample of the sequence of this pixel and is optimized to maximize the sample-space difference between neighboring pixels. This method distributes the errors as a blue noise at one sample per pixel but this advantageous feature vanishes at higher sample counts. Furthermore, toroidal shifts affect the equidistribution of low-discrepancy sequences and hence their convergence rates.

In this paper, we focus on Owen-scambled Sobol sequences [1998] that have state-of-the-art convergence properties. We distribute their Monte Carlo errors as a blue noise in screen space without compromising their convergence properties in the following way:

- In §2.1, we leverage the fact that the points of Sobol sequences can be modified via *scrambling* (their values) and *ranking* (their order) without compromising the convergence rate. We implement these operations with *bitwise xors* with integer keys. We store per-pixel keys as a lightweight representation of per-pixel sequences and use them as degrees of freedom to optimize the error distribution.
- In §2.2, we introduce a new energy term to optimize the blue-noise distribution of the error. In contrast to dithered sampling, we do not optimize a sample-space distance. Our new energy term acts directly on the Monte Carlo errors produced by the sequences. The optimization can thus be rigorously formulated for arbitrary sample counts and arbitrary dimensionalities of the sample space.
- Combining these ideas, we use the scrambling keys to optimize the blue-noise distribution for a target sample count N, typically a high value that the renderer does not exceed. The blue-noise distribution of the error is thus optimal at N samples per pixel. Then, we use the per-pixel ranking keys to optimize for all the sample counts that are powers of two between 1 and $N/2$.

SIGGRAPH '19 Talks, July 28 - August 01, 2019, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6317-4/19/07.
https://doi.org/10.1145/3306307.3328191

2 OUR SAMPLER

Our sampler provides for each pixel (i, j) a D-dimensional sample set of N samples: $P^{ij} = \{p_n^{ij} \ : \ n \text{ in } 1..N\}$ optimized such that the Monte Carlo errors are distributed as a blue noise in screen space for sample counts that are powers of two smaller or equal to N.

2.1 Per-pixel low-discrepancy sample sets

Sample points. We store a unique D-dimensional Owen-scrambled Sobol [1998] sample set $P = \{p_n \ : \ n \text{ in } 1..N\}$.

Scrambling keys. In each pixel (i, j), we store a scrambling key s^{ij} of D integers. We use it to xor the integer representation of the Sobol samples. This preserves their convergence properties [2002].

Ranking keys. In each pixel (i, j), we store a ranking key r^{ij} of one integer. We use it to xor the indices of the points (the order in which they are used). One property of Sobol sample sets is that any power-of-two subset aligned with the same power of two has the same low-discrepancy properties. Hence, xoring their indices does not change the properties of the power-of-two subsets.

Evaluation. Table 1 shows the data and storage requirements of our sampler. The evaluation of the n-th sample of a pixel (i, j) is shown in Algorithm 1. We compute the index m of the sample in the point set P by xoring n with the ranking key r^{ij} and we xor p_m with the scrambling key s^{ij}.

Table 1: Data and storage requirements of our sampler. Following Georgiev and Fajardo we use 128^2 wrappable tiles.

description	symbol	storage
Sample points	$P = (p_1, ..., p_N)$	$N \times D$ integers
Scrambling keys	s^{ij}	$128^2 \times D$ integers
Ranking keys	r^{ij}	$128^2 \times 1$ integers

Algorithm 1 Evaluation of the n-th sample at pixel (i, j)

$m = \text{xor}\left(n, r^{ij}\right)$ // xor the index of the sample
$p_n^{ij} = \text{xor}\left(p_m, s^{ij}\right)$ // xor the value of the sample
return p_n^{ij}

2.2 Distributing the errors as a blue noise

Our objective is to optimize the keys s^{ij} and r^{ij} such that the Monte Carlo errors computed by the sample sets are as different as possible between neighboring pixels.

A family of integrand functions. Our idea is to maximize for the errors computed for a set of functions $(f_1, .., f_T)$ that are representative of typical rendering integrands. We use the space of oriented Heaviside functions f_t that are defined by a $(D-1)$-dimensional normalized direction vector and 1-dimensional phase. We randomly choose a finite set of $T = 65536$ functions from this space. For each pixel (i, j), we compute a vector $E^{ij} = \left(e_1^{ij}, .., e_T^{ij}\right)$ that contains the errors produced by its sample set P^{ij} on the integrands $(f_1, .., f_T)$:

$$e_t^{ij} = \frac{1}{N} \sum_{n=1}^{N} f_t(p_n^{ij}) - \int_{[0,1]^D} f_t(p)\, dp. \tag{1}$$

Optimizing at N samples per pixel using the scrambling keys. The goal is to find the values of the scrambling keys s^{ij} such that the distance between the error vectors of neighboring pixels is maximized. We measure the screen-space proximity of two pixels (i, j) and (k, l) with the Gaussian kernel recommended by Georgiev and Fajardo with $\sigma = 2.1$ and obtain:

$$\mathrm{E_s} = \sum_{(i,j)\neq(k,l)} e^{-\frac{(i-k)^2+(j-l)^2}{\sigma^2}} \left\| E^{ij} - E^{kl} \right\|^2. \tag{2}$$

We initialize each pixel with a unique random scrambling key s^{ij} and swap them to maximize E_s using simulated annealing.

Optimizing below N samples per pixel using the ranking keys. We optimize each power-of-two sample count from N down to 1 by iteratively halving the sample set. Each time we halve, in each pixel we can choose to use either the first or the second half of the sample set. This choice is represented by one bit in the ranking key r^{ij}. For instance, the first bit of r^{ij} swaps $\{p_1^{ij}\}$ and $\{p_2^{ij}\}$, the second bit swaps $\{p_1^{ij}, p_2^{ij}\}$ and $\{p_3^{ij}, p_4^{ij}\}$, etc. For each power-of-two sample count, the goal is to find how to set the associated bit in r^{ij} such that the error vectors E_{first}^{ij} and E_{last}^{ij} obtained with the first and last halves maximize:

$$\mathrm{E_r} = \sum_{(i,j)\neq(k,l)} e^{-\frac{(i-k)^2+(j-l)^2}{\sigma^2}} \left(\left\| E_{\text{first}}^{ij} - E_{\text{first}}^{kl} \right\|^2 + \left\| E_{\text{last}}^{ij} - E_{\text{last}}^{kl} \right\|^2 \right), \tag{3}$$

Note that this energy term maximizes for the first and the last subsets together. Maximizing for the first subset only tends to put high-error subsets first and artificially increases the variance at lower sample counts. Maximizing for the first and last subsets at the same time keeps the errors of both halves to their statistical average. We repeat this operation until the sample count gets down from N to one sample per pixel. After this operation, we obtain per-pixel samples that distribute the error as a blue-noise for any sample count that is a power of two between 1 and N.

3 DISCUSSION

Although the method presented in this paper works reasonably well for a large class of integrands, we observed that it works particularly well with low-dimensional and smooth integrands, typically the direct illumination of an area light. We noticed that optimizing for high dimensions (large D) decreases the quality in lower dimensions. We found the best compromise in optimizing pairs of dimensions separately, i.e. $D = 2$ multiple times. Using our approach has no significant drawbacks. In terms of convergence, each pixel uses an Owen-Scrambled Sobol sequence, which is state-of-the-art. In terms of blue-noise distribution of the error, our method is as good as *dithered sampling* at 1spp and achieves better results at higher sample counts. Finally, our approach is extremely fast using only three memory fetches and two integer xors per sample.

REFERENCES

Iliyan Georgiev and Marcos Fajardo. 2016. Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks*. ACM, 35.

Thomas Kollig and Alexander Keller. 2002. Efficient multidimensional sampling. In *Computer Graphics Forum*, Vol. 21. Wiley Online Library, 557–563.

Art B Owen. 1998. Scrambling Sobol'and Niederreiter–Xing Points. *Journal of complexity* 14, 4 (1998), 466–489.