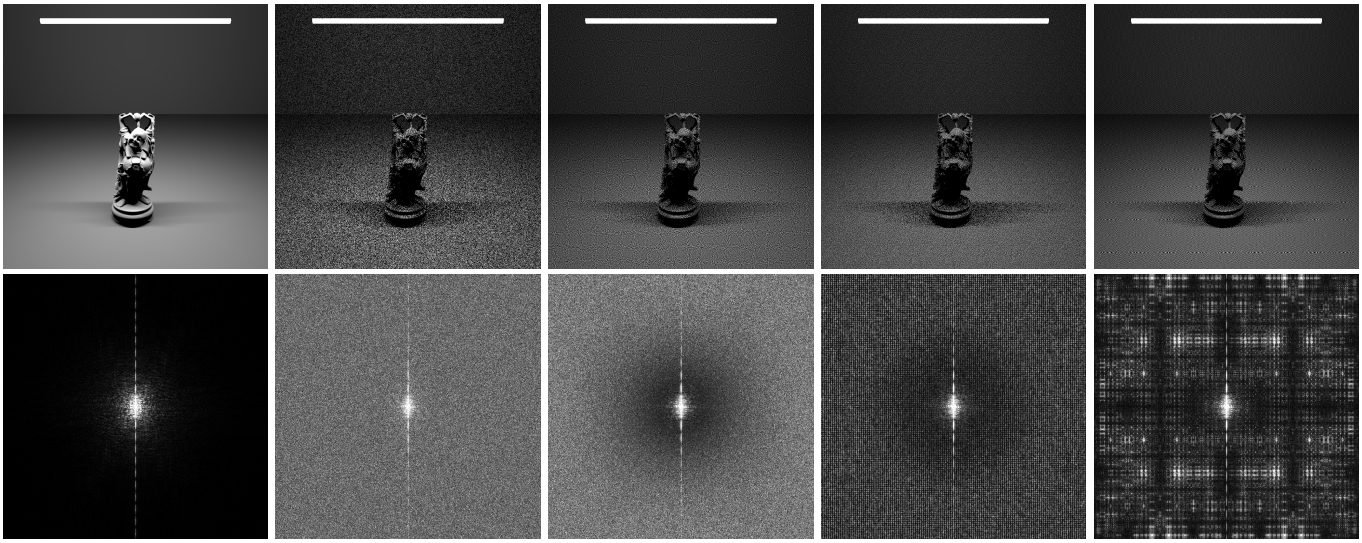
Screen-Space Blue-Noise Diffusion of Monte Carlo Sampling Error via Hierarchical Ordering of Pixels

ABDALLA G. M. AHMED, KAUST, KSA
PETER WONKA, KAUST, KSA



(a) Reference (b) (0, 2)-Sequence (c) Z (ours) (d) Heitz (e) Sobol

Fig. 1. Renderings using one sample per pixel with various samplers showing (top) the rendered images and (bottom) their frequency power spectrum. (a) A reference high-quality rendering using 8k samples per pixel. (b) PBRT's (0, 2)-sequence sampler: each pixel is sampled independently, leading to white noise diffusion of error, as evident in the frequency power spectrum. (c) Z (ours): pixels are ordered in a stochastic locality-preserving ordering, and assigned consecutive samples from a (0, 2) low-discrepancy sequence, hence adjacent pixels receive far-apart samples, leading to a blue-noise diffusion of error, as evident in the frequency power spectrum. (d) Heitz et al. [Heitz et al. 2019]: uses a pre-optimized tile of samples. Note the repetition artifacts at the tile period (1/8 of image width) and the grid-like structure in the spectrum. (e) PBRT's global Sobol sampler: achieves decent error diffusion but in a very systematic way, leading to structured aliasing artifacts, as visible in the image and reflected in the frequency power spectrum. Note that the frequency power spectra in (e) and (d) appear darker not because of reduced aliasing but because of the concentration of aliasing energy in discrete spikes.

We present a novel technique for diffusing Monte Carlo sampling error as a blue noise in screen space. We show that automatic diffusion of sampling error can be achieved by ordering the pixels in a way that preserves locality, such as Morton's Z-ordering, and assigning the samples to the pixels from successive sub-sequences of a single low-discrepancy sequence, thus securing well-distributed samples for each pixel, local neighborhoods, and the whole image. We further show that a blue-noise distribution of the error is attainable by scrambling the Z-ordering to induce isotropy. We present an efficient technique to implement this hierarchical scrambling by defining a context-free grammar that describes infinite self-similar lookup trees. Our concept is scalable to arbitrary image resolutions, sample dimensions, and sample count, and supports progressive and adaptive sampling.

CCS Concepts: • **Computing methodologies** → **Rendering**.

Additional Key Words and Phrases: sampling, error diffusion, dithering, quasi-Monte Carlo, blue noise, low-discrepancy sequences, Sobol sequence, Morton ordering, Z ordering

ACM Reference Format:
Abdalla G. M. Ahmed and Peter Wonka. 2020. Screen-Space Blue-Noise Diffusion of Monte Carlo Sampling Error via Hierarchical Ordering of Pixels. *ACM Trans. Graph.* 39, 6, Article 244 (December 2020), 15 pages. https://doi.org/10.1145/3414685.3417881

Authors' addresses: Abdalla G. M. Ahmed, KAUST, KSA, abdalla_gafar@hotmail.com; Peter Wonka, KAUST, KSA, pwonka@gmail.com.

0730-0301/2020/12-ART244 $15.00
https://doi.org/10.1145/3414685.3417881

1 INTRODUCTION

Sampling is a fundamental process in many categories of computer graphics, including physically-based rendering, halftoning and stippling, geometry discretization, general Monte Carlo integration, and machine learning. We focus on physically-based rendering as the main application in this paper, but present a generic technique.

In physically-based rendering, the most common algorithms make use of Monte Carlo integration. Monte Carlo integration requires a large number of (quasi-)random samples: one high-dimensional sample per light carrying path. To obtain a high quality image it is important that these samples are uniformly distributed.

In this paper, we study the influence of sampling on the distribution of errors in the rendered image. For an individual pixel, Monte Carlo integration may either overestimate or underestimate the pixel color. The question is how the error that is made in one pixel relates to the error made in the neighboring pixels.

As a starting point, we look at positive correlation. If, for example, the same identical sequence of samples is used for all pixels, the error made in one pixel will be similar to the error made by its neighbors. This will cause structured aliasing artifacts and produce very poor results.

Therefore, current state-of-the-art techniques use sampling methods that decorrelate the error between pixels; that is, make the error in one pixel independent of the error made in its neighbors. In terms of sample distribution, the sampler tries to decorrelate the sampling error between the pixels by imposing a randomization process on the samples that still preserve their favorable properties. This can be achieved by various scrambling algorithms. The idea of decorrelated error is commonly associated with the concept of white noise. In Fig. 1(b), for example, we can observe that the Monte Carlo error of samplers such as PBRT's (0, 2)-Sequence [Pharr et al. 2016] manifests as a typical white noise power spectrum, with errors in all frequencies.

Finally, the highest-quality results can be achieved by having negatively correlated error between neighboring pixels. Negatively-correlated errors are associated with the concepts of blue noise and *error diffusion*. It has only recently been spotted that the perceived sampling error may be reduced by exerting some control on the randomization process in such a way that, instead of uncorrelated sets, it produces *negatively correlated* sets of samples for neighboring pixels, so that the sampling error made in one pixel is readily compensated in nearby pixels. Intuitively, if the Monte Carlo integrator makes an error overestimating the true value of one pixel, for example, it would be helpful if the neighboring pixels could compensate for the error by underestimating their true pixel values by a similar amount. In the end, the eye will filter the errors and this will be perceived as a smoother and higher quality image.

The idea of negatively correlating the sample distribution between neighboring pixels was first introduced by Georgiev and Fajardo [2016], who correctly spotted the analogy between halftoning and the sampling error in rendering. They hence introduced an algorithm for rendering that employs a blue-noise dithering mask, akin to the ones used in halftoning [Ulichney 1993], to control the randomization process between adjacent pixels. A recent follow-up by Heitz et al. [2019] presented a similar solution tailored to Sobol sequences. Both works presented impressive initial results, as can be seen by comparing Fig. 1(d) with (b).

The mentioned approaches, however, rely on an offline optimization, which does not seem to scale well for a large number of samples or high-dimensional renderings, as noted by Heitz and Belcour in a followup paper [2019]: "Obtaining such distributions remains an open problem with high sample counts and high-dimensional rendering integrals." For example, the presented implementation of Heitz el al. [2019] recycles the pre-optimised set of samples over the image resolution and higher sample dimensions, leading to multiple types of artifacts, as can be noticed in the rendering and frequency power spectrum of Fig. 1(d).

In this paper we introduce a radically different paradigm to solving the problem of varying the samples between the pixels. We replace the manual offline optimization by a run-time stochastic model that automatically allocates well-distributed samples to neighboring pixels. Instead of coordinating the sample randomization process between nearby pixels, our idea is to generate a grand super-sequence of optimally distributed samples for the whole image, and recursively partition it into sub-sequences of optimally-distributed samples that are assigned to smaller partitions of the image; all the way down to individual pixels. Thus, we diffuse the sampling error not only in the local neighborhood of a pixel, but over hierarchically nested blocks of pixels. Please note that the sequence does not have to be stored; it can be generated as needed using a low-discrepancy sequence like Sobel that maps index numbers to samples. Our approach has many desirable properties. It does not rely on any offline optimization or pre-processing, is very easy to implement, scales smoothly to arbitrary image sizes, sample counts, and sample dimensions, allows for an efficient implementation, and decisively improves upon the state-of-the-art methods.

The paper is organized as follows. In Section 2, we review the generally related literature, and highlight the most related work to ours. In Section 3 we give a theoretical description of our proposed method, and in Section 4 we provide a detailed description of a sampler that implements the concept. We then present actual rendering results in Section 5, and compare different aspects of our sampler to state-of-the-art samplers, before giving concluding remarks in Section 6. We found the discussion of adaptive and progressive sampling too technical and potentially distracting, as it depends on details of the underlying low-discrepancy sequence, so we moved it to Appendix A for the interested readers.

1.1 Naming Convention

To our knowledge, there is still no agreed-on term to describe this relatively new concept of sharing the sampling error between pixels. Georgieve and Fajardo [2016] used "dithering", based on their actual use of a dithering mask. In contrast, we adopt the term "error diffusion" to describe our process. Strictly speaking, error diffusion is not possible in the Monte Carlo integration, since there is no known reference to measure the error from. Nevertheless, we effectively diffuse the anticipated error.

2 RELATED WORK

The idea that regular sampling leads to undesirable aliasing artifacts was noted early in the pioneering works of Dippé and Wold [1985] and Cook [1986], and initiated the research on finding optimal distributions of the samples. Most of the research was devoted to studying the sample points assigned to individual pixels for anti-aliasing or for ray-path construction. Covered areas include: (A) Fast generation, e.g., [Bridson 2007; Dunbar and Humphreys 2006; Ebeida et al.

2014, 2011, 2012; Eldar et al. 1997; Gamito and Maddock 2009; Jones 2006; Kensler 2013; McCool and Fiume 1992; Mitchell et al. 2018]. (B) Optimization, e.g., [Ahmed et al. 2017a; Balzer et al. 2009; Chen et al. 2012; de Goes et al. 2012; Fattal 2011; Heck et al. 2013; Jiang et al. 2015; McCool and Fiume 1992; Mitchell 1991; Öztireli and Gross 2012; Reinert et al. 2016; Schlömer et al. 2011; Zhou et al. 2012]. (C) Tabulation and distribution, e.g., [Ahmed et al. 2015, 2017b; Cohen et al. 2003; Kopf et al. 2006; Lagae and Dutré 2006; Ostromoukhov 2007; Ostromoukhov et al. 2004; Wachtel et al. 2014]. (D) Manipulation, e.g., [Jarosz et al. 2019]. (E) Evaluation, e.g., [Durand 2011; Öztireli 2016; Pilleboue et al. 2015; Schlömer and Deussen 2011; Ulichney 1988]. Most of the cited research, and many other works, focused on two-dimensional sampling point sets with a Poisson-disk property [Dippé and Wold 1985] and a blue-noise frequency power spectrum [Ulichney 1988], which is widely believed to be the ideal distribution; a belief that is rooted in classic signal-processing theory. There is no solid mathematical theory on blue noise, nor a clear model for obtaining it, and this explains the large bulk of literature on it. There are recent attempts, however, to fill this gap, e.g., [Heck et al. 2013; Öztireli 2020]. Our method does not directly use a blue-noise distribution of samples, but leads to a blue-noise distribution of sampling errors, which is the actual end goal of all the mentioned research.

2.1 Low-Discrepancy Sampling

As an alternative to Poisson-disk point sets, Shirely [1991] introduced the idea of considering the *low-discrepancy sequences* commonly used for quasi-Monte Carlo integration in financial and other fields. This idea was greatly advocated by Keller and colleagues, e.g., [Grünschloß et al. 2012; Keller 2013; Kollig and Keller 2002]. Although most of their work is published in specialized Monte Carlo conferences, and is not sufficiently exposed in the graphics research community, low-discrepancy sequences seem to be popular among rendering practitioners. Indeed, PBRT, the celebrated rendering textbook, eventually dropped its Poisson-disk "best-candidate" sampler [Mitchell 1991] in favor of the more efficient low-discrepancy samplers. There were also recent attempts to combine the blue-noise frequency spectrum with the low-discrepancy distribution [Ahmed et al. 2016; Perrier et al. 2018].

Our presented sampling technique is primarily based on low-discrepancy sequences, namely Sobol, although it can also be adapted for other low-discrepancy sampling sequences, e.g., Halton or Weyl, other sample sequences generators, e.g., ART [Ahmed et al. 2017b], and pre-computed sequences of samples, e.g., Progressive Multi-Jittered Sample Sequences [Christensen et al. 2018].

2.2 Error Diffusion in Sampling

The concept of error diffusion [Ulichney 1987] emerged in the halftoning literature long before the advent of physically-based rendering. There is a natural connection of this concept to rendering, since halftoning can be seen as a low-dimensional analog to rendering. Despite this natural connection, the idea of diffusing the (anticipated) Monte Carlo sampling error in screen space was only addressed quite recently. In contrast to the long list mentioned above on generating blue-noise samples, we can only cite three works that are directly related to our work, two published as abstracts [Georgiev and Fajardo 2016; Heitz et al. 2019], and only one full paper [Heitz and Belcour 2019]. Georgiev and Fajardo are credited for demonstrating the possibility of diffusing Monte Carlo noise in an eye-pleasing manner, and for a small sample count they presented convincing results that warrant further research. Heitz and colleagues are credited for demonstrating that low-discrepancy sequences can be optimized to deliver a blue-noise manifestation of the Monte Carlo noise.

The current model for blue-noise error diffusion tries to optimize the randomization process that varies the samples between the pixels in such a way that the samples assigned to each pixel are substantially different from the neighbors. Inspired by the dithering masks used in halftoning [Ulichney 1993], an optimized toroidal tile of samples is obtained as follows. A single set of samples is generated using a sample set/sequence generator, and then a randomization process is applied to it as many times as needed to produce the samples for a large toroidal tile of pixels, typically 128×128. These samples are then distributed over the tile in the desired way: a close distance between pixels corresponds to a large distance between their assigned samples. A stochastic optimization process is employed to reach an optimal layout. Finally, the optimized randomization parameters are stored in a lookup table, and re-applied at run time.

The first variant in this category [Georgiev and Fajardo 2016] used the generic Cranely-Patterson [1976] toroidal-shift randomization that can virtually be used with any sample generation method. A serious limitation is that the distance between the randomized samples is measured by the applied toroidal shift, which is only correct with a single sample per pixel. Indeed, as the number of samples increases, the applied shift may bring a shifted sample close to the original place of another sample; hence, the effect of this approach diminishes with the increased number of samples. The followup by Heitz et al. [2019] tried to address the mentioned limitation by measuring the distance between the samples using their actual sampling error over a selected set of integrated functions, and by computing different optimizations for different sample counts. They also use a more sophisticated randomization technique. The approach works well for low dimensions, but does not scale well with large sample counts and high dimensions [Heitz and Belcour 2019]. We will compare to this method as the current state of the art.

More recently, Heitz and Belcour [2019] extended the idea to motion pictures, taking advantage of temporal correlation to predict a better allocation of randomization seeds used to vary the samples between the pixels. The method, however, is inherently restricted to animated renderings.

While targeting the same problem, our idea was conceived independent of [Georgiev and Fajardo 2016] and [Heitz et al. 2019]. We were inspired by the way that Adaptive Regular Tiles [Ahmed et al. 2017b] distribute the indices of the sample points, and we asked if we can adapt them to distribute optimized indices for a low-discrepancy sampler like Sobol. Our initial idea was optimization-oriented, similar to [Georgiev and Fajardo 2016] and [Heitz et al. 2019], but we ended up with a very different concept that does not use any offline optimization.

2.3 Global Sampling

Since the introduction of distributed ray tracing by Cook [1984], conventional samplers used his idea of sampling the participating domains (lens, lights, etc.) separately, and combining the samples to synthesize the high-dimensional sample vector; as reported, for example, in the earlier editions of PBRT [Pharr and Humphreys 2010, Chapter 7]. This approach is commonly known as "padding" [Kollig and Keller 2002].

Relatively recently, Grünschloß et al. [2012] introduced the idea of using a high-dimensional low-discrepancy sequence to jointly sample the whole domain. They demonstrated that at least the first two dimensions of low-discrepancy sequences like Sobol and Halton are fully stratified, even at a low sample count. Their idea, then, is to allocate these dimensions for sampling the area around the pixel, and invert the coordinates of the pixel to retrieve the index of the samples projected on it in the high-dimensional space, and then use these indices to retrieve the actual samples. This approach comes with many advantages, as advocated by Keller [2013], including a faster convergence rate. Among the advantages is an inherent error diffusion capability offered by the low-discrepancy sequences. On the downside, low-discrepancy sequences come with excessive correlation between the samples, leading to structured aliasing artifacts at low sampling rates, as can be seen in Fig. 1(e).

At the time of this writing, there is no consensus, to our knowledge, about whether to use joint high-dimensional low-discrepancy sampling or continue with the padding strategy. For example, the current version of PBRT chooses a global Sobol sequence as the default sampler [Pharr et al. 2016], whereas Sony Pictures Imageworks seem to continue to use padding for their Arnold renderer, and report that they "have found that replicating high-quality low-dimensional patterns in higher dimensions is more successful than trying to generate very high-dimensional patterns directly" [Kulla et al. 2018].

The method we describe in this paper may be considered a global sampling strategy, in the sense that the samples for the whole image are generated as one contiguous sequence of samples. Our model, however, does not require that the samples are generated jointly for all the dimensions. We use padding in all the renderings in this paper, but our model supports joint sampling as well.

3 OUR METHOD

In this section we develop our proposed model of a sampler that diffuses Monte Carlo sampling error as a blue noise in screen space. We recapitulate that the basic functionality of a sampler is to supply, for each pixel, a set of samples, where each sample is a high-dimensional number in $[0, 1)^d$ that will be used to construct a ray path. As discussed in Section 2.3, samples can be generated jointly for all d dimensions, using a d-dimensional low-discrepancy sequence, e.g., Halton or Sobol, or obtained by padding [Kollig and Keller 2002]: breaking down the d-dimensional space into (typically) one- and two-dimensional sub-domains, e.g., lens, time, area-lights, etc., and synthesizing the sample by combining samples (e.g., Sobol sequences) taken from these sub-domains. In the following discussion, we consider padding, but the concept applies similarly to joint high-dimensional sampling.

To simplify the description of our sampler, we make two assumptions that we will relax later. First, we assume the image to be rendered is a square whose width (height) n in pixels is a power of 2, i.e., $n = 2^r$. Secondly, we assume only a single sample per pixel.

Our method consists of two major steps:

(1) We design a family
$$z(x, y) : \mathbb{N}^2 \rightarrow \{0, \ldots, n^2 - 1\} \tag{1}$$
of functions, one per constituent sub-domain, that map pixel coordinates to integer indices. Each of the functions creates a linear ordering of pixels, so that each pixel receives a unique integer number (per sampled sub-domain) from 0 to $n^2 - 1$.
(2) We use the integer indices at each pixel to index a low discrepancy sequence, namely Sobol.

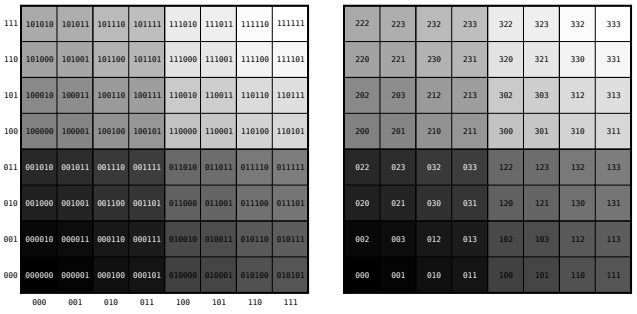
Basically, our algorithm assigns low-discrepancy sequence indices to pixels for sample generation. Our design relies on a well-known property of Sobol sequences that the first 2^r samples of a Sobol sequence can be recursively split into halves, and each half is a low-discrepancy sequence [Pharr et al. 2016]. Our idea, then, is to generate a contiguous 2^{2r} sequence of samples for the whole image, and partition it such that adjacent pixels receive power-of-two blocks of indices; hence, a low-discrepancy sequence of samples. Seen the other way around, we order the pixels into a linear sequence that preserves locality and maps two-dimensional blocks of pixels to power-of-two sub-sequences of indices.

3.1 Morton Ordering

As a basis for constructing our desired functions, we use a well-established solution for this 2D-to-1D ordering problem, known as Morton ordering [Morton 1966], or Z-ordering[1]. An index number
$$z(x, y) =_2 y_{r-1}x_{r-1} \cdots y_1x_1y_0x_0 \tag{2}$$
is computed for each pixel by interleaving the bits of its y and x coordinates, as illustrated in Fig. 2. We then assign the $z(x, y)$th sample of a Sobol sequence to that pixel.

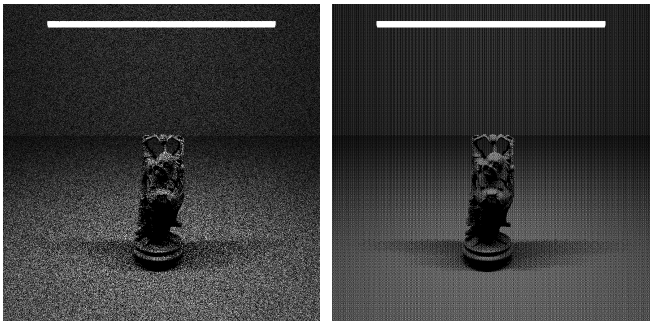
A basic sampler, which we will refer to as Z_0 sampler, is obtained by using this same mapping function in Eq. (2) for all sampled sub-domains. Fig. 3 shows a rendering with this sampler, using a non-scrambled (0, 2) Sobol sequence to generate the samples. It already achieves excellent error diffusion, but not a blue noise one, since there is a clear directional preference in Z-ordering: adjacency is defined in a strict left-to-right bottom-up order, as implied by the bit interleaving process in Eq. (2). The ordering interacts with the systematic ordering of the contracted Sobol sequence, leading to the visible structured aliasing. For example, the first dimension of a Sobol sequence is the van der Corput sequence, which simply mirrors the bits of the sequence number around the fractional point. For a single sample per pixel, the first dimension of the samples therefore reads $0.x_0y_0x_1y_1 \cdots$. Hence, all the evenly-/oddly-indexed columns of pixels receive samples in the first/second half of the sampled axis, respectively. In addition, using the same ordering for all dimensions maximizes inter-dimensional correlation; unless we use a different or a scrambled Sobol sequence for each dimension,

[1] Not to be confused with the Z-ordering of the frame buffer.



(a) (b)

Fig. 2. Morton ordering of pixel indices in an 8×8 image. (a) A binary encoding of the indices and the x and y coordinates to show how they relate. If $x =_2 x_2x_1x_0$ and $y =_2 y_2y_1y_0$ then the index is $y_2x_2y_1x_1y_0x_0$. The indices in (b) are encoded in base-4 to highlight their hierarchical grouping by quadrants. The leading digit indicates which quadrant of the image the pixel belongs to, the second digit indicates which sub-quadrant, and so on, down to the least significant digit that indicates its location in the 2×2 block of siblings. The gradient color is a visual aid to see how the pixels are ordered from (0) black to (63) white.



(a) (b)

Fig. 3. A rendering with 1 sample per pixel using (a) the (0, 2)-Sequence sampler, and (b) the same underlying Sobol sequence, but replacing the xor-scrambling of the samples by a Z-ordeing of the pixels to change the sample indices. Z-ordering exhibits a good performance in diffusing the sampling error, but introduces significant aliasing artifacts.

the samples are identical for all dimensions. Thus, we have to modify the Z-ordering process to avoid these problems.

3.2 Scrambled Morton Ordering

We modify Morton's Z-ordering by adding a stochastic element to 1) break its regular structure and 2) vary its ordering between separately sampled sub-domains. We note that Z-ordering is not unique, but is one of a family of orderings of two- or higher-dimensional arrays into locality-preserving linear sequences. For example, Fig. 4 gives a visual illustration of an alternative ordering process with the same, or arguably better, quality in preserving adjacency. Yet another alternative is the Hilbert space-filling curve, which has already been proposed for dithering in halftoning [Velho and Gomes 1991].

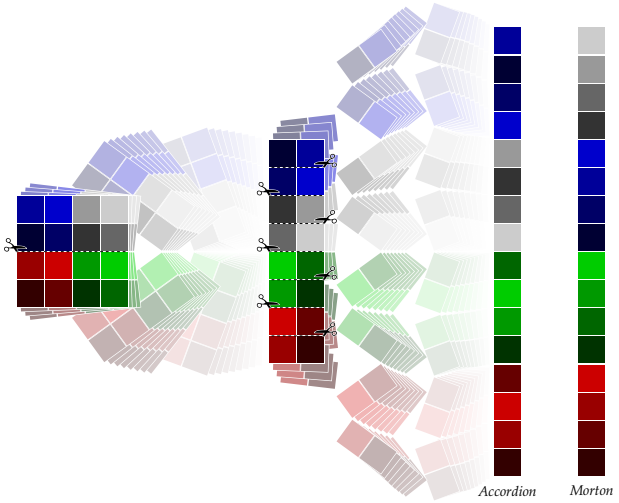


Fig. 4. A visual illustration of a systematic procedure to order a 2D block of pixels into a 1D sequence. Morton ordering is shown for comparison. First, the block is cut horizontally in the middle, leaving a pivot point at the end of the cut, and the two halves are rotated around the pivot. Subsequent steps consist of cutting from alternating directions and rotating. Note that if we invert the direction of the cuts in the second step, we avoid the jump from the bottom-left green pixel to the top right gray one, leading to Moore curve ordering.

The shared property of the mentioned orderings is that they are hierarchical, meaning that they do not only preserve immediate adjacency, but hierarchical levels of neighborhood, that is, each quadrant of the 2D image maps to a contiguous sequence of 1D indices, and this applies recursively all the way down to 2×2 pixel blocks. Besides preserving adjacency, this family of orderings perfectly matches our design requirement of having power-of-two block sizes to align with the Sobol sub-sequences.

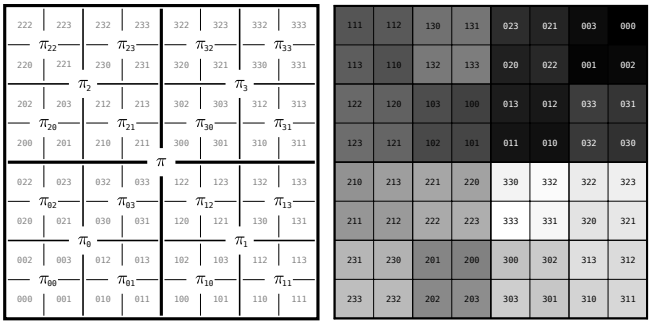
Morton ordering is arguably the easiest to compute; thus, we use it as a reference, and observe that all the alternative orderings are hierarchical permutations of it. For example, the accordion ordering in Fig. 4 can be derived from Morton ordering by swapping the blue and gray blocks, and applying the appropriate permutations inside each of the four blocks. A systematic ordering such as the Hilbert curve would have a systematic permutation to derive it from Morton. A stochastic Morton-like ordering, on the other end, is obtained by applying random permutations

$$\pi : \{0, 1, 2, 3\} \rightarrow \{0, 1, 2, 3\} \tag{3}$$

to the order of children of every block in the hierarchy. These permutations replace the fixed directional preference of Morton ordering by 4! = 24 directions, leading to a more isotropic diffusion of error, as required to obtain a blue noise distribution of error. Overall we have

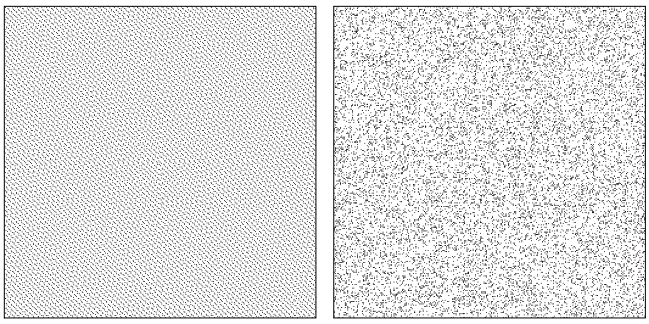
$$\sum_{i=0}^{\log_2(n)-1} 4^i = (4^{\log_2(n)-1} - 1)/3 \tag{4}$$

permutations to specify, organized as illustrated in Fig. 5. Our naming convention is to index the permutation nodes after the block



(a) (b)

Fig. 5. (a) Available degrees of freedom to scramble a Morton ordering. Each π is a permutation of $\{0, 1, 2, 3\}$ that indicates the order of the bottom-left, bottom-right, top-left, and top-right quadrants. Our naming convention is to index the permutation nodes after the block they operate on in the canonical Morton ordering, not the scrambled one. For example, π_3 will always refer to the order of child quadrants of the top-left quadrant, irrespective of the permuted order of this quadrant in the topmost π. (b) An example scrambled Morton ordering, obtained by setting $\pi = \{2, 3, 1, 0\}$, $\pi_0 = \{3, 0, 1, 2\}$, etc. Again we note that π_0 refers to the order of the second digit in the bottom-left quadrant, irrespective of the first digit assigned by π.



(a) (b)

Fig. 6. Union sets of respective 2D samples assigned to the same 128×128 block of pixels in (a) Fig. 1(c) and (b) Fig. 1(d). The points in (a) are generated as one low-discrepancy sequence of 16384, whereas the points in (b) are generated using xor scramblings of points from a small set of 256 points. The measured discrepancies are (a) 0.000372996695 and (b) 0.011730194092.

they operate on in the canonical Morton ordering defined in Eq. (2), not the scrambled one. For example, a permutation named π_{abc} applies to all the pixels whose canonical Morton indices share the leading three base-4 digits abc, and permutes the fourth digit. With this naming convention, the index of pixel (x, y) in a scrambled Z-ordering becomes

$$z^{\pi}(x, y) =_2 \pi(z_{r-1})\pi_{z_{r-1}}(z_{r-2}) \cdots \pi_{z_{r-1} \cdots z_1}(z_0), \tag{5}$$

where each

$$z_j =_2 y_j x_j \tag{6}$$

is the jth base-4 digit of the cananical Z-index, as in Eq. (2).

Eq. (5) represents the final version of our desired sampler mapping functions of Eq. (1), using a different permutation for each sampled sub-domain. We call the corresponding sampler Z sampler. An actual realization of this sampler produces the result in Fig. 1(c), which is quite satisfactory, and the distribution of noise is verily a blue noise, as evident in the frequency power spectrum. Fig. 6 shows the union set of the samples assigned to a block of pixels in a typical rendering. In contrast to existing error-diffusion sampling techniques, our sampler, by design, attains the highest-quality low-discrepancy sequences for both individual pixels and blocks of adjacent pixels.

Finally, we review the simplification assumptions we made at the beginning of this section. Handling non-square image sizes is done easily by padding the size to the nearest power of two, and just ignoring the padding pixels. Multiple samples per pixel can also be easily handled by appending the sample number to the canonical Z-order of the pixel in Eq. (2), and continuing the permutation process of Eq. (5), virtually treating sub-pixels as if they were pixels of a higher-resolution rendering.

This completes our theoretical model, and it remains to make the appropriate choices for a practical implementation. The challenging part is the permutation tree, which can grow very large, as implied in Eq. (4). Indeed, this hierarchical permutation of Morton ordering is reminiscent of Owen's scrambling [Owen 1995, 1998], though it should not be mixed with it. Owen's scrambling applies to the sampled domain, and is axis-wise, while we scramble the pixels, and the scrambling is made jointly in two dimensions. Nevertheless, we face the same challenge of finding an efficient implementation.

4 IMPLEMENTATION

A rendering application may scan the pixels in any order; hence, it is not an option to generate and consume the permutations on the fly, as described in [Friedel and Keller 2002], for example. The ordering of pixels has to be randomly accessible. As discussed in Section 3.2, multiple sets of permutations may be needed, one per sampled sub-domain. Storing such a huge set of permutations does not sound feasible. In fact, even generating such sets is time-consuming. Thus, building a sampler with our proposed concept reduces to the efficient implementation of a hierarchical permutation. In this section we describe two different practical approaches that achieve this goal. The first approach aims at efficiency in both speed and memory, and is well suited for GPU-based real-time applications. The second aims at flexibility, and is more suitable as a basis for future research.

4.1 Arithmetic Hashing

One option that is already in common use with Owen's scrambling [Christensen et al. 2018] is hashing. The idea is to design a function

$$\phi\colon \mathbb{N}^2 \to \{0..23\} \tag{7}$$

that maps consecutive prefixes of the unscrambled Z-index into one of the 24 permutations of $\{0, 1, 2, 3\}$, as suggested by Eq. (5). This function has to meet the following requirements:

(1) Injective, producing the same output for the same input, so as to enable random access for an arbitrary evaluation order of the pixels.
(2) Two-dimensional input, producing a different hashing for each sampled dimension.

(3) Evenly distributed and quasi-random over the output range.
(4) Efficient!

The last requirement is especially important for a practical sampler, since any significant processing may render the whole solution incompetent. Indeed, if the error diffusion is costly it becomes more feasible to increase the sampling rate and reduce the sampling error instead of diffusing it.

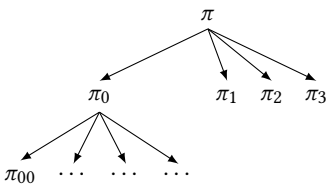
With these requirements in mind, we designed the multiplicative hashing function

$$\phi(i, d) = \texttt{int(frac((}i\ \texttt{\^{}}\ \texttt{(0x55555555 *}\ d\texttt{)) *}\ \alpha\texttt{) * 24)}, \quad (8)$$

where i is the designated prefix of the unscrambled Z-index, and d is the sampled dimension. The multiplicand α is chosen to be a badly approximable number. In our implementation, we use the golden ratio $\varphi = (1 + \sqrt{5})/2$. The purpose of multiplying by `0x55555555` is to inject the dimension information in all the bits of the hashed number. We use fixed-point arithmetic over integers for maximum efficiency. Please note, however, that we do not claim optimality in this design.

4.2 Hashing with Recursive Lookup Trees

We now describe a more sophisticated hashing approach based on self-similar lookup trees, as commonly used in recursive tile-based blue noise sample generators [Ahmed et al. 2017b; Ostromoukhov 2007; Ostromoukhov et al. 2004; Wachtel et al. 2014]. We first note that the permutations in Fig. 5, even though independent, are overlaid in a hierarchical tree structure



that essentially reflects the recursive subdivision of the image into quadrants. The idea, then, is to reuse the same sequence of permutations at different depths down the hierarchy. For example, we make all the permutations from π_0 and below identical to all the permutations from π and below. Indeed, we are cautious not to replicate the same permutations at the same level, but supposedly it should not harm to scramble the first quadrant as we do the whole image, or the second quadrant of the third quadrant as we do the first quadrant of the image.

Such a self-similar tree may be constructed using a context-free grammar. We choose an alphabet of symbols, for example,

$$\Sigma = \{A, B, C, D, E\}, \quad (10)$$

and define a uniform production rule between them, for example,

$$\psi: \begin{array}{l} A \mapsto BCDE \\ B \mapsto ACED \\ C \mapsto DEBA \\ D \mapsto CEBA \\ E \mapsto ABCD \end{array}. \quad (11)$$

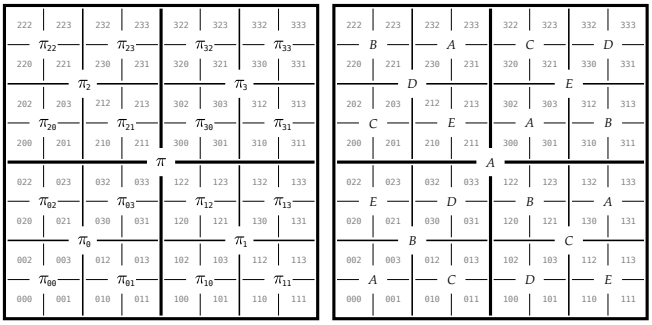
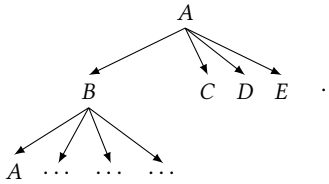


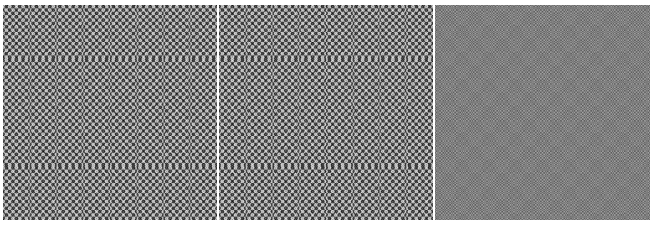
Fig. 7. Alignment between (a) the permutation nodes for scrambling the Morton ordering, and (b) the nodes of the two-dimensional self-similar tree defined by the alphabet in Eq. (10) and the production rule in Eq. (11).

This grammar defines a set of five self-similar quadtrees, starting at each symbol, for example,



Thus, each symbol actually represents an infinite tree rooted at that symbol, and every occurrence of the same symbol at any level reproduces the same tree down the hierarchy. Traversing such a tree is low-cost, calling only for a lookup into the production rule. Now, such a set of symbols may be equipped with any information, and used to distribute that information to any tree with the same branching rate. For example, we may assign permutations to the five symbols in our example. By aligning the tree in Eq. (12) with the tree in Eq. (9), π takes the permutation stored in A, and $\pi_0, \pi_1, \pi_2, \pi_3$ take the permutations stored in B, C, D, E, respectively, while π_{00} takes the permutation in A, and so on. Please note that these quadtrees are actually two-dimensional nested tiles that correspond to areas of the image, as illustrated in Fig. 7. The primary constraint is to avoid local repetitions at the same level. The production in Eq. (11) was carefully selected for this illustration, but in practice a large alphabet and a random production would be unlikely to produce local repetition. For an alphabet of 4k symbols, we compared the carefully designed repetition-avoiding grammar of [Ahmed et al. 2017b] to a random production rule, and there was no noticeable difference in the results, as will be shown in the following section.

This approach is quite simple and very efficient. Both the production rule and the permutations can be generated randomly as the sampler is initialized. The only parameter to decide is the size of the alphabet, which we choose to be reasonably large to ensure sufficient entropy in the distribution. For the examples in this paper, we used a set of 4k entries.



(a) (b) (c)

Fig. 8. In a 256×256 film resolution, we used dimension 5 of PBRT's global Sobol sequence to sample a grayscale ramp, and plotted the sampled gray level onto the pixels, showing (a) the first sample alone, (b), the second sample alone, and (c) the average. There is an evident correlation between this dimension and the first two dimensions used to designate the pixel indices, where, for 64k samples, the sequence associates each half of dimension 5 with certain blocks of dimensions 0 and 1, and for the subsequent 64k samples, it swaps the association. The sequence will eventually attain a fair distribution, but it may call for a large number of samples.

4.3 Sample Sequence Generation

Sobol low-discrepancy sequence comes as a natural choice for our sampler thanks to its binary nature that matches our base-4 neighborhood hierarchy. The remaining question is whether to use a high-dimensional sequence or to pad low-dimensional constituents. Indeed, our model supports both.

In our first implementation we considered a global Sobol sequence, where our model offers the advantage of saving the first two dimensions — the most precious ones — for actually sampling the path rather than for anti-aliasing and indexing the samples. However, we eventually reverted to padding. We arrived at that choice after a lot of experimentation and inspection. We found that some of the aliasing artifacts in high-dimensional Sobol sequences may not be removable with error diffusion techniques like our proposed one, because they stem from inherent inter-dimensional correlations in these sequences, as demonstrated in Fig. 8. Unfortunately, correlation patterns such as the illustrated one seem to be aligned with the *elementary intervals* [Owen 1995] of the aliased dimension, so scrambling techniques like Owen's seem to keep these blocks together.

Thus, we designate one- and two-dimensional Sobol sequences as the preferred sample sequence generator in our sampler, and use padding to synthesize high-dimensional sample vectors. All the results shown in this paper use this configuration. To avoid inter-dimensional correlation, we maintain a different pixel ordering function for each sampled sub-domain, as in Section 3.2.

4.4 A Completed Sampler

We now briefly describe a fully functional sampler based on the design principles outlined in the preceding subsections. Our description assumes the more sophisticated tree-based hashing, but we provide code for a PBRT implementation using both hashing methods in the supplementary materials of this paper.

The only manually set parameter is the size

$$N = |\Sigma| \tag{13}$$

of the alphabet of the recursive lookup tree. This is specified at compile time. The alphabet is implicitly defined by an ordinal sequence of integers

$$\Sigma = \{0..N-1\}\,. \tag{14}$$

The image resolution n is passed by the rendering application as a parameter, and is rounded up to the nearest power of 2.

When the sampler is initialized, two lookup tables are created. The first is an $N \times 4$ list, ψ, that defines the production rule, like in Eq. (11), and is populated uniformly-randomly with indices in the range $\{0..N-1\}$ that refer to indices of distinct nodes in the recursive tree. The second table is a $d \times N \times 4$ list, π, that assigns permutations per node per dimension or pair of dimensions. An index is nominated as the root node, logically aligned with π in Fig. 5.

The integrator passes a pixel coordinate, and requests a number of samples m for a list of dimensions The number of samples may be different for each dimension, since the integrator may choose to *split* the light path at some point [Keller 2013; Pharr et al. 2016], but it should be a power of 2. For now let us assume it is also a power of 4, and we will comment on odd powers of 2 later. The list of samples is computed separately for each dimension, using the respective list of permutations for that dimension.

To retrieve the dth component of the ith sample, a canonical Z-index of the pixel

$$z =_4 z_{\log_2(n)-1} \cdots z_1 z_0 \tag{15}$$

is computed using Eq. (2), then the base-4 digits of the sample index i are appended to generate a Z-index of the sample:

$$z \leftarrow z \cdot m + i =_4 z_{\log_2(n)-1} \cdots z_1 z_0 i_{\log_4(m)-1} \cdots i_1 i_0\,, \tag{16}$$

as if we are continuing to partition the pixel into finer resolution sub-pixels, and use one sample per sub-pixel. Please note that this association of samples with sub-pixel quadrants is only nominal: the actual location of the samples inside the pixel is taken from one of the requested pairs of dimensions. The reason that we have to continue ordering down sub-pixels is that, even though the samples in each pixel are different from the other pixels, Sobol sequence generates the samples in a systematic order over consecutive powers of two, leading to correlations between the pixels that may cause undesirable effects [Pharr and Humphreys 2010, Chapter 7]. Recall that we have eliminated the sample randomization process, leaving our scrambling of sequence numbers as the only stochastic element.

Translating z, the canonical Z-index of the sample, into a scrambled one is performed digit by digit, starting at the most significant digit, because we have to traverse the scrambling tree. We use a variable $\alpha \in \Sigma$ to traverse the recursive tree that distributes the permutation data, and initialize it to the index of the nominated root node, typically 0. For each digit j, then, we perform two steps: we first translate the digit

$$z_j^{\pi} = \pi[d][\alpha][z_j] \tag{17}$$

using the permutation stored in the current α, and then we advance down the tree

$$\alpha \leftarrow \psi[z_j] \tag{18}$$

to the appropriate quadrant. These steps are iterated until we translate the least significant digit. When m is an odd power of two, we

proceed as above on $i/2$, and append the left bit, optionally xoring it against a random bit. Finally, the scrambled index number z^π is passed to a Sobol sequence generator to retrieve the actual sample point.

This concludes the description of the core functionality of our sampler, as implemented in the code in the supplementary materials. In Appendix A we discuss the more advanced topic of adaptive and progressive sampling, which may assist in future development.

5 RESULTS AND COMPARISON

In this section we compare various aspects of the sampler we developed in the preceding section, using lookup trees, to state-of-the-art samplers; namely PBRT's traditional (0, 2)-Sequence sampler, PBRT's new global Sobol sampler, and the error-diffusion sampler presented by Heitz et al. [Heitz et al. 2019], which we will refer to as "Heitz". To our knowledge, the latter is the current state of the art in blue-noise error diffusion. We then compare between different implementations of our sampler.

5.1 Visual Quality

While admittedly subjective, visual quality remains the most important evaluation aspect in rendering, because that is what the end users care about. Our sampling strategy stands out distinctively in this aspect, as demonstrated in Figs. 1, 9, and 10. We strongly encourage the reader to look at the full-resolution renderings in the supplementary materials, where more examples can also be found. Evidently, the Sobol sampler exhibits noticeable aliasing artifacts all the way up to convergence, whereas the (0, 2)-Sequence sampler is free of aliasing, but exhibits excessive noise. The performance of Heitz is inconsistent. For example, it leads in the Sportscar scene, but fails in the higher-dimensional Breakfast scene with the bidirectional path integrator. Our sampler, in contrast, exhibits a consistent behavior, and simultaneously avoids aliasing, reduces noise, and diffuses the residual noise as blue noise.

While our sampler evidently leads the competition, it is still not perfect. Specifically, careful inspection reveals a grid structure of the noise, apparently stemming from the hierarchical block-based distribution of the samples. That is, there is a better distribution of the samples between sibling pixels than between cousins, second cousins, and so on, and this kinship is aligned with a multi-resolution grid. This issue is reminiscent of an analogous problem with ordered dithering [Ulichney 1987], and the treatment considered there may be imported, e.g., Ostromoukhov's rotated matrices [1994], but given that these artifacts are significantly less noticeable than artifacts introduced by state-of-the-art methods, we consider such an improvement to be beyond the scope of this introductory paper, and we leave it for future research.

Before leaving this subsection, we would like to comment on our observation that the scene complexity and the integration strategy play significant roles in the comparison. Simple integrators, such as Whitted's and Direct Lighting, are more prone to aliasing, making the difference between the samplers more evident. We noticed that the effect of error diffusion decreases as the sample dimension increases, e.g., in complex scenes and/or more sophisticated integrators such as bidirectional path-tracing. This confirms earlier observations by Georgiev and Fajardo [2016], but we were able to cope with higher dimensions than their original effort.

5.2 Spectral Analysis

Fig. 1 shows the frequency power spectra of renderings with the four samplers, along with the frequency power spectrum of a reference high-quality rendering for comparison. We found this more informative than displaying the noise spectrum in selected regions.

The (0, 2)-Sequence exhibits the typical white-noise distribution of Monte Carlo error, as expected. Sobol adds structured noise that reflects the frequency structure of the sequence. Heitz exhibits a discretized blue-noise spectrum, concentrating the noise at periodic spikes. Our intuition is that this is related to the employed xor scrambling, which apparently tends to align the sample points, as can be seen in Fig. 6(b). Our sampler, in contrast, exhibits a typical blue-noise distribution of Monte Carlo noise, as desired.

5.3 Convergence rate

In Fig. 11, we show a quantitative comparison of the convergence rates of the renderings in Fig. 1 and Fig. 10, compared to high-quality reference renderings. Generally speaking, a high-dimensional global sampler like Sobol and Heitz would eventually overtake at considerably high sampling rates. This makes sense, because the low-discrepancy sequence would become fully stratified over all the important dimensions then. By the time this point is reached, however, the typical situation is that the rendering error would have already been suppressed below the noticeable range. We conclude that the advantage of global Sobol samplers is only earned where it is absolutely determined to reach full convergence.

Heitz stops at 256 samples because that is the size of the supplied table, but judging from the lower sample counts, its behavior is inconsistent. It generally tends to lead the competition, surpassing Sobol. We think this is not related to the error-diffusion optimization, but may be attributed to the inclusion of Owen's scrambling in the stored samples, which is known to make a significant improvement [Christensen et al. 2018]. On the other hand, Heitz sampler fails badly with bidirectional path tracing, as can be seen in the Breakfast scene. We have seen a very similar failure with the Halton sampler, which warrants more investigation.

The convergence behavior of our sampler is almost identical to the (0, 2)-Sequence sampler, as expected.

5.4 Speed Performance

A sampler is called extensively with very different combinations of parameters during the rendering process; hence, quantitative assessment of the speed performance of a sampler is not straightforward. Instead of comparing the sample generation rates, hundreds of millions per second, we find it more objective to compare the actual difference in using various samplers for the same scene setup. In Table 1 we show the actual rendering times for the scene in Figure 9. They should serve as a hint of how Z performs in comparison with other samplers. It performs pretty similarly to the (0, 2)-Sequence and Heitz, and consistently outperforms global Sobol. Thus, even where Sobol offers better convergence rates in terms of the sampling density, our sampler partially compensates this in terms of

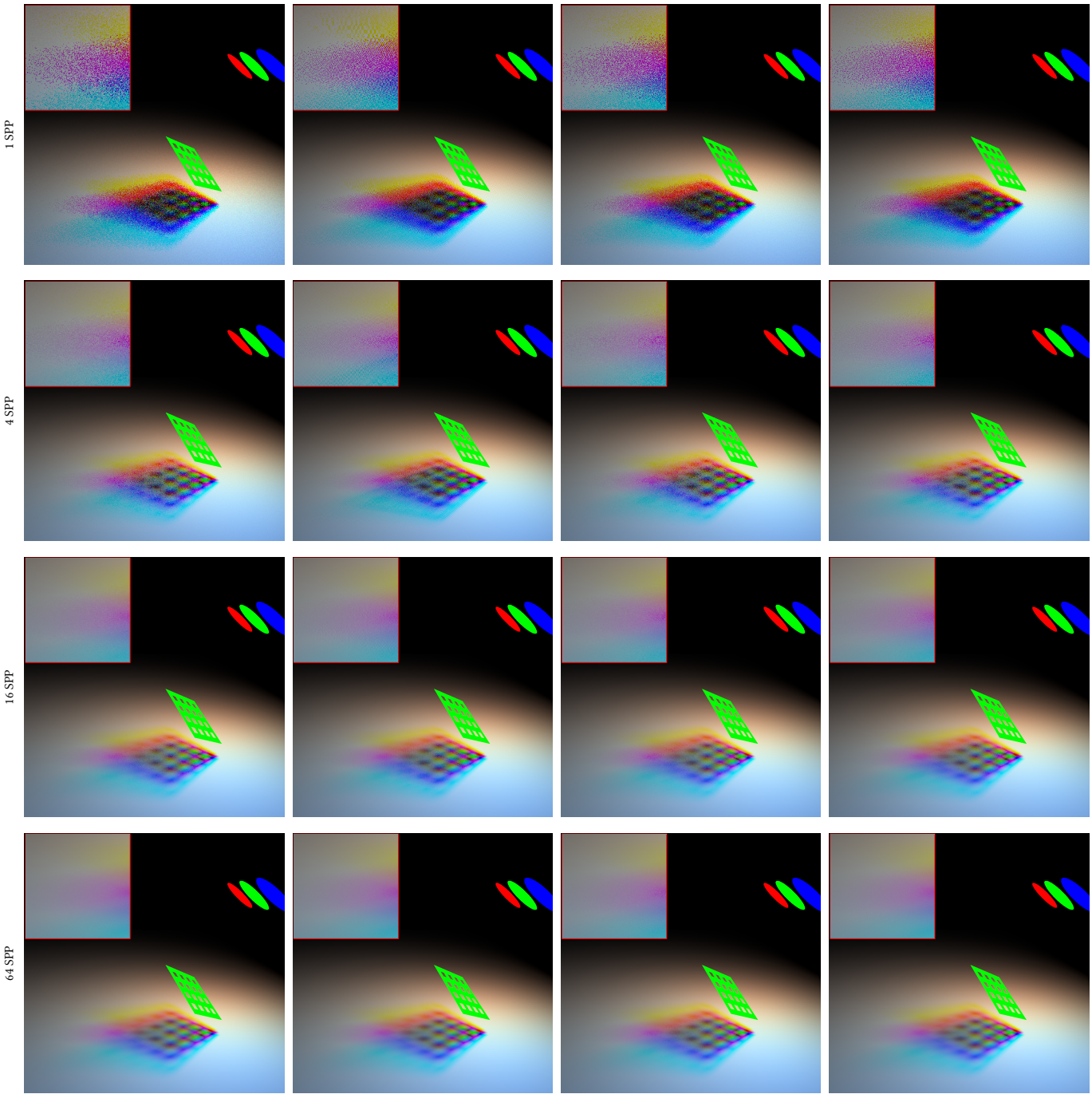


Fig. 9. A scene with a complex occluder and three area lights, sampled with different samplers. The color of the lights is varied to capture the interplay between the sampled dimensions in the different samplers. Our sampler evidently generates the common blue-noise distribution of the error.

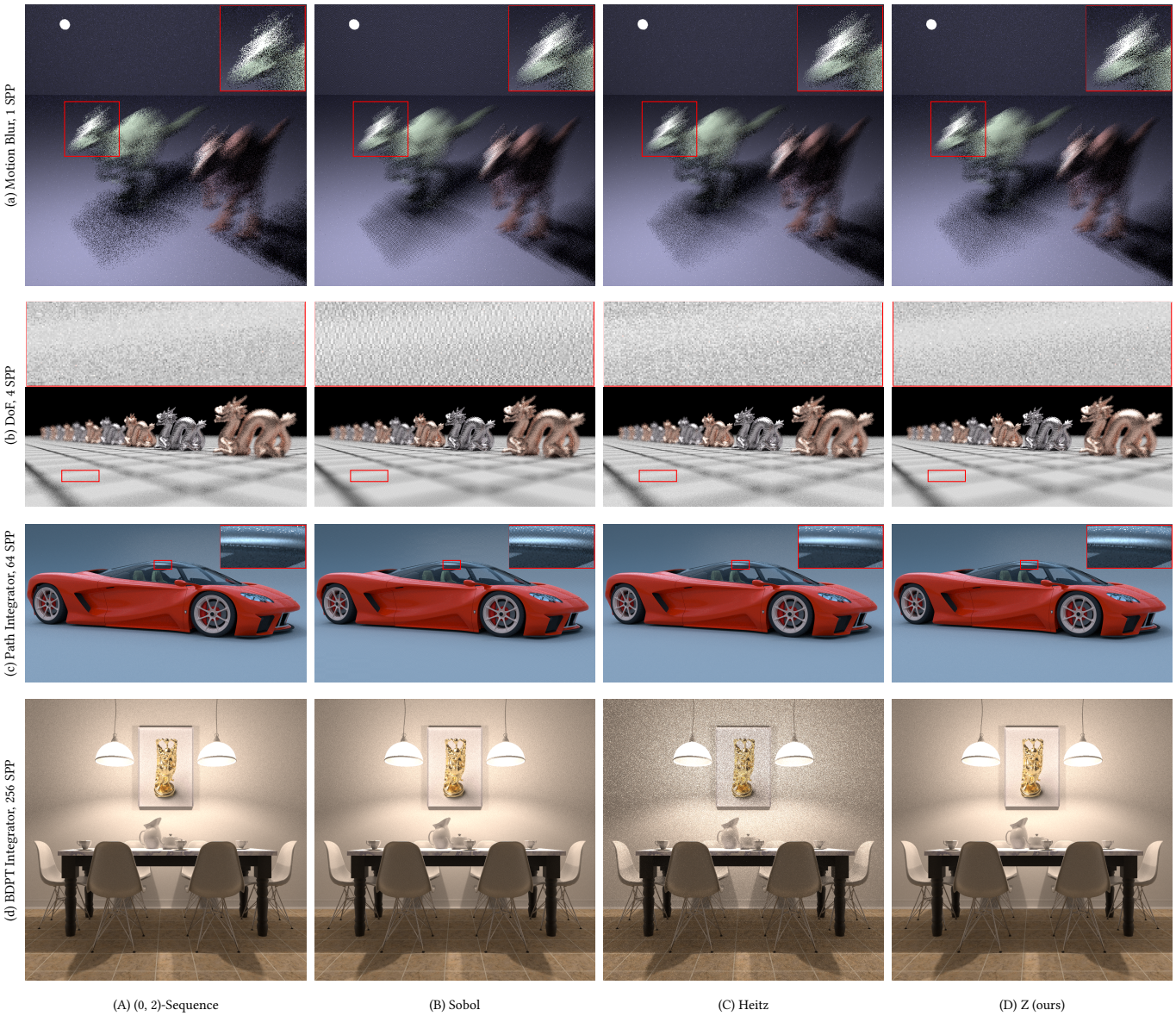


Fig. 10. Renderings of various scenes to illustrate how the Z-sampler compares to state-of-the-art samplers with different effects and scenarios: (a) motion blur, (b) depth of field, and (c) path tracing, and (d) bidirectional path tracing. While each of the three other samplers manifest some undesirable artifacts in at least one scenario, our sampler consistently delivers almost the best quality, although Heitz's slightly outperforms our method in the Sportscar scene, possibly thanks to the use of Owen's scrambled samples.

speed. This can only be stated qualitatively, however, since sample generation is not the only factor in the sample evaluation cost.

5.5 Disk and Memory Usage

The arithmetic hashing variant of our method does not use any tables, making it extremely efficient in this aspect. The look-up variant does not have to store any tables, but generates two lookup tables at run time that are populated with randomly computed entries. One table is for the production rules, and the other for the permutations. In all our tests, we used an alphabet (Section 4.4) of 4K entries, leading to a 64KB production table. This is a rather conservative size. Indeed, we experimented with as few as 256 entries, and it worked equally fine. However, we prefer a safe setting.

For the permutation table, permutations of four children, either by themselves or as a pointer, may be conveniently stored in a single byte, leading to a 4K table. We compute a different set of permutations for each dimension. The storage requirement scales linearly with the number of output dimensions. For example, for 1K

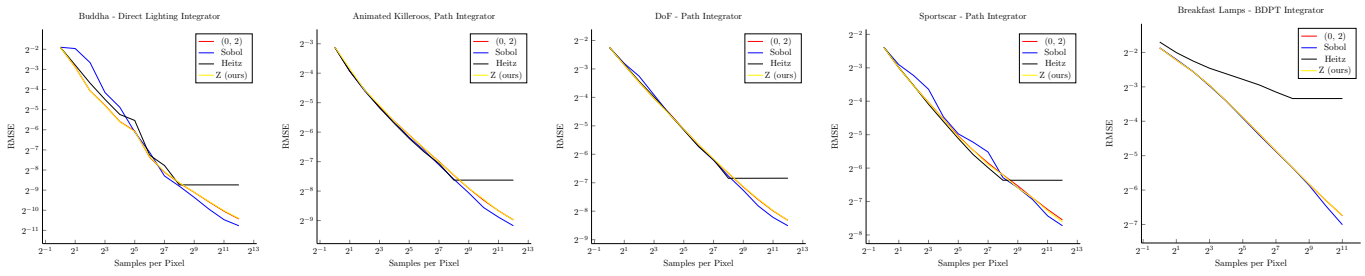


Fig. 11. Convergence rates of different samplers for the scenes in Fig. 1 and Fig. 10.

Table 1. Execution times (seconds) of the renderings in Fig. 9 at different sample-per-pixel (SPP) rates in a contemporary machine.

SPP	(0-2)	Sobol	Heitz	Z
1	0.2	0.2	0.2	0.5
2	0.5	0.5	0.5	0.5
4	0.5	0.8	0.5	0.5
8	0.8	1.0	0.8	0.8
16	1.2	1.8	1.2	1.2
32	2.0	3.5	2.0	2.0
64	4.0	6.5	4.0	4.0
128	7.5	12.5	8.0	7.5
256	14.0	25.5	14.0	15.0
512	28.0	51.5	27.5	28.5
1024	64.5	105.5	57.5	57.5
2048	107.5	213.5	113.5	113.5
4096	213.5	418.5	265.5	230.5
8192	427.5	837.5	422.5	461.5

dimensions (the choice of the Sobol sampler in PBRT) this leads to a 4MB memory footprint.

We think this table size is reasonably small for production rendering, but since our strategy is generic and may be adopted for real-time rendering in mobile devices, it may be desirable to reduce the size of the tables even further. Then a smaller production table can be used, and in that case it is a good idea to store a well-tested table than generate it on the fly. A production table of 256 entries takes only 1K storage space.

5.6 Coding Complexity

Our sampling strategy is based on the same underlying Sobol sequences used in PBRT and commercial rendering systems, and does not involve any external libraries other than a random number generator. We designed it to use the same interface as the (0, 2)-Sequence sampler of PBRT, making it easily portable to other frameworks. The implementation of the core routines fits in less than a hundred lines of code, and once the concept is clear, a competent undergrad student should be able to code it. Our implementation is available as a supplementary material.

5.7 Implementation Variants

The preceding comparisons were all based on the look-up tree-based hashing variant of our sampler. We now compare different aspects between four different implementations, using 1) Unscrambled Morton ordering, referred to as Z_0, 2) arithmetic hashing, referred to as Z_{hash}, 3) a lookup tree with random production rule, referred to as Z, and 4) a lookup tree using the production rule of Adaptive Regular Tiles (ART) [Ahmed et al. 2017b], referred to as Z_{ART}, which supposedly has excellent repetition avoidance behavior thanks to the underlying Thue-Morse word used to produce these tiles. The code for PBRT is provided in the supplementary materials.

Fig. 12(a) shows a visual comparison of the rendering quality between the different variants, and illustrates the effect of the table size in the lookup-based implementation. The full images are available in the supplementary materials of the paper. There is no visually noticeable difference between the hashing and the lookup variants with large tables. The latter also works reasonably well even with small table sizes, e.g., 16 symbols. Thus, this comparison is inconclusive for the choice of an implementation.

In Fig. 12(b) and (c) we compare the convergence behavior. The difference between the Morton variant: fixed ordering, and the 1-symbol tree that picks a random one of the 24 child-orderings for each dimension, gives some insight about the effect of the table size. Apparently, the small table sizes eventually lead to inter-dimensional correlations that hinder full convergence. Note that even the random sampler of PBRT exhibits a slowing down, compared to the expected Monte Carlo behavior, apparently because the samples are only *quasi*-random. Note also that the hashing-based implementation starts to slow down at some point, apparently because the hashing function starts to impose strong inter-dimensional correlations as well. Please note that all these differences are actually below the visually noticeable level.

In terms of speed, the performance is almost identical between arithmetic and tree-based hashing in our CPU-based implementation. We believe, however, that the described hashing function in Section 4.1 would be an order of magnitude faster than the lookup trees in a GPU, since it avoids the rather costly random memory accesses required to evaluate the tree-based hashing. Even for CPU it leads to a very compact implementation and produces satisfactory results. On the downside, multiplicative hashing uses Weyl sequences, which are known to have strong spikes at harmonic

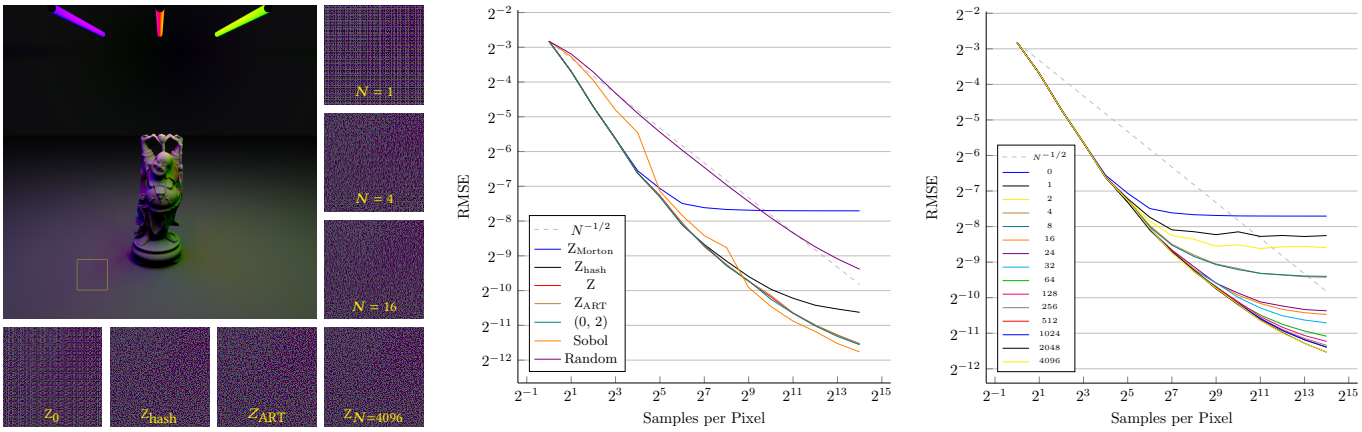


Fig. 12. Visual and convergence comparison between different implementations of our sampler. The Scene is designed to highlight inter-dimensional correlations. The reference image uses 64k (0, 2)-Sequence samples per pixel. Insets use two samples per pixel to reveal anomalies. Insets in the bottom show different implementation models. Insets on the right show renderings using small alphabet sizes of the lookup variant. Z_0 differs from the 1-entry lookup in that the former uses a fixed ordering for all dimensions, while the latter picks a random ordering for each dimension.

frequencies of the used multiplicand. While we have not seen a failure case, it is not clear to us where they might happen. Furthermore, convergence curves in Figure 12 indicate that hashing imposes a slight bias, possibly due to inter-dimensional correlation, that eventually hinders full convergence.

6 CONCLUSION

In this paper we presented a scaleable fully functional sampler that automatically diffuses the Monte Carlo error as a blue noise, without any offline optimization. We implemented our technique in PBRT-3 (code available in the supplementary materials), and tested it with a range of scenes. The Z-sampler we presented consistently surpasses the classic (0, 2)-sequence, is more reliable than global Sobol sampling that may unexpectedly introduce spurious aliasing artefacts, and is more scaleable than state-of-the-art error-diffusion samplers, while also offering better image quality. Our sampler uses Sobol sequences for the actual generation of the samples, and we do not compromise the performance of in-pixel sampling; hence, we maintain the same convergence rates as the traditional (0, 2)-Sequence sampler, but with error diffusion.

The concept we presented is also applicable to high-dimensional Sobol sequences, but we do favor padding for the reasons discussed in Section 4.3. An interesting idea is to adaptively decide on the number of joint dimensions, based on the requested number of samples, but this requires a thorough investigation of the employed Sobol matrices, and so we leave it for future research.

While we do not claim optimality, we believe that our model is already sufficiently reliable for adoption in real-time, interactive, and high-end rendering applications. Indeed, we tested on a larger variety of scenes, integration algorithms, and lighting configurations, and we are unaware of any failure case. For very high-dimensional integration scenarios, the quality of our sampler falls back to the quality of other state-of-the-art samplers, but not lower.

Besides the practical utility of our method, it also induces some theoretical interests. We managed to demonstrate that, without any explicit optimization, a low-discrepancy sequence combined with a quasi-random number generator can be arranged to deliver a blue-noise distribution of the Monte Carlo rendering error. This is interesting, as it gives some insights of what a high-dimensional blue-noise dithering-mask may look like. More interestingly, the arithmetic hashing variant, using Weyl's sequence, is 100% based on low-discrepancy sequences, and yet delivers a blue-noise distribution of sampling error.

There are many directions for future research. The technique can be improved in all of its aspects. For example, instead of using only the (0, 2) Sobol sequence, it would be interesting to adapt the jointly sampled dimensions in accordance with the sample count. It is also interesting to investigate efficient ways of integrating Owen's scrambling, since it apparently makes a considerable difference. An alternative is to use a pre-computed list of high-quality samples, as in [Christensen et al. 2018]. For the scrambling tree, it would be interesting to explore the possibilities of using an optimized distribution of the scrambling data, rather than a random one. Yet another direction for future research is to explore the options for perturbing the strict grid structure of the hierarchical blocks.

ACKNOWLEDGMENTS

Thanks to the anonymous reviewers for the valuable comments. We credit reviewer #1 for pointing out the advantage of arithmetic hashing for GPU implementation. Thanks to the scientific editing team at KAUST for proofreading the paper and to Mohanad Ahmed for his insightful discussions.

REFERENCES

A. G. M. Ahmed, J. Guo, D. M. Yan, J. Y. Franceschia, X. Zhang, and O. Deussen. 2017a. A Simple Push-Pull Algorithm for Blue-Noise Sampling. *IEEE Transactions on Visualization and Computer Graphics* 23, 12 (Dec. 2017), 2496–2508. https://doi.org/10.1109/TVCG.2016.2641963

Abdalla G. M. Ahmed, Hui Huang, and Oliver Deussen. 2015. AA Patterns for Point Sets with Controlled Spectral Properties. *ACM Trans. Graph.* 34, 6, Article 212 (Oct. 2015), 8 pages. https://doi.org/10.1145/2816795.2818139

Abdalla G. M. Ahmed, Till Niese, Hui Huang, and Oliver Deussen. 2017b. An Adaptive Point Sampler on a Regular Lattice. *ACM Trans. Graph.* 36, 4, Article 138 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073588

Abdalla G. M. Ahmed, Hélène Perrier, David Coeurjolly, Victor Ostromoukhov, Jianwei Guo, Dong-Ming Yan, Hui Huang, and Oliver Deussen. 2016. Low-Discrepancy Blue-Noise Sampling. *ACM Trans. Graph.* 35, 6, Article 247 (Nov. 2016), 13 pages. https://doi.org/10.1145/2980179.2980218

Michael Balzer, Thomas Schlömer, and Oliver Deussen. 2009. Capacity-Constrained Point Distributions: A Variant of Lloyd's Method. *ACM Trans. Graph.* 28, 3, Article 86 (July 2009), 8 pages. https://doi.org/10.1145/1531326.1531392

R. Bridson. 2007. Fast Poisson-Disk Sampling in Arbitrary Dimensions. In *ACM SIGGRAPH 2007 Sketches*.

Zhonggui Chen, Zhan Yuan, Yi-King Choi, Ligang Liu, and Wenping Wang. 2012. Variational Blue Noise Sampling. *IEEE Transactions on Visualization and Computer Graphics* 18, 10 (Oct. 2012), 1784–1796. https://doi.org/10.1109/TVCG.2012.94

Per Christensen, Andrew Kensler, and Charlie Kilpatrick. 2018. Progressive Multi-Jittered Sample Sequences. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 21–33.

Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. 2003. Wang Tiles for Image and Texture Generation. *ACM Trans. Graph.* 22, 3 (July 2003), 287–294. https://doi.org/10.1145/882262.882265

Robert L. Cook. 1986. Stochastic Sampling in Computer Graphics. *ACM Trans. Graph.* 5, 1 (Jan. 1986), 51–72. https://doi.org/10.1145/7529.8927

Robert L. Cook, Thomas Porter, and Loren Carpenter. 1984. Distributed Ray Tracing. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '84)*. ACM, New York, NY, USA, 137–145. https://doi.org/10.1145/800031.808590

Roy Cranley and Thomas NL Patterson. 1976. Randomization of Number-Theoretic Methods for Multiple Integration. *SIAM J. Numer. Anal.* 13, 6 (1976), 904–914.

Fernando de Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue Noise through Optimal Transport. *ACM Trans. Graph.* 31, 6, Article 171 (Nov. 2012), 11 pages. https://doi.org/10.1145/2366145.2366190

Mark A. Z. Dippé and Erling Henry Wold. 1985. Antialiasing through Stochastic Sampling. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85)*. ACM, New York, NY, USA, 69–78. https://doi.org/10.1145/325334.325182

Daniel Dunbar and Greg Humphreys. 2006. A Spatial Data Structure for Fast Poisson-Disk Sample Generation. *ACM Trans. Graph.* 25, 3 (July 2006), 503–508. https://doi.org/10.1145/1141911.1141915

Fredo Durand. 2011. A Frequency Analysis of Monte Carlo and Other Numerical Integration Schemes. *MIT CSAIL Tech. Rep. TR-2011-052* (2011).

Mohamed S. Ebeida, Muhammad A. Awad, Xiaoyin Ge, Ahmed H. Mahmoud, Scott A. Mitchell, Patrick M. Knupp, and Li-Yi Wei. 2014. Improving Spatial Coverage while Preserving the Blue Noise of Point Sets. *Computer-Aided Design* 46, Supplement C (2014), 25–36. https://doi.org/10.1016/j.cad.2013.08.015 2013 SIAM Conference on Geometric and Physical Modeling.

Mohamed S. Ebeida, Andrew A. Davidson, Anjul Patney, Patrick M. Knupp, Scott A. Mitchell, and John D. Owens. 2011. Efficient Maximal Poisson-Disk Sampling. *ACM Trans. Graph.* 30, 4, Article 49 (July 2011), 12 pages. https://doi.org/10.1145/2010324.1964944

Mohamed S. Ebeida, Scott A. Mitchell, Anjul Patney, Andrew A. Davidson, and John D. Owens. 2012. A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions. *Comp. Graph. Forum* 31, 2pt4 (May 2012), 785–794. https://doi.org/10.1111/j.1467-8659.2012.03059.x

Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi. 1997. The Farthest-Point Strategy for Progressive Image Sampling. *Trans. Img. Proc.* 6, 9 (Sept. 1997), 1305–1315. https://doi.org/10.1109/83.623193

Raanan Fattal. 2011. Blue-Noise Point Sampling Using Kernel Density Model. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 48, 12 pages. https://doi.org/10.1145/1964921.1964943

Ilja Friedel and Alexander Keller. 2002. Fast Generation of Randomized Low-Discrepancy Point Sets. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*. Springer, 257–273.

Manuel N. Gamito and Steve C. Maddock. 2009. Accurate Multidimensional Poisson-Disk Sampling. *ACM Trans. Graph.* 29, 1, Article 8 (Dec. 2009), 19 pages. https://doi.org/10.1145/1640443.1640451

Iliyan Georgiev and Marcos Fajardo. 2016. Blue-Noise Dithered Sampling. In *ACM SIGGRAPH 2016 Talks*. 1–1.

Leonhard Grünschloß, Matthias Raab, and Alexander Keller. 2012. Enumerating Quasi-Monte Carlo Point Sequences in Elementary Intervals. In *Monte Carlo and Quasi-Monte Carlo Methods 2010*, Leszek Plaskota and Henryk Woźniakowski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 399–408.

Daniel Heck, Thomas Schlömer, and Oliver Deussen. 2013. Blue Noise Sampling with Controlled Aliasing. *ACM Trans. Graph.* 32, 3, Article 25 (July 2013), 12 pages. https://doi.org/10.1145/2487228.2487233

Eric Heitz and Laurent Belcour. 2019. Distributing Monte Carlo Errors as a Blue Noise in Screen Space by Permuting Pixel Seeds Between Frames. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 149–158.

Eric Heitz, Laurent Belcour, V. 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 *ACM SIGGRAPH 2019 Talks (SIGGRAPH '19)*. Association for Computing Machinery, New York, NY, USA, Article 68, 2 pages. https://doi.org/10.1145/3306307.3328191

Wojciech Jarosz, Afnan Enayet, Andrew Kensler, Charlie Kilpatrick, and Per Christensen. 2019. Orthogonal Array Sampling for Monte Carlo Rendering. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 135–147.

Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. 2015. Blue Noise Sampling Using an SPH-Based Method. *ACM Trans. Graph.* 34, 6, Article 211 (2015), 11 pages. https://doi.org/10.1145/2816795.2818102

Thouis R Jones. 2006. Efficient Generation of Poisson-Disk Sampling Patterns. *Journal of graphics, gpu, and game tools* 11, 2 (2006), 27–36.

Alexander Keller. 2013. Quasi-Monte Carlo Image Synthesis in a Nutshell. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, Josef Dick, Frances Y. Kuo, Gareth W. Peters, and Ian H. Sloan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 213–249.

Andrew Kensler. 2013. Correlated Multi-Jittered Sampling. *Pixar Technical Memo 13-01* 7 (2013), 86–112.

Thomas Kollig and Alexander Keller. 2002. Efficient Multidimensional Sampling. In *Computer Graphics Forum*, Vol. 21. 557–563.

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang Tiles for Real-Time Blue Noise. *ACM Trans. Graph.* 25, 3 (July 2006), 509–518. https://doi.org/10.1145/1141911.1141916

Christopher Kulla, Alejandro Conty, Clifford Stein, and Larry Gritz. 2018. Sony Pictures Imageworks Arnold. *ACM Trans. Graph.* 37, 3, Article 29 (Aug. 2018), 18 pages. https://doi.org/10.1145/3180495

Ares Lagae and Philip Dutré. 2006. An Alternative for Wang Tiles: Colored Edges Versus Colored Corners. *ACM Trans. Graph.* 25, 4 (Oct. 2006), 1442–1459. https://doi.org/10.1145/1183287.1183296

Michael McCool and Eugene Fiume. 1992. Hierarchical Poisson-Disk Sampling Distributions. In *Proceedings of the Conference on Graphics Interface '92*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 94–105. http://dl.acm.org/citation.cfm?id=155294.155306

Don P. Mitchell. 1991. Spectrally Optimal Sampling for Distribution Ray Tracing. *SIGGRAPH Comput. Graph.* 25, 4 (July 1991), 157–164. https://doi.org/10.1145/127719.122736

Scott A. Mitchell, Mohamed S. Ebeida, Muhammad A. Awad, Chonhyon Park, Anjul Patney, Ahmad A. Rushdi, Laura P. Swiler, Dinesh Manocha, and Li-Yi Wei. 2018. Spoke-Darts for High-Dimensional Blue-Noise Sampling. *ACM Trans. Graph.* 37, 2, Article Article 22 (May 2018), 20 pages. https://doi.org/10.1145/3194657

GM Morton. 1966. A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. (1966).

Victor Ostromoukhov. 2007. Sampling with Polyominoes. *ACM Trans. Graph.* 26, 3, Article 78 (July 2007). https://doi.org/10.1145/1276377.1276475

Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. 2004. Fast Hierarchical Importance Sampling with Blue-Noise Properties. In *ACM SIGGRAPH 2004 Papers (SIGGRAPH '04)*. ACM, New York, NY, USA, 488–495. https://doi.org/10.1145/1186562.1015750

Victor Ostromoukhov, Roger D. Hersch, and Isaac Amidror. 1994. Rotated Dispersed Dither: A New Technique for Digital Halftoning. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. Association for Computing Machinery, New York, NY, USA, 123–130. https://doi.org/10.1145/192161.192188

Art B. Owen. 1995. Randomly Permuted (t,m,s)-Nets and (t, s)-Sequences. In *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, Harald Niederreiter and Peter Jau-Shyong Shiue (Eds.). Springer New York, New York, NY, 299–317.

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

A. Cengiz Öztireli. 2016. Integration with Stochastic Point Processes. *ACM Trans. Graph.* 35, 5, Article 160 (Aug. 2016), 16 pages. https://doi.org/10.1145/2932186

A. Cengiz Öztireli. 2020. A Comprehensive Theory and Variational Framework for Anti-aliasing Sampling Patterns. *Computer Graphics Forum* 39, 4 (2020), 133–148. https://doi.org/10.1111/cgf.14059 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14059

A. Cengiz Öztireli and Markus Gross. 2012. Analysis and Synthesis of Point Distributions Based on Pair Correlation. *ACM Trans. Graph.* 31, 6, Article 170 (Nov. 2012), 10 pages. https://doi.org/10.1145/2366145.2366189

Hélène Perrier, David Coeurjolly, Feng Xie, Matt Pharr, Pat Hanrahan, and Victor Ostromoukhov. 2018. Sequences with Low-Discrepancy Blue-Noise 2-D Projections. *Computer Graphics Forum (Proceedings of Eurographics)* 37, 2 (2018), 339–353.

Matt Pharr and Greg Humphreys. 2010. *Physically-Based Rendering: from Theory to Implementation* (2nd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Adrien Pilleboue, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov. 2015. Variance Analysis for Monte Carlo Integration. *ACM Trans. Graph.* 34, 4, Article 124 (July 2015), 14 pages. https://doi.org/10.1145/2766930

Bernhard Reinert, Tobias Ritschel, Hans-Peter Seidel, and Iliyan Georgiev. 2016. Projective Blue-Noise Sampling. *Computer Graphics Forum* 35, 1 (2016), 285–295. https://doi.org/10.1111/cgf.12725

Thomas Schlömer, Daniel Heck, and Oliver Deussen. 2011. Farthest-Point Optimized Point Sets with Maximized Minimum Distance. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics (HPG '11)*. ACM, New York, NY, USA, 135–142. https://doi.org/10.1145/2018323.2018345

Thomas Schlömer and Oliver Deussen. 2011. Accurate Spectral Analysis of Two-Dimensional Point Sets. *Journal of Graphics, GPU, and Game Tools* 15, 3 (2011), 152–160. https://doi.org/10.1080/2151237X.2011.609773 arXiv:http://dx.doi.org/10.1080/2151237X.2011.609773

Peter Shirley. 1991. Discrepancy as a Quality Measure for Sample Distributions. In *Proc. Eurographics '91*, Vol. 91. 183–194.

Robert Ulichney. 1987. *Digital Halftoning*. MIT Press, Cambridge, MA, USA.

R.A. Ulichney. 1988. Dithering with Blue Noise. *Proc. IEEE* 76, 1 (Jan 1988), 56–79. https://doi.org/10.1109/5.3288

Robert A Ulichney. 1993. Void-and-Cluster Method for Dither Array Generation. In *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*. International Society for Optics and Photonics, 332–343.

Luiz Velho and Jonas de Miranda Gomes. 1991. Digital Halftoning with Space Filling Curves. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. Association for Computing Machinery, New York, NY, USA, 81–90. https://doi.org/10.1145/122718.122727

Florent Wachtel, Adrien Pilleboue, David Coeurjolly, Katherine Breeden, Gurprit Singh, Gaël Cathelin, Fernando de Goes, Mathieu Desbrun, and Victor Ostromoukhov. 2014. Fast Tile-Based Adaptive Sampling with User-Specified Fourier Spectra. *ACM Trans. Graph.* 33, 4, Article 56 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601107

Yahan Zhou, Haibin Huang, Li-Yi Wei, and Rui Wang. 2012. Point Sampling with General Noise Spectrum. *ACM Trans. Graph.* 31, 4, Article 76 (July 2012), 11 pages. https://doi.org/10.1145/2185520.2185572

A ADAPTIVE AND PROGRESSIVE SAMPLING

The sampler design in Section 4.4 assumes a fixed number of samples per dimension, in alignment with the current sampler models of PBRT. However, thanks to the underlying low-discrepancy sequence, it can easily be adapted for adaptive and progressive sampling, although this would affect the error diffusion performance. We identify three distinct scenarios to provide some insight. Some familiarity with the actual operation of Sobol sequences is assumed; PBRT [Pharr et al. 2016, Chapter 7] is a good starting point.

The first scenario we would like to discuss is the screen-space adaptive sampling, where the rendering application may choose to sample some high-variance or more-important regions of the image at a higher sampling rate. The sampler in Section 4.4 will gracefully generate the samples if the requested number of samples m is varied between the pixels. It should be noted, however, that the sequence of samples is effectively independent for a different sample count, as can be seen by examining Eq. (16), since the sequence of bits is shifted up at the higher sample count. Thus, the error diffusion feature is maintained only across regions with the same sample count. This should not be a problem; all that is needed is to add a safety margin to the region with the higher sample count, and the low-variance pixels in the margin should absorb the error diffused from the high-variance pixels.

Next, we discuss the case where the rendering application chooses to adaptively terminate the integration process before consuming all the m nominated samples. Let, for example, m be four samples per pixel, and the rendering application only uses one of them. Consider two sibling pixels in the scrambled Z-ordering, which means that their indices share all but the least significant bit. When the two bits of the sample number are appended, the discriminating bit between the pixels is shifted to the third position, and when the index is passed to a Sobol sequence generator, this bit, possibly toggled, is mirrored against the fractional point, and rests in the third place after the fractional point. It is actually this bit that discriminates between the samples assigned to the two sibling pixels, and accounts for error diffusion. Now, since each pixel may have a different ordering of its assigned samples, the only guarantee is that each sample assigned to one of the pixels is at least $0.001_2 = 0.125_{10}$ away from every sample assigned to the other pixel, where the sampled domain is scaled to 1.

Thus, the error diffusion capability is not progressive, but is fixed by the nominated m. That is, if $m = 1$ then every pair of sibling pixels will enjoy the largest offset of 0.5 between their samples, in all axes, but if m is 8 then the first sample of each pixel may be any where in $\{1/8, 3/8, 5/8, 7/8\}$ from the corresponding sample of the sibling pixel, which is still a reasonable guarantee. This makes sense because we give the priority to securing the well-spaced-ness of the samples assigned to the same pixel, and then come the siblings. In practice, the mentioned shortcoming should not be a problem. Indeed, if the rendering application decides not to take more samples, then presumably it is already satisfied with the variance of the concerned pixel; hence, error diffusion is no longer needed.

Finally, we consider the scenario when the rendering application requests more samples after it consumes the nominated m sample; e.g., m more samples are requested. It is an option to treat this as a new dimension and generate a brand-new set of samples; then the error diffusion would continue to work, but at the cost of degrading the quality of the in-pixel samples by combining two independent sets. The more reasonable option is, again, to prioritise the distribution of the pixel samples and sacrifice the quality of error diffusion. To understand how it effects error diffusion, we note in Eq. (16) that incrementing i beyond m makes an overflow from the bits allocated to the sample to the range of bits allocated for the pixel index, which may lead to a very different index after scrambling. To avoid this, we should scramble the pixel index separately from the sample index, and xor the overflown bit against the least significant bit of the pixel index, effectively swapping the assigned samples between the pixels. For a single sample per pixel, this is very undesirable, but for higher sample counts, each pixel would have a different ordering of the samples, and hence synthesises a different set of paths. Still, this is sub-optimal, and degrades the error diffusion capacity.

We conclude from all these discussions that the parameter m is crucial to the error diffusion capacity, and has to be planned carefully. Error diffusion is maximized for the nominated number of samples, but if there is a chance of requesting more or less samples, then it seems better to choose a large m than a small one, since the former option only reduces the magnitude of the negative correlation between the samples assigned to sibling pixels, while the latter induces some positive correlation.