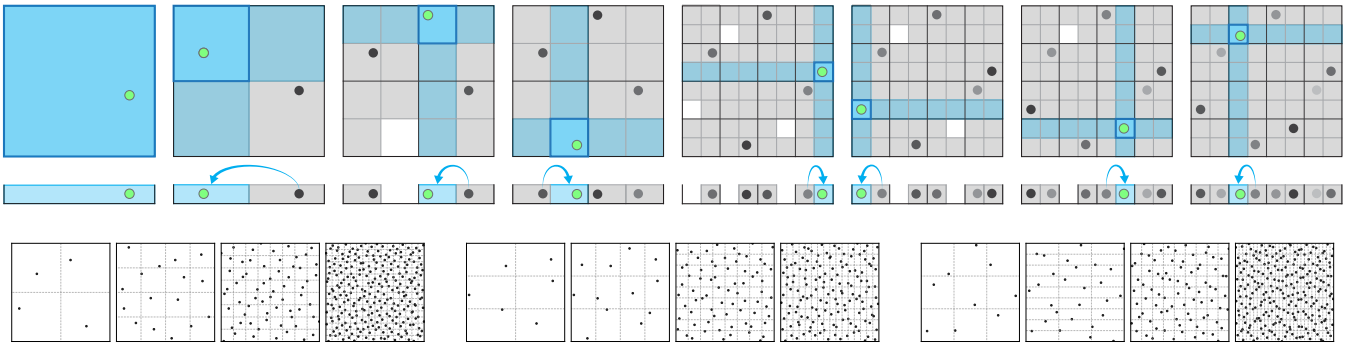
Eurographics Symposium on Rendering (DL-only Track) (2021)
A. Bousseau and M. McGuire (Editors)

Stochastic Generation of (t, s) Sample Sequences

Andrew Helmer Per Christensen[1] Andrew Kensler[2]

[1]Pixar Animation Studios [2]Amazon



(a) *ssobol02bn* **(b)** *shalton23bn* **(c)** *sfaure03bn*

Figure 1: *Top: stochastic generation of an Owen-scrambled Sobol' (0,2)-sequence. Unoccupied strata are determined in O(1) time by referencing earlier points in the sequence. Bottom: novel 2D sequences and some of their progressive stratifications, generated using our technique with best-candidate samples. These sequences are each extensible to higher dimensions.*

Abstract

We introduce a novel method to generate sample sequences that are progressively stratified both in high dimensions and in lower-dimensional projections. Our method comes from a new observation that Owen-scrambled quasi-Monte Carlo (QMC) sequences can be generated as stratified samples, merging the QMC construction and random scrambling into a stochastic algorithm. This yields simpler implementations of Owen-scrambled Sobol', Halton, and Faure sequences that exceed the previous state-of-the-art sample-generation speed; we provide an implementation of Owen-scrambled Sobol' (0,2)-sequences in fewer than 30 lines of C++ code that generates 200 million samples per second on a single CPU thread. Inspired by pmj02bn sequences, this stochastic formulation allows multidimensional sequences to be augmented with best-candidate sampling to improve point spacing in arbitrary projections. We discuss the applications of these high-dimensional sequences to rendering, describe a new method to decorrelate sequences while maintaining their progressive properties, and show that an arbitrary sample coordinate can be queried efficiently. Finally we show how the simplicity and local differentiability of our method allows for further optimization of these sequences. As an example, we improve progressive distances of scrambled Sobol' (0,2)-sequences using a (sub)gradient descent optimizer, which generates sequences with near-optimal distances.

CCS Concepts

• ***Mathematics of computing*** → ***Stochastic processes;*** *Computations in finite fields; Mathematical software performance;* • ***Computing methodologies*** → ***Rendering;*** *Ray tracing;*

1. Introduction

The rendering of computer generated scenes is often achieved by computing a set of integrals, representing the light reaching each pixel. Monte-Carlo and quasi-Monte Carlo (MCQMC) integration have become popular methods to approximate these integrals numerically, especially with path tracing [PJH17; Pha18; CFS*18].

Monte Carlo integration is performed by randomly sampling the integrand and adding weighted values to approximate the integral. Random error in the numerical approximation is visible in a rendered image as noise, and a large number of samples can be necessary to resolve this noise. Many techniques are employed to reduce the error, requiring fewer samples to reach an equivalent quality.

Fundamental to Monte Carlo integration is the generation of multidimensional sample points. These samples are often generated with each coordinate in the unit interval $[0,1)$ and warped to another domain. Sample points that are well-distributed in the unit hypercube give much lower error than uniform random samples [Mit96; Owe13; PJH17; CKK18; SÖA*19]. The generation of points in the unit hypercube is an ongoing area of research, with many different methods of sample generation [KAC*19]. These sample generation methods can be divided into those that generate well-distributed *sample sets* – a fixed number of samples – and methods that generate progressive *sample sequences*, with good distributions for prefixes of the sequence. Sample sequences have lower error in adaptive sampling and interactive (progressive) rendering, which has contributed to their popularity [Pha18].

Stratification splits the domain into non-overlapping regions (strata), typically hyperrectangles in rendering, and limits the number of sample points in each stratum, ensuring more even distributions. For simple integrals, stratification gives provably lower error and faster asymptotic convergence [Mit96], and distributions are better maintained when warping to other domains [PJH17]. Empirically, even complex rendering integrals have dramatically reduced error with stratified samples [CFS*18; JEK*19].

For those reasons, we focus on progressively stratified sample sequences. Two approaches are commonly taken to generate these sequences for rendering. The first is to use explicit stratification; data structures track the occupancy of strata, and points are placed stochastically into unoccupied strata. Alternatively, deterministic quasi-Monte Carlo (QMC) sequences are generated with constructions that guarantee stratification, and then are randomized in a way that maintains the stratification [LEc18; Owe95].

In this paper, we present a new approach that combines stratified sampling and randomized quasi-Monte Carlo: random samples are directly generated within unoccupied strata, without needing data structures to represent those strata. Our approach is capable of generating many different sample sequences and provides multiple improvements over previous sample generation methods:

- The pmj02(bn) sequences from Christensen et al. [CKK18] are naturally extended to higher dimensions, retaining the option of using best-candidate sampling. One way to extend to higher dimensions is as base-s $(0,s)$-sequences, which have even stronger stratifications in all lower dimensional projections than the orthogonal array sample sets of Jarosz et al. [JEK*19].
- The pmj02 sequences are generated roughly two orders of magnitude faster than with Pharr's efficient traversal [Pha19].
- Owen-scrambled sequences are simpler to implement and can be generated at a new state-of-the-art performance, faster than both hashing [LK11; Bur20] in base 2 and lazy permutation trees [FK02] in higher bases.
- The points of Owen-scrambled sequences are locally differentiable with respect to pseudorandom parameters.

The simplicity, high performance, and differentiability of this stochastic generation allows for new optimizations of multidimensional sample sequences. Section 6 will show an example using gradient descent to improve point spacing of sequences.

2. Related work

We begin with a review of previous work in stratified sampling for rendering.

Stratified sample sets. Cook [Coo86] introduced jittered sampling for solving rendering problems, randomly offsetting points on a uniform grid. Chiu et al. [CSW94] combined jittered sampling with Latin hypercube sampling to generate multi-jittered samples. Kensler [Ken13] improved the distribution of multi-jittered sampling with correlated shuffling. Jarosz et al. [JEK*19] presented a construction of sample sets based on orthogonal arrays, stratifying points in both high dimensions and lower-dimensional projections. The generation of these sample sets share an elegant property: none of them require data structures to implement, instead they are directly constructed to yield those stratifications.

Stochastically generated sample sequences. Unlike the algorithms to generate sample sets, stochastic generation of sample sequences generally requires either a spatial data structure, or $O(N^2)$ time to compute a sequence, as each sample must be checked against the location of previous samples. Kajiya [Kaj86] first presented algorithms for hierarchical stratified sampling for rendering, using a kd-tree to keep track of occupied strata. Keros et al. [KDS20] also used a kd-tree for high-dimensional stratification of arbitrary sample counts. Mitchell [Mit91] introduced best-candidate samples, where a set of random candidate points are generated, and the next point in a sequence is chosen to be the furthest of those candidates from all previous points.

Recently, Christensen et al. [CKK18] presented methods to generate stratified 2D sample sequences, notably the pmj02 sequences. They combined stratification with Mitchell's best-candidate sampling to generate "pmj02bn" sequences with improved point spacing. Pharr [Pha19] described a clever traversal of the strata to efficiently generate the pmj02 and pmj02bn sequences. The techniques we will present in this paper extend their work into higher dimensions, while also providing faster generation of pmj02 sequences.

Quasi-Monte Carlo sample sequences. Quasi-Monte Carlo sequences are an appealing alternative to sample sets and stochastically generated sequences. Mathematical constructions give deterministic sequences that have progressive stratification guarantees, while being easy to compute. We will use the constructions of the van der Corput [Cor35], Sobol' [Sob67], Halton [Hal60], and Faure [Fau82] sequences, and the definition of (t,s)-sequences from Niederreiter [Nie87]. For a more extensive overview of quasi-Monte Carlo sequences, see chapters 15 through 17 of Owen's book [Owe13] and Dick and Pillichshammer's book [DP10].

Scrambling and optimization. The deterministic construction of QMC sequences displays rendering errors as structured pixel artifacts [PJH17], rather than noise. These artifacts are less appealing visually and harder for modern denoisers to handle. The construction also prevents the quantification of uncertainty [Owe13]. To overcome those issues, QMC sequences must be *randomized*. L'Ecuyer [LEc18] provides a tutorial of randomized QMC techniques. Notably, Owen [Owe95] introduced a randomization that preserves stratifications, called nested uniform scrambling, Owen's scrambling, or *Owen-scrambling*, which will be described in Section 3.2. Owen-scrambling also reduces the error on smooth

integrals by decorrelating the less significant digits. Considerable work has been done to improve the performance of Owen-scrambling [FK02; AKI10]. Laine and Karras [LK11] observed that, in base 2, Owen-scrambling can be performed efficiently with hashing, which Burley [Bur20] expanded upon. Alternatively, simpler methods of scrambling are used [Mat98; KK02], which have slower convergence on smooth integrals [Owe03].

A related direction is the optimization of QMC sequences. Ahmed et al. [APC*16] and Perrier et al. [PCX*18] optimized van der Corput and Sobol' sequences, respectively, to achieve blue noise. Ahmed and Wonka [AW21] described an elegant method to optimize point sets with strict stratifications, but left the optimization of sequences as future work. Common to both randomization and optimization techniques is a set of *discrete* permutations on pre-generated points. In contrast, our new method generates Owen-scrambled sequences using a continuous, stochastic algorithm.

3. Generation of one-dimensional sequences

This section discusses techniques to generate 1-dimensional stratified sequences in the unit interval $[0,1)$, which will be expanded to higher dimensions in Section 4. The goal is to generate sequences with progressive stratification in a chosen base, b, where any prefix of $N = b^m$ sample points will have exactly one point in each sub-interval of length 1/N: $[\frac{0}{N}, \frac{1}{N}), [\frac{1}{N}, \frac{2}{N}), \ldots, [\frac{N-1}{N}, \frac{N}{N})$. We begin with background on the van der Corput sequence in Section 3.1 and Owen-scrambling in Section 3.2 before moving on to our contributions to 1D sequences in Section 3.3.

3.1. The van der Corput sequence

Van der Corput [Cor35] described a simple algorithm to generate one-dimensional stratified sample sequences. Intuitively, the van der Corput sequence progressively subdivides the unit interval into sub-intervals ("strata") and each new sample is placed at the lowest boundary of an unoccupied sub-interval, in the same order as the initial points. Table 1 shows the computation and values of the first 5 samples in base 2, for which Kollig and Keller [KK02] give an efficient implementation using bit arithmetic.

Table 1: *Computation of the base-2 van der Corput sequence.*

index	base-2 expansion	radical inverse	
0	0	$\frac{0}{2}$	$= 0.0$
1	1	$\frac{1}{2}$	$= 0.5$
2	0,1	$\frac{0}{2} + \frac{1}{4}$	$= 0.25$
3	1,1	$\frac{1}{2} + \frac{1}{4}$	$= 0.75$
4	0,0,1	$\frac{0}{2} + \frac{0}{4} + \frac{1}{8}$	$= 0.125$

To generate x_i, the i'th sample of the sequence, i is first expanded into its base-b digits $d_{0,b}(i), d_{1,b}(i), d_{2,b}(i) \ldots$ such that

$$i = \sum_{m=0}^{\lfloor \log_b(i) \rfloor} d_{m,b}(i) b^m$$

with $d_{m,b}(i) \in \{0, 1, \ldots, b-1\}$ for an integer base $b \geq 2$.

x_i in the base-b van der Corput sequence is then computed as the *radical inverse* of that b-ary expansion:

$$x_i = \sum_{m=0}^{\lfloor \log_b(i) \rfloor} d_{m,b}(i) b^{-m-1},$$

where the digits are flipped around the radix point.

A notable property of the unscrambled van der Corput sequence is that, for $N = b^m$ samples, all sample positions are equidistant from their neighbors. This correlation is not optimal when calculating smooth integrals [Owe13]. For example, when integrating a 1D Gaussian distribution, root mean square error diminishes at $O(N^{-1})$.

3.2. Owen-scrambling

Owen [Owe95] introduced a method to scramble the digits of a sample sequence such as the base-b van der Corput sequence. The interval $[0,1)$ is first divided into b intervals of equal length. These intervals are randomly shuffled. Then each interval is recursively divided into b sub-intervals, those are randomly shuffled as well, and so on. The random shuffles in each sub-interval are independent of shuffles in other sub-intervals; see Figure 2. Because the sample scramblings in each interval are independent, the original fixed distance between adjacent samples is broken up, and this added randomness improves convergence when the samples are used to integrate certain classes of functions. For example, Owen-scrambling the van der Corput sequence improves the error convergence of a Gaussian integral from $O(N^{-1})$ to $O(^{-3/2})$ [Owe13]. Laine and Karras [LK11] and Burley [Bur20] have introduced efficient hash-based implementations of Owen scrambling.

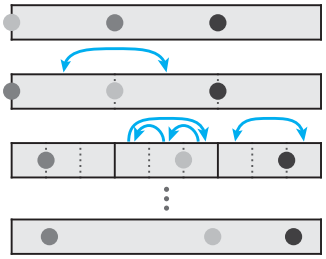


Figure 2: *Owen-scrambling the first three sample points of the base-3 van der Corput sequence. The three sub-intervals within each interval are shuffled randomly and independently.*

3.3. Stochastic generation of one-dimensional sequences

Before introducing the new multidimensional sampling technique, we discuss a stochastic algorithm that is a straightforward adaptation of progressive jittered sequences [Kaj86; CKK18] to one dimension.

The algorithm starts by placing the first sample randomly in the unit interval. The unit interval is then subdivided into two sub-intervals of length $1/2$ and the second sample is placed at a random position in the opposite sub-interval of the first sample.

The two half-intervals are subdivided into four quarter-intervals.

The third sample is placed in the same half-interval as the first sample, but the opposite quarter-interval. The fourth sample is placed in the same way, adjacent to the second sample. This process is illustrated in Figure 3.

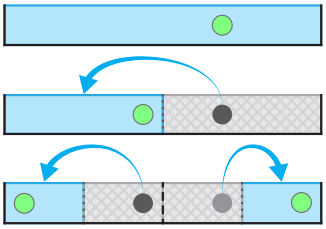


Figure 3: *Stochastic generation of the first four 1D sample points. The valid sub-interval (blue) for each new point is determined by swapping from a previous sample to the adjacent sub-interval.*

For a sub-interval $[\frac{t}{N}, \frac{t+1}{N})$ with $t \in \{0,\dots,N-1\}$, $N \in \{2^1,2^2,2^3,\dots\}$, the adjacent unoccupied sub-interval can be obtained using the bitwise exclusive-or (xor) operator $\oplus_2$ as $[\frac{t\oplus_2 1}{N}, \frac{(t\oplus_2 1)+1}{N})$. We refer to this as *swapping* the sub-intervals or *swapping from* a previous sample point. In Section 4.6 we will generalize swapping to other prime bases.

To further extend the sequence, the unit interval is progressively subdivided after each set of sub-intervals is filled. At each stage the previous samples are iterated over in their original order, and for each previous sample, a new sample point is generated randomly in the adjacent sub-interval. The C++ code for this algorithm is given in Listing 1 using the `drand48()` pseudorandom number generator (PRNG).

We make a new observation about this intuitive algorithm: it generates the Owen-scrambled base-2 van der Corput sequence. If the calls to the PRNG were replaced with the value 0.0, it would generate the unscrambled van der Corput sequence. Rather than generating the quasi-Monte Carlo points and then scrambling them, here the generation and scrambling have been merged into a stratified sample generation algorithm that is considerably faster and shorter than separate implementations.

Listing 1: *C++ code to generate a stratified 1D sequence, identical to the Owen-scrambled base-2 van der Corput sequence.*

```
void get1DSamples(int nSamples, double samples[]) {
  samples[0] = drand48();
  for (int prevLen = 1; prevLen < nSamples; prevLen *= 2) {
    int nStrata = prevLen * 2;
    for (int i = 0; i < prevLen && (prevLen+i) < nSamples; i++) {
      int prevXStratum = samples[i] * nStrata;
      samples[prevLen+i] = ((prevXStratum^1) + drand48()) / nStrata;
    }
  }
}
```

4. Generation of multidimensional sequences

We now move from one dimension to higher dimensions. We start with background on (t,s)-sequences in general and the Sobol' sequence in particular in Sections 4.1 and 4.2, respectively. We will then expand our stochastic generation algorithm to multidimensional base-2 sequences in Sections 4.3 and 4.4, allowing the generation of Owen-scrambled Sobol' sequences. Similarly, we review Halton and Faure sequences in Section 4.5, before showing how our algorithm can generate randomized Halton and Faure sequences in Sections 4.6 and 4.7.

4.1. (t,s)-sequences

The one-dimensional intervals of Section 3 can be generalized to higher dimensions as elementary intervals. For any $m \geq 0$, the s-dimensional unit hypercube $[0,1)^s$ can be divided multiple ways into $N = b^m$ non-overlapping boxes of volume $1/N$. For example, in base $b = 2$ with $m = 3$ and $s = 3$, the unit cube can be partitioned (stratified) into ten grids: 8×1×1, 1×8×1, 1×1×8, 4×2×1, 4×1×2, 2×4×1, 1×4×2, 1×2×4, 2×1×4, and 2×2×2. In the 2×4×1 stratification, each stratum would be a sub-interval

$$\left[\frac{k_0}{2^1}, \frac{k_0+1}{2^1}\right) \times \left[\frac{k_1}{2^2}, \frac{k_1+1}{2^2}\right) \times \left[\frac{0}{2^0}, \frac{1}{2^0}\right),$$

where $k_0 \in \{0,1\}$, $k_1 \in \{0,1,2,3\}$.

These strata of the unit hypercube are the base-b *elementary intervals*, with an elementary interval defined as

$$E = \prod_{d=1}^{s} \left[\frac{k_d}{b^{m_d}}, \frac{k_d+1}{b^{m_d}}\right)$$

for integers $0 \leq k_d < m_d$ and where $\sum_{d=1}^{s} m_d = m$.

Using elementary intervals, Niederreiter [Nie87] introduced the concept of (t,s)-sequences. A base-b (t,s)-sequence is an s-dimensional sample sequence that has at most b^t points in each elementary interval of volume $1/b^{m-t}$, for all disjoint subsequences $\{x_{ib^m},\dots,x_{(i+1)b^m-1}\}$ with integers $i,m \geq 0$. Figure 4 shows 2D elementary interval stratifications for 16 points with different t values.

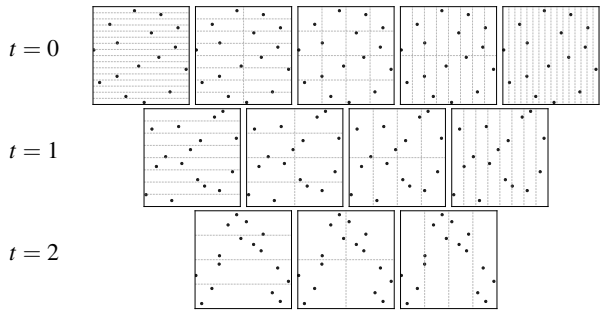


Figure 4: *Elementary interval stratifications of 16 points for base-2 (t,2)-sequences. Each elementary interval has an area of 2^{t-4} and contains 2^t points.*

A t value of zero, i.e., a $(0,s)$-sequence, achieves the ideal stratification, where all elementary intervals have exactly one point in them. Unfortunately it is only possible to have a $(0,s)$-sequence for $s \leq b$, where b is a prime power. For a constant base, the t value must be greater for sequences with more dimensions.

Owen-scrambling can be applied to (t,s)-sequences by scrambling each dimension independently, and this does not affect the t value. Regardless of their t value, Owen-scrambled (t,s)-sequences

achieve asymptotically faster convergence on integrals than uniform random sampling. For sufficiently smooth integrals, the expected root mean square error for N sample points, when N is a power of the base b, diminishes as $O(N^{-3/2}(\log N)^{(s-1)/2})$ compared to $O(N^{-1/2})$ for uniform random sampling [Owe13]. The asymptotic convergence is faster even for discontinuous integrals, e.g. $O(N^{-3/4})$ for 2D integrals [Mit96; CKK18].

(t,s)-sequences are often generated with the *digital method*. The digital method allows for the base b to be a prime power using finite field (Galois field) arithmetic. In this paper, we will focus on finite prime fields with modular arithmetic, although theoretically our contributions can be generalized to any finite field.

An s-dimensional digital sequence is defined by its *generator matrices* $\{\mathbf{C}^{(k)} \mid k \in S\}$, $S = \{0,1,\dots,s-1\}$, with all elements $\mathbf{C}^{(k)}_{ij} \in \{0,1,\dots b-1\}$. To generate $x^{(k)}_j$, the k'th coordinate of the j'th sample, the index j is again expanded into its base-b representation $d_{0,b}(j), d_{1,b}(j), \dots$ up to n digits, typically determined by the machine precision. The digits are treated as a column-vector

$$\mathbf{d}_{j,b} = \begin{pmatrix} d_{0,b}(j) \\ d_{1,b}(j) \\ \vdots \\ d_{n-1,b}(j) \end{pmatrix}.$$

For a given dimension k, the generator matrix $\mathbf{C}^{(k)}$ is multiplied by the digit vector $\mathbf{d}_{j,b}$ to obtain a scrambled column-vector of digits:

$$\mathbf{y}^{(k)}_j = \mathbf{C}^{(k)}\mathbf{d}_{j,b} \mod b.$$

The radical inverse is then applied to these scrambled digits to generate the coordinate $x^{(k)}_j$:

$$x^{(k)}_j = \sum_{i=0}^{n-1} y^{(k)}_{j,i} b^{-i-1}.$$

The t value of a sequence is determined by the selection of generator matrices, where linear independence between leading rows and columns of the matrices ensure certain multidimensional stratifications [DP10]. In sections 4.2 and 4.5 we will describe the Sobol', Halton, and Faure sequences, which all have non-singular upper triangular (NUT) generator matrices. Every dimension of a sequence with NUT matrices is a $(0,1)$-sequence, as the leading k rows and columns form a matrix of rank k for any k. The van der Corput sequence is the simplest $(0,1)$-sequence, where the generator matrix $\mathbf{C}$ is the identity matrix $\mathbf{I}$.

4.2. The Sobol' sequence

The Sobol' sequence [Sob67] is a base-2 (t,s)-sequence popular for multidimensional integration problems. The sequence is constructed using "direction numbers", and it is often generated with the digital method where the binary representation of the direction numbers correspond to columns of the generator matrices. The exact choice and meaning of the direction numbers is beyond the scope of this paper; we refer interested readers to Bratley and Fox [BF88] for the details. We use the direction numbers found by Joe and Kuo [JK08], with the generator matrices provided as part of the source code for the PBRT v3 renderer [PJH17].

The first dimension of the Sobol' sequence is typically chosen to be the base-2 van der Corput sequence, and the generator of the second dimension is the upper-triangular Pascal matrix $\mathbf{P}$ modulo 2 with

$$\mathbf{P} = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots \\ 0 & 1 & 2 & 3 & \dots \\ 0 & 0 & 1 & 3 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad \text{and} \quad \mathbf{P} \bmod 2 = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots \\ 0 & 1 & 0 & 1 & \dots \\ 0 & 0 & 1 & 1 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

This choice of generator matrices makes the first two dimensions a base-2 (0,2)-sequence. These matrices hint at two significant properties of the Sobol' sequence. First, all generator matrices are NUT, so all one-dimensional projections of the sequence are (0,1)-sequences. Second, the t values of the Sobol' sequence are smaller for earlier sets of dimensions in the sequence. This is a useful property in path tracing, where the earlier dimensions generally have a greater weight in the rendering integral. Despite not having minimal t values, these characteristics are part of why the Sobol' sequence has found success in many applications [Owe13]. Base-2 sequences are also appealing because they can be generated efficiently using bit arithmetic for matrix multiplications, and Owen-scrambling can be performed with hashing [LK11; Bur20].

4.3. Stochastic generation of base-2 (t,s)-sequences

The use of generator matrices to scramble a digit vector can be seen as shuffling the coordinates for each dimension. For example, the 3×3 leading matrix of the second Sobol' dimension

$$\mathbf{C}^{(1)} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

will swap the third and fourth point of the van der Corput sequence, and the fifth and the sixth point, as illustrated in Figure 5.

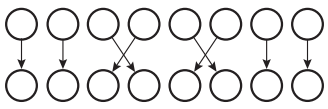


Figure 5: *To generate the Sobol' (0,2)-sequence, the coordinates of the first dimension (top) can be permuted to form the second dimension (bottom).*

This observation inspires our new multidimensional stochastic algorithm. To extend the algorithm from Listing 1 to multiple dimensions, each dimension will permute the index of previous samples used to choose sub-intervals for new samples.

The following example shows how we stochastically generate the first four samples of an Owen-scrambled Sobol' (0,2)-sequence: The first sample is placed randomly in the unit square. Each 1D unit interval is divided into two strata, and in both dimensions the next sample is placed in the opposite 1D stratum of the first point. The unit intervals are again subdivided. The x stratum of the third

sample is swapped from the first sample, while the y stratum is swapped from the second sample. This is illustrated in Figure 6.

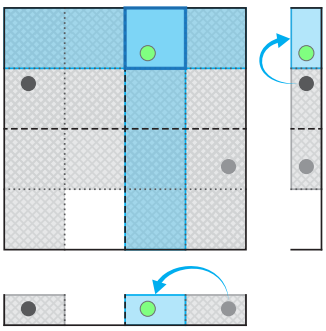


Figure 6: *Generation of the third sample (green) in 2D, swapping the sub-intervals of the first sample in the x-dimension and the second sample in the y-dimension.*

The fourth sample is generated conversely, swapping the x stratum of the second sample and the y stratum of the first, which will place it in the remaining empty cell of Figure 6. By permuting the order of the samples used to select sub-intervals on the y-axis, the four points are guaranteed to be placed in different 2D quadrants as well as different 1D sub-intervals.

Our base-2 algorithm starts by calculating n "xor-values" for each of the s dimensions, $\{\chi_m^{(k)} \mid 0 \le m < n,\ k \in S\}$, in order to generate up to 2^n samples. In Section 4.4, we will show how these xor-values can be derived from the $n \times n$ NUT generator matrices of a QMC sequence.

Given the xor-values, the first sample in the base-2 sequence is generated randomly in the s-dimensional unit hypercube $[0,1)^s$. To extend the sequence from 2^m samples to 2^{m+1} samples, the s one-dimensional unit intervals are each stratified into 2^{m+1} sub-intervals. The indices $i \in \{0,1,\dots,2^m-1\}$ are iterated over, and for each new sample an unoccupied sub-interval in dimension k is selected, with the new coordinate $x_{2^m+i}^{(k)}$ generated randomly within that sub-interval. The unoccupied sub-interval is selected by swapping from a previous coordinate $x_j^{(k)}$, with the m'th xor-value used to calculate the index: $j = i \oplus_2 \chi_m^{(k)}$.

The Owen-scrambled van der Corput sequence is a special case of this algorithm, where the xor-values χ_* are all zero. The second dimension of the scrambled Sobol' (0,2)-sequence can be generated with the xor-values $\chi_*^{(1)} = \{0,1,1,7,1,19,\dots\}$. This yields a new implementation of the Owen-scrambled Sobol' (0,2)-sequence shown in Listing 2, which is both short and remarkably fast. The supplemental code contains a stochastic implementation of the 64-dimensional Sobol' sequence.

In Table 2 we compare the performance against four other methods of generating equivalent or similar (0,2)-sequences:

Hash-based Owen scrambling. Code is provided with the supplemental material of Burley [Bur20] to generate Owen-scrambled Sobol' (0,2)-sequences using Laine-Karras hashing [LK11].

Hash-based Owen scrambling with precomputed Sobol' points.

Listing 2: *Stochastic generation of an Owen-scrambled Sobol' (0,2)-sequence in C++.*

```
void getStochasticSobol02Samples(int nSamples,
                                 double samples[][2]) {
  static constexpr uint32_t yXors[30] =
      {0x00000000, 0x00000001, 0x00000001, 0x00000007,
       0x00000001, 0x00000013, 0x00000015, 0x0000007f,
       0x00000001, 0x00000103, 0x00000105, 0x0000070f,
       0x00000111, 0x00001333, 0x00001555, 0x00007fff,
       0x00000001, 0x00010003, 0x00010005, 0x0007000f,
       0x00010011, 0x00130033, 0x00150055, 0x007f00ff,
       0x00010101, 0x01030303, 0x01050505, 0x070f0f0f,
       0x01111111, 0x13333333};
  samples[0][0] = drand48();
  samples[0][1] = drand48();
  for (int logN = 0, prevLen = 1;
       prevLen < nSamples;
       logN++, prevLen *= 2) {
    int nStrata = prevLen * 2;
    for (int i = 0; i < prevLen && (prevLen+i) < nSamples; i++) {
      // Get strata of previous samples.
      int prevXStratum = samples[i][0] * nStrata;
      int prevYStratum = samples[i^yXors[logN]][1] * nStrata;
      // Generate new sample in adjacent strata.
      samples[prevLen+i][0] =
            ((prevXStratum^1) + drand48()) / nStrata;
      samples[prevLen+i][1] =
            ((prevYStratum^1) + drand48()) / nStrata;
    }
  }
}
```

The majority of computation time with hash-based scrambling is used recalculating the unscrambled values of the Sobol' (0,2)-sequence. For a fixed memory overhead, we precompute these samples and store them as 16-bit integers.

Lazy permutation trees. We use the libseq library [FK01] with the specialized `RandomizedTSSequence_base_2` class and the same generators as the Sobol' (0,2)-sequence.

Efficient generation of (0,2)-sequences. Samples are generated in the same diagonally opposing order as Christensen et al. [CKK18], using the traversal described by Pharr [Pha19]. Optimal disjoint subsequences are maintained implicitly using the strategy of Brown [Bro19], generating true (0,2)-sequences.

Table 2: *Single-threaded performance of generating 65536 samples of a scrambled (0,2)-sequence, measured on a 2.9 GHz Intel Core i7 (average time over 1024 runs). Memory and time for lazy permutation trees were measured using the libseq library [FK01].*

		memory usage	
algorithm	time (samples/sec)	per sequence	constant
Hashing [LK11; Bur20]	1.6 ms (40M)	< 1 KB	< 1 KB
Hashing precomputed Sobol' points	0.6 ms (103M)	< 1 KB	262 KB
Lazy permutation trees [FK02]	2.5 ms (26M)	~2,200 KB	< 1 KB
Efficient generation [CKK18; Pha19]	40 ms (1.66M)	1,065 KB	< 1 KB
Stochastic generation (ours)	0.30 ms (215M)	1,048 KB	< 1 KB

Table 2 shows that stochastic generation is faster than any other technique, and two orders of magnitude faster than Pharr [Pha19], the only other stochastic generation algorithm. Hashing may still be preferable for many scenarios due to its low memory footprint. We applied a few obvious optimizations to each technique; these are detailed in the supplemental material.

4.4. Calculation of xor-values

The xor-values used in our stochastic algorithm are, in a sense, a reversal of the generator matrices. We start by defining a function

$$m_b(i) = \lfloor \log_b(i) \rfloor$$

that gives the index of the most significant digit of i in a prime base b, shortened to $m(i)$ when discussing a base b sequence. For a sample i in base b with digital expansion vector $\mathbf{d}_i$, we would like to find an earlier sample j to swap from, such that

$$\mathbf{C}\mathbf{d}_j + \mathbf{e}_{m(i)} = \mathbf{C}\mathbf{d}_i \mod b,$$

with

$$e_{m(i),k} = \begin{cases} 1 & \text{if } m(i) = k \\ 0 & \text{otherwise,} \end{cases}$$

for integers $k \geq 0$. The sample index j can then be calculated with

$$\mathbf{d}_j = \mathbf{d}_i - \mathbf{c}^{-1}_{*,m(i)} \mod b, \tag{1}$$

where $\mathbf{c}^{-1}_{*,m(i)}$ is the $m(i)$'th column of the inverse of the generator matrix. However the stochastic generation algorithm iterates over the previous indices of the sequence, rather than using the new index of each sample, so the most significant digit needs to be truncated. We define a truncation operator that, in base 2, sets the last non-zero digit to zero:

$$\tau(\mathbf{d}_i) = \mathbf{d}_i - \mathbf{e}_{m(i)} \mod b.$$

We can apply τ to each of the right-hand terms in Equation 1:

$$\mathbf{d}_j = \tau(\mathbf{d}_i) - \tau(\mathbf{c}^{-1}_{*,m(i)}) \mod b.$$

In base 2, vector addition mod 2 is equivalent to an element-wise xor, hence the last term is the negated b-ary expansion of the xor-value:

$$\chi_m = \sum_{i=0}^{m} (-\tau(\mathbf{c}^{-1}_{*,m})_i \bmod b) b^i \tag{2}$$

In summary, to compute n xor-values, the $n \times n$ generator matrix $\mathbf{C}$ is inverted, one is subtracted from the diagonal entries, and the negated m'th column of the resulting matrix (modulo the base) is multiplied by powers of b and summed to calculate χ_m. Python code to generate these values is given in Listing 3.

Listing 3: *Computation of the xor-values from a generator matrix in Python.*

```
import numpy as np
def get_xor_values(gen_matrix, base=2):
  # Invert the matrix.
  m_inv = np.linalg.inv(gen_matrix).astype(int) % base
  # Truncate the diagonal.
  m_inv -= np.identity(m_inv.shape[0], dtype=int)

  # Compute the xor-values from the negated columns.
  # For base 2, base_pow = (1,2,4,8,16...)
  base_pow = np.power(base, np.arange(0, gen_matrix.shape[0]))
  return [np.sum((-col % base)*base_pow) for col in m_inv.T]
```

4.5. The Halton and Faure sequences

The Halton sequence [Hal60] is a multidimensional sequence where each dimension is a van der Corput sequence in a different base, with all bases coprime. Typically the bases are chosen to be successive primes, i.e., the first dimension is in base 2, the second dimension base 3, base 5, etc. The sequence is sometimes referenced using the bases, for example "the Halton 2,3" sequence is the 2D sequence with bases 2 and 3 for the x and y dimensions, respectively.

The Halton sequence is not strictly a (t,s)-sequence, although each dimension is a (0,1)-sequence, and the Owen-scrambled Halton sequence does not have the same asymptotic convergence on smooth integrals. However, it has a number of appealing properties. All subsequences of the Halton sequence are well-stratified, not only disjoint ones. This is useful for temporal antialiasing [YLS20], where occlusions may cause earlier samples to be discarded. The Halton sequence also gives high-dimensional stratification for all lower dimensional projections. For a given set of s co-prime bases $\{b_0, b_1, \ldots, b_{s-1}\}$, any subsequence of $N = \prod_{i=0}^{s-1} b_i^{m_i}$ samples will be stratified in the $b_0^{m_0} \times b_1^{m_1} \times \cdots \times b_{s-1}^{m_{s-1}}$ grid. Figure 1b shows some of these stratifications for a Halton 2,3 sequence. Finally, the Owen-scrambled Halton sequence has the property that projections of higher dimensions have distributions similar to uniform random samples at low sample counts. This is unlike the Sobol' sequence, which can have harmful correlations between dimensions.

Faure [Fau82] described a general technique to construct a $(0,s)$-sequence in a prime base $b \geq s$, equivalent to the Sobol' (0,2)-sequence for $s = b = 2$. The generator matrices $\{\mathbf{C}^{(k)} \mid k \in S\}$ are constructed using upper-triangular Pascal matrices taken to the power of their dimension:

$$\mathbf{C}^{(k)} = \mathbf{P}^k \mod b,$$

where $\mathbf{P}^0 = \mathbf{I}$, $\mathbf{P}^k = \Pi_{i=1}^{k} \mathbf{P}$ for $k \geq 1$, and

$$\mathbf{P} = \begin{pmatrix} \rho_{0,0} & \rho_{0,1} & \rho_{0,2} & \cdots \\ \rho_{1,0} & \rho_{1,1} & \rho_{1,2} & \cdots \\ \rho_{2,0} & \rho_{2,1} & \rho_{2,2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix},$$

with

$$\rho_{i,j} = \begin{cases} 1 & \text{if } i = 0 \text{ or } i = j \\ 0 & \text{if } i > j \\ \rho_{i,(j-1)} + \rho_{(i-1),j} & \text{otherwise.} \end{cases}$$

The base b is often assumed to be the smallest prime greater than or equal to s, and we say the Faure $(0,5)$-sequence in short for the base-5 Faure $(0,5)$-sequence. Dick and Pillichshammer [DP10] provide a proof that using the Pascal matrices gives $(0,s)$-sequences.

The progressive stratifications of a $(0,s)$-sequence ensure that all lower-dimensional projections of the sequence are well-stratified, themselves $(0,s)$-sequences, which is a valuable characteristic for rendering integrals [JEK*19]. Figure 7 shows the stratifications for just 27 points of a scrambled Faure (0,3)-sequence. A drawback of Faure sequences is that many samples are needed to fill out the stratifications, especially for high dimensional sequences. They also only achieve optimal error convergence at powers of the base [Owe13], which can be prohibitively far apart for high bases.

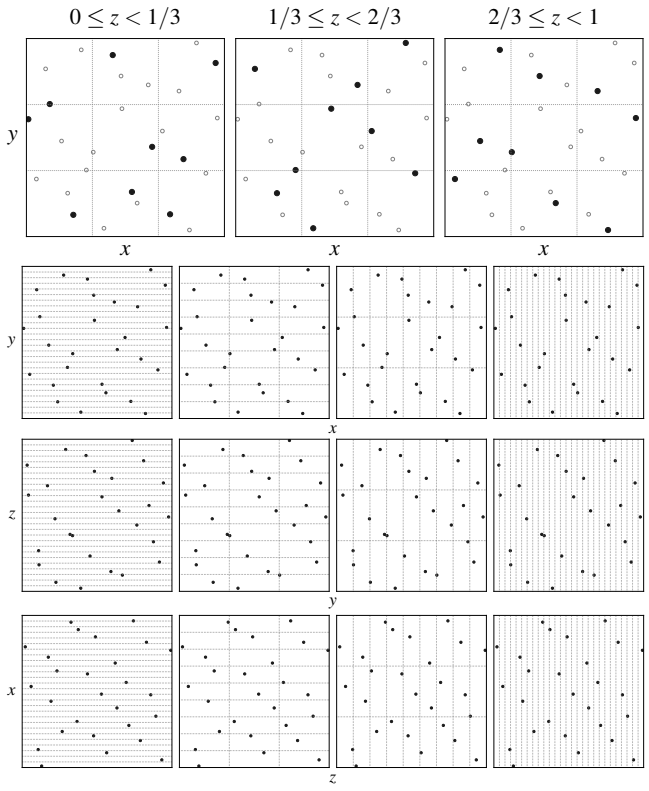


Figure 7: *The first 27 samples of a scrambled Faure (0,3)-sequence, stratified for all base-3 elementary intervals. Top row: stratification in the 3×3×3 grid. Bottom 3 rows: all 2D projections stratified in 27×1, 9×3, 3×9, and 1×27.*

4.6. Stochastic generation in prime bases

The new stochastic generation algorithm from Section 4.3 can be extended to prime bases with three generalizations, which will enable the generation of Owen-scrambled Halton and Faure sequences.

First, when the sequence is extended from b^m to b^{m+1} samples, rather than iterating over the previous indices once, they need to be iterated over $b-1$ times. Each of these iterations is a *pass* with an index $1 \le p < b$.

The second change involves adapting the meaning and use of "xor-values". Equation 2 and Listing 3 can be used for prime-base NUT matrices with ones on the diagonal, such as the Halton and Faure sequences, and a more general form can be derived for matrices with larger values on the diagonal. We note that our derivation of xor-values actually references samples from the previous *pass*, not from the previous power, so the inner loop must permute sample indices from the previous pass. Alternatively, one could multiply the digits of the xor-values by p to reference samples from the previous power.

The bitwise xor operator $i \oplus_2 j$ is generalized to $i \oplus_b j$ as a vector addition on the base-b digits of i and j, modulo the base (i.e., as carryless addition). Using template metaprogramming, compilers can efficiently optimize the integer divisions by the base [PJH17], without storing an array of digits. Listing 4 shows an implementation of this function in C++.

Listing 4: *Add the digits of x and y in a prime base.*

```
1  template<unsigned BASE>
2  unsigned carrylessAdd(unsigned x, unsigned y) {
3    unsigned sum = 0, bPow = 1;
4    while (y > 0 && x > 0) {
5      sum += ((x + y) % BASE) * bPow;
6      x /= BASE; y /= BASE;
7      bPow *= BASE;
8    }
9    return sum + bPow*(x+y);
10 }
```

Adding the digits in higher bases will have $O(\log_b N)$ time complexity to generate each new coordinate, rather than the $O(1)$ complexity of the base-2 algorithm. However, the results of adding the digits of each sample index with the corresponding χ_m are fixed for a given generator matrix, so they can be precomputed and stored in an "index map".

Finally, strata swapping must be generalized to higher bases. When the sequence is extended from b^m to b^{m+1} points, the interval containing x_j with $0 \le j < b^m$ is subdivided into b sub-intervals. We now have multiple choices of strata for the new samples, as shown in Figure 8.

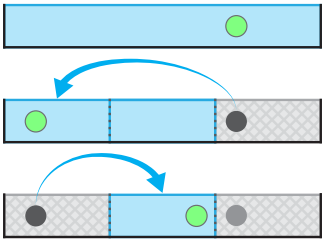


Figure 8: *Stochastic generation of the 1D coordinates in base 3. The second sample can be placed in either of the two unoccupied sub-intervals, and the third sample is placed in the remaining one.*

We address this by randomly assigning *strata offsets* $\{\delta_p\} = \{1,\dots,b-1\}$ to each of the $b-1$ new samples within an interval. We define $t(j) = \left\lfloor x_j b^{m+1} \right\rfloor$, the index of the occupied sub-interval of coordinate x_j. Each new coordinate x_i in pass p, where $i = j \oplus_b p(b^{m+1} \ominus_b \chi_{m+1})$, is placed randomly in the sub-interval $[\frac{t(i)}{b^{m+1}}, \frac{t(i)+1}{b^{m+1}})$, with $t(i) = t(j) \oplus_b \delta_p$.

With this generalization, we can stochastically generate Owen-scrambled Sobol', Halton, and Faure sequences. Choosing the best multidimensional sequence for rendering integrals remains an open area of experimentation and research. Figure 9 shows a common 5D integral in path tracing, with BRDF sampling, area light sampling, and light selection. We compare de-correlated scrambled Sobol' (0,2)-sequences (ssobol02), scrambled 5D Sobol' sequences (ssobol5D), and scrambled Faure (0,5)-sequences (sfaure05), using correlated swapping (Section 4.7). The Faure (0,5)-sequence yields the lowest overall error, but not in the blue inset. In the supplemental material we compare different sequences, including some shipped with PBRT [PJH17], in complex high-dimensional renders.

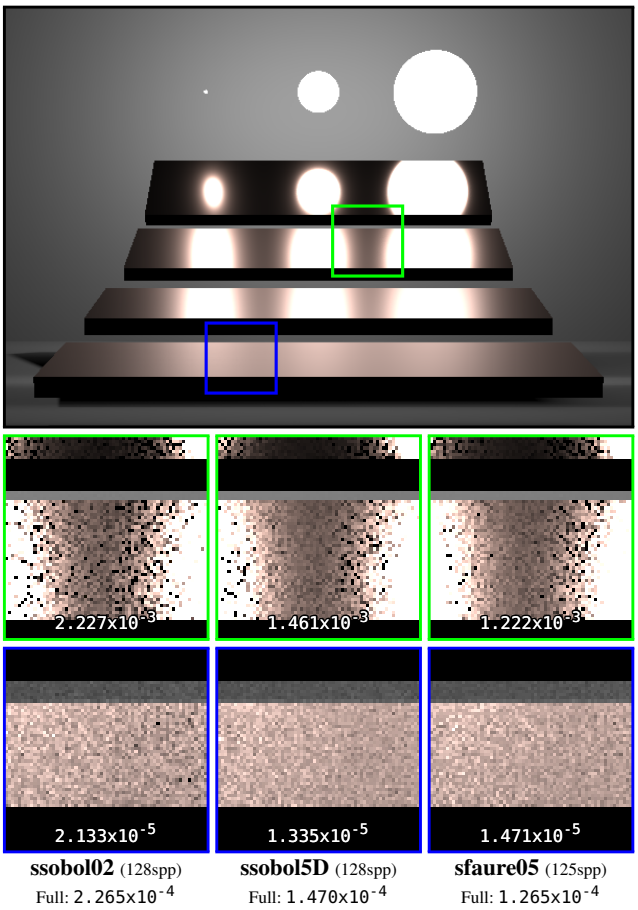
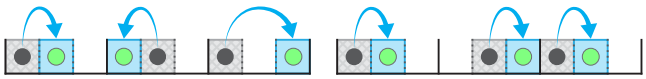


Figure 9: *A 5D scene from [Bit16; VG95] demonstrating multiple importance sampling with light selection, area light sampling, and BRDF sampling, rendered with path tracing in PBRT v3 [PJH17]. Both 5D sequences have lower RMS error than shuffled and scrambled (0,2)-sequences, with the stochastic Faure (0,5)-sequence performing the best overall. Differences from the reference image are amplified in the insets by 5× (top row) and 20× (bottom row).*

4.7. Correlated swapping

Keeping track of the shuffled strata offsets for each interval adds the most complexity to the base-2 algorithm. Inspired by Kensler's correlated shuffling [Ken13], *correlated swapping* simplifies the process. While full Owen-scrambling independently assigns a strata offset to every new sample, correlated swapping uses the same strata offset for all the samples in a given pass and dimension. This is illustrated in Figure 10b, where $\delta_1 = 1$ for samples 3-5, and δ_2 must be 2 for samples 5-8.

Different strata offsets are still used for each dimension, and the offsets are shuffled for each new power of the base. This simplification slightly reduces the randomness, but it is considerably more random than other scrambling techniques such as random digit scrambling [Mat98], since the previous points may have been placed in differently offset sub-intervals. Unlike correlated shuffling, it does not improve the point distributions, but error on test



(a) *Owen-scrambling* **(b)** *Correlated swapping*

Figure 10: *Generating the 4th, 5th, and 6th coordinates in a base-3 sequence. In complete Owen-scrambling the strata offsets from earlier samples are chosen independently, while correlated swapping uses the same offsets.*

integrals is unaffected, and the error differences we observed in renders were minor (typically a 2% difference in RMS error), with neither swapping method consistently better than the other.

Table 3 compares the performance of our stochastic generation with two other methods for generating a scrambled Faure (0,5)-sequence. We use the libseq [FK01] library for lazy permutation trees, and we use Burley's implementation of base-5 Owen-scrambling [Bur20] that hashes each base-b digit, with the hash randomized by more significant digits.

Table 3: *Single-threaded performance of generating 3125 samples of a scrambled Faure (0,5)-sequence, measured on a 2.9 GHz Intel Core i7 (average time over 1024 runs). Memory and time for lazy permutation trees were measured using the libseq library [FK01].*

		memory usage	
algorithm	time (samples/sec)	per sequence	constant
Stateless permutations [Bur20]	3.3 ms (0.95M)	< 1 KB	< 1 KB
Lazy permutation trees [FK02]	1.06 ms (2.9M)	~266 KB	< 1 KB
Stochastic generation (ours)			
no map, uncorrelated swapping	0.53 ms (5.9M)	138 KB	< 1 KB
index map, uncorrelated swapping	0.39 ms (8M)	138 KB	62 KB
no map, correlated swapping	0.23 ms (13.4M)	125 KB	< 1 KB
index map, correlated swapping	0.08 ms (39.5M)	125 KB	62 KB

Even without index mapping or correlated swapping, stochastic generation is the fastest technique. With both it is over an order of magnitude faster than either hashing or lazy permutation trees. A general implementation of prime base sequences with correlated swapping is provided in our supplemental code, as well as xor-values for some Faure sequences.

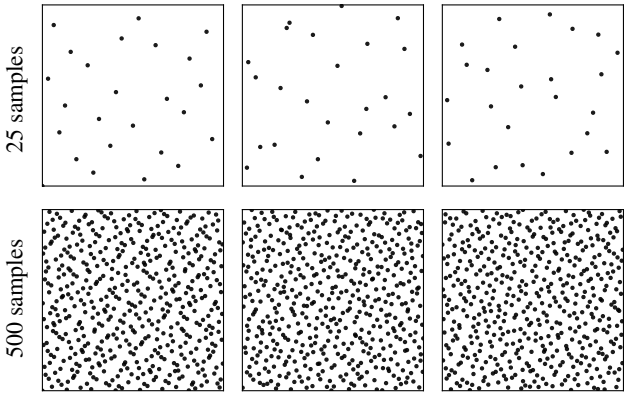
5. Application and variants

We now discuss the application of these sequences to rendering.

5.1. Best-candidate sampling

In addition to the improved performance and simplicity of stochastic generation, the approach of stratified sampling allows for optimization of the point sequences. Inspired by the pmj02bn sequences [CKK18], we can generate multiple candidates in the valid strata and choose the candidate with the highest minimum distance from previous points [Mit91]. This allows us to generate ssobol02bn sequences, shalton23bn sequences, or Faure-bn sequences such as the sfaure03bn sequence in Figure 1c. The unscrambled Halton 2,3 sequence has good point spacing, however

higher dimensions have poor correlations, and scrambling is necessary. Best-candidate sampling allows for a new trade-off between those properties, as shown in Figure 11 and in the supplemental materials.



(a) *Unscrambled* **(b)** *Stochastic* **(c)** *Best-candidate*

Figure 11: *Comparison of Halton 2,3 sequences generated with different scrambling methods. Scrambling breaks apart correlations between higher dimensions (see the supplemental materials), but samples can end up closer together; best-candidate samples improve the spacing.*

Figure 12 compares stochastic Sobol' (0,2)-sequences with and without best-candidates for disk light sampling. In this scene, best-candidate samples reduce error across all tested sample counts. Figure 13 compares the sequences on simple test integrals and finds that best-candidate samples are helpful on symmetric integrals, but slightly harmful on the asymmetric Gaussian integral. We find, similar to Christensen et al. [CKK18], that the spectral properties of these sequences are not improved, and more investigation is needed to determine when best-candidate samples are useful.

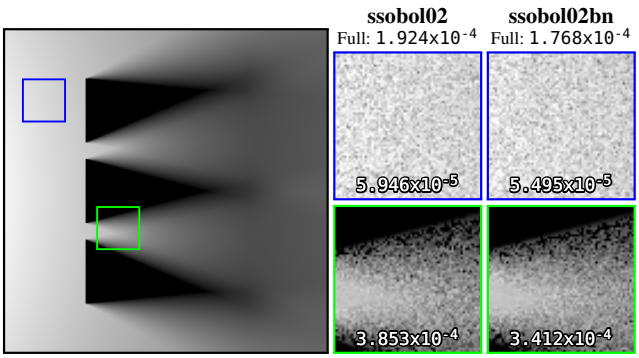


Figure 12: *A scene with a disk area light path-traced at 16spp in PBRT v3. The ssobol02bn sequence has a lower RMS error in the full image, in the unoccluded area, and in the penumbra. Differences from the reference image are amplified in the insets by* $20\times$ *(top row) and* $5\times$ *(bottom row).*

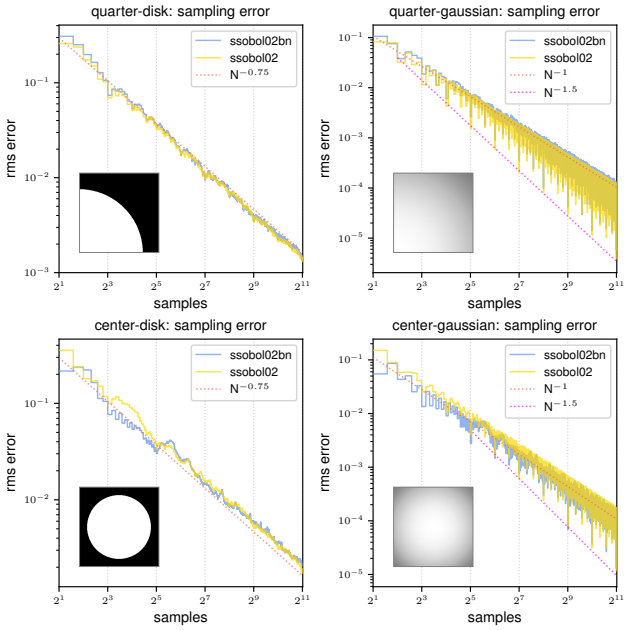


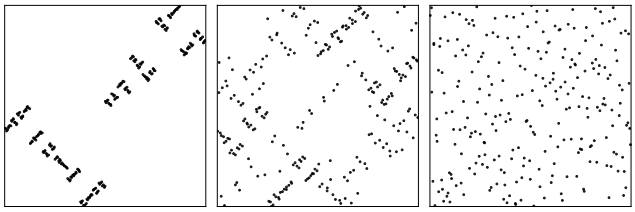
Figure 13: *Comparison of average integration error for the stochastic Sobol' (0,2)-sequence, with and without best-candidate sampling, for 256 independently generated sequences. Best-candidate sampling has lower error on symmetric integrals (bottom row) for most* $N \leq 32$ *and higher error integrating the asymmetric Gaussian function. Asymptotic convergence (slope) is unaffected.*

5.2. Decorrelation of sequences

Although (t,s)-sequences have better asymptotic convergence on integrals than uniform random sampling, high t values can have worse error for practical sample counts. Christensen et al. [CFS*18] observed this for pairs of higher dimensions of the Sobol' sequence. The Owen-scrambled Halton sequence does not have detrimental correlations in higher dimensions, but still degenerates quickly to purely random sampling even for adjacent dimensions. Scrambled Faure sequences have finite dimensionality, and a high-base Faure sequence takes too many samples to fill up the high dimensional space.

These reasons favor using juxtapositions of lower-dimensional sequences, rather than one high-dimensional sequence per pixel. However, juxtaposing lower dimensional (t,s)-sequences results in such harmful correlations between dimensions, as shown in Figure 14a, that the integral may never converge to the correct value. Instead, sequences need to be decorrelated and then juxtaposed, a technique known as *padding*. At the same time, we want to maintain desirable progressive qualities of the sequence, such as optimized minimum distances. In this section, we describe two progressive decorrelation techniques.

Faure-Tezuka scrambling [FT02] progressively shuffles sample indices by scrambling the generator matrices. Given a set of generator matrices $\{\mathbf{C}^{(k)} \mid k \in S\}$, we can multiply all generator matrices by one NUT scrambling matrix $\mathbf{X}$, such that the generator matrices become $\{\mathbf{C}^{(0)}\mathbf{X}, \mathbf{C}^{(1)}\mathbf{X}, \ldots, \mathbf{C}^{(s-1)}\mathbf{X}\}$. Sequences can be partially



(a) *No shuffling* **(b)** *Faure-Tezuka* **(c)** *Full progressive*

Figure 14: *Juxtapositions of two 1D sequences generated with Listing 1 and different types of shuffling.*

decorrelated with random NUT scrambling matrices. NUT scrambling matrices preserve any power-of-b prefix of the samples, maintaining progressive qualities. The full progressive shuffling we will describe is better for padding multiple dimensions in path tracing, but Faure-Tezuka scrambling may be sufficient in other situations.

Some scrambling matrices also have interesting properties. The generation of pmj02 sequences as described by Christensen et al. [CKK18] corresponds to a particular type of scrambling matrix. In the supplemental material we give a construction of pmj02 sequences using generator matrices [HP11].

Faure and Tezuka also proposed more extensive decorrelation by randomly permuting disjoint subsequences. This is equivalent to Owen-scrambling the sample indices within the array, maintaining progressive stratification properties, and it can also be done efficiently with hashing in base 2 [Bur20].

However, this breaks progressively optimized point qualities. We address this with a new progressive shuffling that has equivalent decorrelation. For a base-2 sequence of only two samples, swapping the first two samples is unnecessary. Similarly, for sequences of length $N = 4$, there is no added decorrelation from swapping the initial two points with the latter two. Full decorrelation can be achieved by randomly swapping only the third and the fourth point. These observations were also made by Ahmed and Wonka [AW21], who noted that scrambling the indices is partially redundant with scrambling the coordinates.

In general, when extending a sequence from b^m to b^{m+1} points, we only need to shuffle within each pass of b^m points, but the passes do not need to be shuffled with each other. We refer to this new technique as *full progressive* shuffling, shown in Figure 15. This technique also preserves any power-of-b prefix of samples from the original sequence. Figure 14 compares the decorrelation of full progressive shuffling with no shuffling and Faure-Tezuka scrambling.

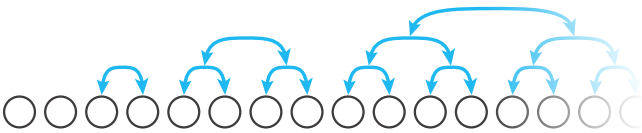


Figure 15: *Base-2 progressive shuffling of sample indices, with arrows representing potential permutations of samples. Full progressive shuffling will randomly decide every permutation independently.*

Progressive shuffling can be performed during stochastic generation using a lookup table. The order of new points is shuffled for each extension of the sequence. When a new point is generated, the lookup table will store its actual index at the location of the original unpermuted index, and when we reference a previous point to swap from, we find its permuted index using the lookup table. Progressive shuffling can also be done during rendering by storing shuffled index arrays separately from the original sequences, allowing a renderer to effectively multiply the number of available sequences with an additional array lookup per sample.

5.3. Stateless stochastic generation

Our algorithm can also be recast in a stateless implementation, trading computation time to eliminate the need to store previous samples. In the sequential algorithm, each coordinate for a sample depends on only two dynamic values: a uniformly distributed random number, and the coordinate (in the same dimension) for a previous sample. The computation of the coordinates for the different dimensions is separable.

Instead of a sequential random number generator, we can use one that hashes the current sample index and dimension together with a seed value, as in Kensler's `randfloat()` function [Ken13] or an alternative [JO20]. This lets us repeatably and deterministically reproduce the random number for a given sample and dimension at any time without the need to traverse the whole random number sequence. The coordinate of the previous sample, and the sample that one depends on, can then be regenerated recursively, all the way back to the first sample; see Figure 16. We provide an implementation of this stateless recursive algorithm for the Owen-scrambled Sobol' (0,2)-sequence in Listing 5.

Figure 16: *Stateless generation of the 8th y-coordinate of the Sobol' $(0,2)$-sequence. The coordinates of the 3rd, 2nd, and 1st points are recursively generated.*

This recursive algorithm can also be implemented as a backwards iteration, generating the random value for each coordinate and then dividing (or right shifting) to "carry" it to the earlier coordinate, which reduces the memory complexity from $O(\log n)$ to $O(1)$. An iterative C++ implementation is provided in the supplemental code. The iterative implementation generates 65536 samples of a scrambled Sobol' (0,2)-sequence in 6.75ms, at 9.7M samples/second.

While this is slower than hashing, it allows a renderer to make new tradeoffs between memory, speed, and quality. A renderer can store a smaller precomputed table of samples, perhaps with best-candidates, and stateless generation could extend to higher sample counts if necessary. This is similar to the suggestion of Matoušek [Mat98] for memory-constrained Owen-scrambling. However our backwards algorithm uses $O(\log \frac{n}{k})$ time to query an arbitrary coordinate, where k is the size of the precomputed table, avoiding the $O(\log n)$ for digital matrix multiplication. This stateless algorithm is also trivially parallelizable, as every sample can be generated independently.

Listing 5: *Stateless generation of one dimension of a single Owen-scrambled Sobol' (0,2) sample in C++.*

```
double hashToRnd(uint32_t idx, uint32_t seed); // See [Ken13] Lst 4
double getSobol02Stateless(int idx, int dim, uint32_t seed) {
  static uint32_t xors[2][30] =
    {{0}, // First dimension special case, xor-values all zero.
     {0x00000000, 0x00000001, 0x00000001, 0x00000007, 0x00000001,
      0x00000013, 0x00000015, 0x0000007f, 0x00000001, 0x00000103,
      0x00000105, 0x0000070f, 0x00000111, 0x00001333, 0x00001555,
      0x00007fff, 0x00000001, 0x00010003, 0x00010005, 0x0007000f,
      0x00010011, 0x00130033, 0x00150055, 0x007f00ff, 0x00010101,
      0x01030303, 0x01050505, 0x070f0f0f, 0x01111111, 0x13333333}};
  // Base case, return first randomly placed point.
  if (!idx)
    return hashToRnd(dim, seed);
  // Determine stratum size and place in previous strata.
  int logN = getMSB(idx); // (Right-most bit is numbered zero)
  int prevLen = 1 << logN;
  int nStrata = prevLen * 2;
  int i = idx - prevLen;
  // Recursively get stratum of previous sample.
  int prevStratum =
    getSobol02Stateless(i^xors[dim][logN], dim, seed) * nStrata;
  // Generate new sample in adjacent stratum.
  return ((prevStratum^1) + hashToRnd(idx*2+dim, seed)) / nStrata;
}
```

6. Optimization of sample positions

The simplicity and continuous formulation of our technique opens the door to new optimizations of stratified sample sequences. Although the strata swapping is discontinuous, the location of samples is locally differentiable with respect to each random number, which means that sequences can be optimized using subgradient descent algorithms. As an experiment, we implemented a stochastic Sobol' (0,2)-sequence in Python with the JAX auto-differentiation library and optimized it using Adam [KB17]. Our loss function L optimizes progressive minimum distances across all sample counts:

$$L = -\sum_{i=2}^{n} \log(\text{min\_dist}(i)) * (\log(i) - \log(i-1)),$$

where min_dist(i) is the minimum toroidal distance between any of the first i points. Conceptually, this is minimizing the area above the curve on a log-log plot of distances vs. number of points.

This optimization will frequently settle on regular structures similar to rank-1 lattices at $N = 8$ and $N = 32$, even when the optimized sequence is much longer. Figure 17 shows the first 128 points of one of these sequences. At 8 and 32 points, the minimum distances are 0.354 and 0.167, nearly identical to the optimal matrix constructions of point sets found by Grünschloß et al. [GHSK08].

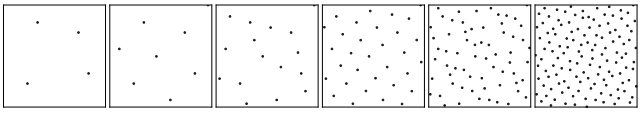


Figure 17: *A Sobol' (0,2)-sequence optimized for minimum point distances using the Adam optimizer.*

The optimization does not generate regular structures or fully optimal distances at higher N, but the distances still improve considerably over best-candidate samples. The regular structure (at low sample counts) may be undesirable for rendering. Either way, this example demonstrates the viability of this optimization approach. Penalizing regularity with arbitrary-edge discrepancy [DEM96] may improve distributions for rendering.

For higher sample counts, global gradient descent can be unstable due to the discontinuities and acceptance of every move. Our current work in progress using simulated annealing with Ahmed and Wonka's formulation of blue-noise [AW21] produces sequences with improved spectra across multiple sample counts.

Beyond optimization, it may be worthwhile to design new QMC constructions tailored specifically to rendering. Christensen et al. [CKK18] shuffle $(0,2)$-sequences together to get sequences that are jittered in 3D, 4D, or 5D. Generator matrix constructions for these sequences should be possible, and they may perform better than a Sobol' sequence for common rendering integrals. Since our technique allows for sequences that have not been thoroughly explored in rendering, such as Owen-scrambled Halton and Faure sequences, there is also new opportunity for investigation into choosing the best sequences for a given render.

7. Conclusion

The main contribution of this paper is a new stochastic approach to generating Owen-scrambled sequences. This technique connects randomized QMC sequences with stratified sampling. Stochastic generation is faster and easier to implement, provides a differentiable parameterization of Owen-scrambled sequences, and extends state-of-the-art stratified sample sequences to higher dimensions. We have also discussed methods to apply these sequences to rendering and to further improve the quality of these sequences through best-candidate sampling or optimization. Our hope is that these techniques are immediately useful to rendering practitioners and provide a stepping stone to further research of multidimensional sample sequences.

Acknowledgements

We want to thank our anonymous peer reviewers for their helpful feedback. We would also like to thank the RenderMan team at Pixar and the Scout simulation team at Amazon. Andrew Helmer would like to thank his brother Edmund for reviewing an early draft of the paper. And finally, we would like to give a special thanks to our partners/wives for supporting the work on this research, especially while being stuck at home for the past year.

References

[AKI10] ATANASSOV, E., KARAIVANOVA, A., and IVANOVSKA, S. "Tuning the generation of Sobol sequence with Owen scrambling". *Large-Scale Scientific Computing*. Springer, 2010, 459–466 3.

[APC*16] AHMED, A., PERRIER, H., COEURJOLLY, D., et al. "Low-discrepancy blue noise sampling". *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 35.6 (2016) 3.

[AW21] AHMED, A. and WONKA, P. "Optimizing dyadic nets". *ACM Transactions on Graphics (Proc. SIGGRAPH)* 40.4 (2021). (To appear.) 3, 11, 12.

[BF88] BRATLEY, P. and FOX, B. "Algorithm 659: implementing Sobol's quasirandom sequence generator". *ACM Transactions on Mathematical Software* 14.1 (1988), 88–100 5.

[Bit16] BITTERLI, B. *Rendering Resources*. https://benedikt-bitterli.me/resources/. 2016 9.

[Bro19] BROWN, S. *Progressive Multi-Jittered Sample Sequences*. https://github.com/sjb3d/pmj. 2019 6.

[Bur20] BURLEY, B. "Practical hash-based Owen scrambling". *Journal of Computer Graphics Techniques* 10.4 (2020), 1–20 2, 3, 5, 6, 9, 11.

[CFS*18] CHRISTENSEN, P., FONG, J., SHADE, J., et al. "RenderMan: an advanced path tracing architecture for movie rendering". *ACM Transactions on Graphics* 37.3 (2018) 1, 2, 10.

[CKK18] CHRISTENSEN, P., KENSLER, A., and KILPATRICK, C. "Progressive multi-jittered sample sequences". *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering)* 37.4 (2018), 21–33 2, 3, 5, 6, 9–12.

[Coo86] COOK, R. "Stochastic sampling in computer graphics". *ACM Transactions on Graphics* 5.1 (1986), 51–72 2.

[Cor35] VAN DER CORPUT, J. "Verteilungsfunktionen. I." *Proc. Akademie van Wetenshappen te Amsterdam* 38 (1935), 813–821 2, 3.

[CSW94] CHIU, K., SHIRLEY, P., and WANG, C. "Multi-jittered sampling". *Graphics Gems IV*. Academic Press, 1994, 370–374 2.

[DEM96] DOBKIN, D., EPPSTEIN, D., and MITCHELL, D. "Computing the discrepancy with applications to supersampling patterns". *ACM Transactions on Graphics* 15.4 (1996), 354–376 12.

[DP10] DICK, J. and PILLICHSHAMMER, F. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration*. Cambridge University Press, 2010 2, 5, 7.

[Fau82] FAURE, H. "Discrépance de suites associées à un système de numeration". *Acta Arithmetica* 41 (1982), 337–351 2, 7.

[FK01] FRIEDEL, I. and KELLER, A. *libseq*. http://www.multires.caltech.edu/software/libseq/. 2001 6, 9.

[FK02] FRIEDEL, I. and KELLER, A. "Fast generation of randomized low-discrepancy point sets". *Monte Carlo and Quasi-Monte Carlo Methods 2000*. Ed. by FANG, K.-T., NIEDERREITER, H., and HICKERNELL, F. Springer, 2002, 257–273 2, 3, 6, 9.

[FT02] FAURE, H. and TEZUKA, S. "Another random scrambling of digital (t,s)-sequences". *Monte Carlo and Quasi-Monte Carlo Methods 2000*. Ed. by FANG, K.-T., NIEDERREITER, H., and HICKERNELL, F. Springer, 2002, 242–256 10.

[GHSK08] GRÜNSCHLOSS, L., HANIKA, J., SCHWEDE, R., and KELLER, A. "(t,m,s)-nets and maximized minimum distance". *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Ed. by KELLER, A., HEINRICH, S., and NIEDERREITER, H. Springer, 2008, 397–412 12.

[Hal60] HALTON, J. "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals". *Numerische Mathematik* 2.1 (1960), 84–90 2, 7.

[HP11] HOFER, R. and PIRSIC, G. "An explicit construction of finite-row digital (0,s)-sequences". *Uniform Distribution Theory* 6.2 (2011), 13–30 11.

[JEK*19] JAROSZ, W., ENAYET, A., KENSLER, A., et al. "Orthogonal array sampling for Monte Carlo rendering". *Computer Graphics Forum (Proceedings of EGSR)* 38.4 (2019), 135–147 2, 7.

[JK08] JOE, S. and KUO, F. "Constructing Sobol' sequences with better two-dimensional projections". *SIAM Journal on Scientific Computation* 30 (2008), 2635–2654 5.

[JO20] JARZYNSKI, M. and OLANO, M. "Hash functions for GPU rendering". *Journal of Computer Graphics Techniques* 9.3 (2020), 20–38 11.

[KAC*19] KELLER, A., AHMED, A., CHRISTENSEN, P., et al. "My favorite samples". *SIGGRAPH Course Notes*. ACM, 2019 2.

[Kaj86] KAJIYA, J. "The rendering equation". *Computer Graphics (Proc. SIGGRAPH)* 20.4 (1986), 143–150 2, 3.

[KB17] KINGMA, D. and BA, J. *Adam: A Method for Stochastic Optimization*. arXiv 1412.6980. 2017 12.

[KDS20] KEROS, A. D., DIVAKARAN, D., and SUBR, K. *Jittering samples using a kd-tree stratification*. https://arxiv.org/abs/2002.07002. 2020 2.

[Ken13] KENSLER, A. *Correlated multi-jittered sampling*. Tech. rep. 13-01. Pixar Animation Studios, 2013 2, 9, 11.

[KK02] KOLLIG, T. and KELLER, A. "Efficient multidimensional sampling". *Computer Graphics Forum (Proc. Eurographics)* 21.3 (2002), 557–563 3.

[LEc18] L'ECUYER, P. "Randomized quasi-Monte Carlo: an introduction for practitioners". *Monte Carlo and Quasi-Monte Carlo Methods*. Ed. by OWEN, A. and GLYNN, P. Springer, 2018, 29–52 2.

[LK11] LAINE, S. and KARRAS, T. "Stratified sampling for stochastic transparency". *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering)* 30.4 (2011) 2, 3, 5, 6.

[Mat98] MATOUŠEK, J. "On the L_2-discrepancy for anchored boxes". *Journal of Complexity* 14.4 (1998), 527–556 3, 9, 11.

[Mit91] MITCHELL, D. "Spectrally optimal sampling for distribution ray tracing". *Computer Graphics (Proc. SIGGRAPH)* 25.4 (1991), 157–164 2, 9.

[Mit96] MITCHELL, D. "Consequences of stratified sampling in graphics". *Computer Graphics (Proc. SIGGRAPH)* 30.4 (1996), 277–280 2, 5.

[Nie87] NIEDERREITER, H. "Point sets and sequences with small discrepancy". *Monatshefte für Mathematik* 104.4 (1987), 273–337 2, 4.

[Owe03] OWEN, A. "Variance and discrepancy with alternative scramblings". *ACM Transactions on Modeling and Computer Simulation* 13 (2003), 363–378 3.

[Owe13] OWEN, A. *Monte Carlo Theory, Methods and Examples*. 2013 2, 3, 5, 7.

[Owe95] OWEN, A. "Randomly permuted (t,m,s)-nets and (t,s)-sequences". *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*. Ed. by NIEDERREITER, H. and SHIUE, P. Springer, 1995, 299–317 2, 3.

[PCX*18] PERRIER, H., COEURJOLLY, D., XIE, F., et al. "Sequences with low-discrepancy blue-noise 2-D projections". *Computer Graphics Forum (Proc. Eurographics)* 37.2 (2018), 339–353 3.

[Pha18] PHARR, M., ed. *ACM Transactions on Graphics (Special Issue on Production Rendering)*. Vol. 37. 3. 2018 1, 2.

[Pha19] PHARR, M. "Efficient generation of points that satisfy two-dimensional elementary intervals". *Journal of Computer Graphics Techniques* 8.1 (2019), 56–68 2, 6.

[PJH17] PHARR, M., JACOB, W., and HUMPHREYS, G. *Physically Based Rendering: From Theory To Implementation*. 3rd. Morgan Kaufmann, 2017 1, 2, 5, 8, 9.

[SÖA*19] SINGH, G., ÖZTIRELI, C., AHMED, A., et al. "Analysis of sample correlations for Monte Carlo rendering". *Computer Graphics Forum (Proceedings of Eurographics – State of the Art Reports)* 38.2 (2019), 473–491 2.

[Sob67] SOBOL', I. "On the distribution of points in a cube and the approximate evaluation of integrals". *USSR Computational Mathematics and Mathematical Physics* 7.4 (1967), 86–112 2, 5.

[VG95] VEACH, E. and GUIBAS, L. "Optimally combining sampling techniques for Monte Carlo rendering". *Computer Graphics (Proc. SIGGRAPH)* (1995), 419–428 9.

[YLS20] YANG, L., LIU, S., and SALVI, M. "A survey of temporal antialiasing techniques". *Computer Graphics Forum* 39.2 (2020), 607–621 7.