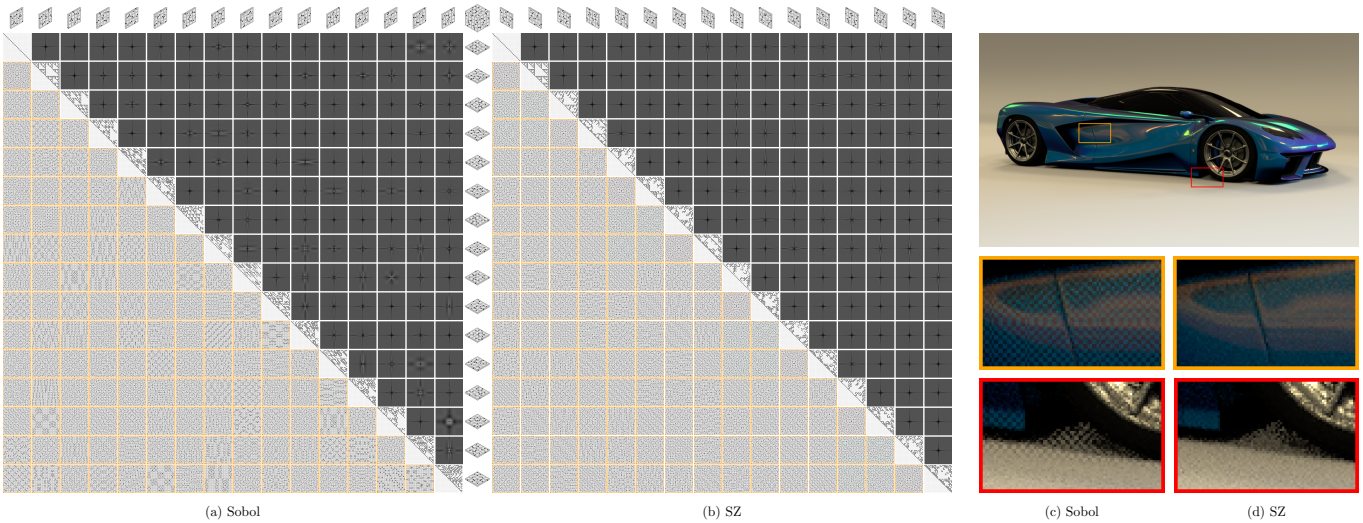
SZ Sequences: Binary-Constructed $(0, 2^q)$-Sequences

ABDALLA G. M. AHMED, CSSE, Shenzhen University, China
MATT PHARR, NVIDIA, USA
VICTOR OSTROMOUKHOV, Univ Lyon 1, CNRS, INSA Lyon, France
HUI HUANG*, CSSE, Shenzhen University, China



(a) Sobol (b) SZ (c) Sobol (d) SZ

Fig. 1. (a) Visualization of all pairwise projections of the first 16 dimensions of the Sobol sequence. The binary Sobol generator matrices appear along the diagonal. Each square below shows the first 256 points of the 2D projection using the corresponding pair of matrices in its row and column. Above the matrices are the power spectra of these projections, in diagonally mirrored positions, averaged over one million Owen-scrambled realizations. (b) The corresponding visualization for our SZ sequence. SZ points are well distributed across all 2D projections; here, all sets of 256 points satisfy 16×16 stratification, unlike Sobol points, and their power spectra are more regular and smoother at higher frequencies. Furthermore, unlike Sobol, where only the first two dimensions form a $(0, 2)$-sequence (red background), each successive pair of SZ matrices yields a $(0, 2)$-sequence. Moreover, each consecutive set of four SZ matrices generates 4D points that are $(0, 4)$-sequences (cyan background), which means they also exhibit all combinations of 4×64 stratifications in 2D, all combinations of 4×4×16 stratifications for all four 3D triplets, as illustrated in the exploded cube for the first three dimensions, and then there is a full 4×4×4×4 stratification in 4D. Finally, all 16 SZ matrices together form a $(0, 16)$-sequence in base 16. (c, d) These properties yield superior results in common rendering applications. The insets on the left, rendered using Sobol points, show Sobol's characteristic unconverged checkerboard pattern, while the insets on the right, rendered using SZ points, avoid this artifact and exhibit lower numerical error, achieving a 1.93× lower mean relative squared error for this scene at 64 samples per pixel.

Low-discrepancy sequences have seen widespread adoption in computer graphics thanks to the superior rates of convergence that they provide. Because rendering integrals often are comprised of products of lower-dimensional integrals, recent work has focused on developing sequences that are also well-distributed in lower-dimensional projections. To this end, we introduce a novel construction of binary-based $(0, 4)$-sequences; that is, progressive fully multi-stratified sequences of 4D points, and extend the idea to higher power-of-two dimensions. We further show that not only it is possible to nest lower-dimensional sequences in higher-dimensional ones—for example, embedding a $(0, 2)$-sequence within our $(0, 4)$-sequence—but that we can ensemble two $(0, 2)$-sequences into a $(0, 4)$-sequence, four $(0, 4)$-sequences into a $(0, 16)$-sequence, and so on. Such sequences can provide excellent rates of convergence when integrals include lower-dimensional integration problems in 2, 4, 16,... dimensions. Our construction is based on using 2×2 block matrices as symbols to construct larger matrices that potentially generate a sequence with the target $(0, s)$-sequence in base s property. We describe how to search for suitable alphabets and identify two distinct, cross-related alphabets of block symbols, which we call s and z, hence SZ for the resulting family of sequences. Given the alphabets, we construct candidate generator matrices and search for valid sets of matrices. We then infer a simple recurrence formula to construct full-resolution (64-bit) matrices. Because our generator matrices are binary, they allow highly-efficient implementation using bitwise operations and can be used as a drop-in replacement for Sobol matrices in existing applications. We compare SZ sequences to state-of-the-art low discrepancy sequences, and demonstrate mean relative squared error improvements up to 1.93× in common rendering applications.

*Corresponding author: Hui Huang

Authors' addresses: Abdalla G. M. Ahmed, CSSE, Shenzhen University, China, abdalla_gafar@hotmail.com; Matt Pharr, NVIDIA, USA, matt@pharr.org; Victor Ostromoukhov, Univ Lyon 1, CNRS, INSA Lyon, France, victor.ostromoukhov@liris.cnrs.fr; Hui Huang, CSSE, Shenzhen University, China, hhzhiyan@gmail.com.

0730-0301/2025/12-ART206 $15.00
https://doi.org/10.1145/3763272

CCS Concepts: • **Mathematics of computing → Quadrature**; • **Computing methodologies → Rendering**.

Additional Key Words and Phrases: Quasi-Monte Carlo Integration, Low Discrepancy Sequences, Sobol Sequences, Rendering

ACM Reference Format:
Abdalla G. M. Ahmed, Matt Pharr, Victor Ostromoukhov, and Hui Huang. 2025. SZ Sequences: Binary-Constructed $(0, 2^q)$-Sequences. *ACM Trans. Graph.* 44, 6, Article 206 (December 2025), 14 pages. https://doi.org/10.1145/3763272

1 INTRODUCTION

Sampling is a fundamental process in computer graphics (CG), underlying many areas such as halftoning and stippling, geometry processing, machine learning, and—most notably—rendering, where pixels are computed through Monte Carlo integration of complex light-transport paths. Although these paths are inherently high-dimensional, they are mostly composed of 2D surface interactions, along with some 1D constituents, which led to the idea of constructing high-dimensional samples analogously by pairing 2D and 1D samples over these constituents [Cook et al. 1984; Glassner 1994]. Research therefore focused on optimizing 2D distributions, and it was long believed that optimizing over the high-dimensional space was "at best a secondary concern" [Pharr and Humphreys 2004]. Even when the inherently high-dimensional low-discrepancy (LD) sequences were introduced to CG, typically only the first two dimensions were used and similarly combined, "padded", to build high-dimensional samples [Kollig and Keller 2002].

While LD sequences could be used to sample the image plane directly [Mitchell 1992], doing so with parallel rendering based on screen-space decomposition was challenging since consecutive samples from the sequence end up at far-away pixels. A milestone came with Grünschloß et. al. [2012] showing how to invert LD sequences, making it possible to determine which sample in the sequence corresponds to a given sample index in a pixel. This inversion greatly facilitated *global* LD sampling, where a single high-dimensional LD sequence is used to generate all samples for all pixels in all dimensions. Despite noticeable artifacts before convergence, the undeniably superior performance of this sampling strategy clearly indicated that high-dimensional uniformity does indeed matter. It is worth noting that the employed Sobol sequences had already been optimized for improved 2D projections by Joe and Kuo [2008]; thus, this success actually reflects a combination of two- and high-dimensional uniformity. Soon after, *pbrt* and Mitsuba—two widely used research rendering engines—both adopted global Sobol sampling as their default, and subsequent sampling research in CG largely focused on LD constructions, with the primary goal of exerting greater control over these distributions, especially their lower-dimensional projections.

In this paper, we make a major advancement in this research direction of controlling LD distributions. Rather than adapting one of the few known sequences, we present a novel construction of a complete family of LD sequences, built entirely from first principles. We use binary matrices to construct $(0, 4)$-sequences in base 4, as defined in Section 2. We then show how to extend this sequence to additional dimensions, achieving $(0, 2)$-sequences in each pair of dimensions, $(0, 4)$-sequences in each quartet, $(0, 16)$-sequences in each set of 16 dimensions, and so forth. Our sequences thus deliver superior integration performance in many cases where lower-dimensional projections are important. Because our generator matrices are binary, they can serve as a drop-in replacement for systems currently using Sobol samples: no algorithmic changes are required, and computational performance remains unaffected.

2 TECHNICAL BACKGROUND AND RELATED WORK

Our work belongs to the research area of uniform point distribution [Kuipers and Niederreiter 1974], which aims to produce point distributions that *uniformly* cover a domain better than *random* (white noise) distributions. The s-dimensional sampled domain is usually scaled to the half-open unit hypercube $(0, 1]^s$. In this section, we furnish a basic technical background needed to understand our proposed method, and briefly review the most closely related work.

2.1 Discrepancy

The term "discrepancy" refers to the error incurred when using a point set to estimate the volume of a region by counting the proportion of points falling inside. Star discrepancy, commonly denoted as D^*, is the maximum discrepancy over all axis-aligned hyperrectangles in the domain and provides a reliable measure of a point set's performance in numerical integration. A point set is considered a low-discrepancy (LD) set if it attains $O\left(\log^{s-1}(N)/N\right)$ discrepancy for N points in s-dimensions. A low-discrepancy sequence is an infinite sequence of points that yields an LD set for specific contiguous blocks of points of certain sizes, and attains $O\left(\log^{s}(N)/N\right)$ discrepancy for all contiguous blocks.

2.2 Radix-Based Constructions

A significant one-dimensional LD sequence construction was introduced by van der Corput [1935]. If

$$\ldots v_3 v_2 v_1 = \sum_{i=1} v_i 2^{i-1} \tag{1}$$

denotes the binary encoding of the sample index v, then the vth sample is computed by mirroring the bits of v around the fractional point:

$$x_v = \sum_{i=1} v_i 2^{-i}, \tag{2}$$

This approach can be generalized to higher dimensions by generating the kth component (dimension) of the sample through a distinct linear mapping:

$$X_v^{(k)} = C^{(k)} V, \tag{3}$$

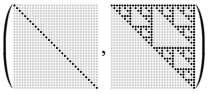
where V is a vector representing the vth digit-reversed van-der-Corput-like sample in some base b, $X_v^{(k)}$ is a vector representing the same-base fractional digits of the final sample location along the kth axis, and $C^{(k)}$ is a matrix in Galois Field (GF) b, known as a *generator matrix*, that linearly maps the reversed digits of the sample index to the digits of the computed sample—hence the term "digital" to collectively describe these methods. We next briefly outline the three most established digital construction approaches to LD sequences.

The first known higher-dimensional extension to van der Corput sequences is due to Halton [1960], who simply used a different prime

base for encoding the index v along each axis. Identity matrices may be used for $C^{(k)}$ in Eq. (3), but more complex *scrambling* matrices have been proposed to improve pairwise 2D projections.

A very different approach was taken by Sobol [1967], who retained a binary-base encoding and constructed a distinct generator matrix for each dimension, using a recurrence operation to compute each column of the matrix from the preceding columns, with coefficients derived from a primitive polynomial. These polynomials, in a sense, replace the role of different bases in Halton's construction. We skip the details and only highlight the most relevant one here that the first two primitive polynomials lead to the pair

$$(I, P) = \left(\quad , \quad \right) \tag{4}$$



that constitutes the first two, invariant, and most widely known dimensions of Sobol's construction, where

$$P = \left(p_{i,j} : \binom{j}{i} \bmod 2 \right) \tag{5}$$

happens to be the Pascal matrix in GF(2), with i and j denoting row and column indices, respectively, indexed from 0. For better visibility, we hereinafter adopt the convention of showing binary matrices as arrays of (0) light and (1) dark dots, and also occasionally use '.' instead of '0' for the same purpose.

These first two dimensions of Sobol inspired a third construction by Faure [1982], which uses Pascal matrices

$$C^{(k)} = \left(c_{i,j}^{(k)} : \binom{j}{i} k^{j-i} \bmod b \right), \tag{6}$$

in GF(b) with a prime base b, generating a b-dimensional LD sequence that is analogous to the 2D Sobol sequence in a sense explained in the following subsection. Niederreiter [1987] extended Faure construction to power-of-prime bases b^q using symbolic matrices and arithmetic over GF(b^q).

Thus, we identify three well-established construction approaches for high-dimensional LD sequences:

(1) Halton, using different prime bases.
(2) Sobol and variants, using binary-polynomial-based matrices.
(3) Faure and Niederreiter's extension, using Pascal matrices over fields.

Among the three, two factors have made Sobol the most widely adopted in computer graphics: (a) its binary base that leads to extremely fast computation, and (b) its superior distribution quality in lower dimensions, as explained in the following subsection. Halton follows, sharing the second but not the first of Sobol's properties. Because the prime bases are known at compile-time, the cost of integer divisions and modulus operations can be reduced [Warren 2012], bringing the digit reversal of Halton within tolerable limits, which led to Halton sequences being adopted in common rendering platforms such as *pbrt* [Pharr et al. 2023].

In contrast, we are unaware of any mention of Faure sequences and their derivatives in CG, which we attribute primarily to their large bases across all dimensions, requiring a considerably greater number of points to achieve the advertised uniformity. The lookup-based arithmetic of Niederreiter's extension [Bratley et al. 1992] exacerbates this limitation by adding significant computational overhead.

While our work is developed from first principles, the resulting construction may be considered an extension of Faure's—specifically its Niederreiter extension—that addresses both limitations: we ensemble higher-dimensional sequences from lower-dimensional ones and use GF(2) matrices like Sobol's. Thus, our construction effectively introduces the third category of LD sequences to CG.

2.3 (t, m, s)-Nets and (t, s)-Sequences

Niederreiter [1987] also established a complete theoretical framework for developing and studying a large class of LD constructions, including the radix-based methods listed above. At the heart of this theory is so-called (t, m, s)-Nets in base b, which describe a *multi-stratified* distribution of b^m points in an s-dimensional hypercube such that, for all possible stratifications (slicings) of the domain into b^{m-t} similar rectangular *strata* (cells), exactly b^t points appear in each stratum. For example, Fig. 1(b) illustrates the essence of a $(0, 4, 4)$-net in base 4. Then come (t, s)-sequences in base b, which are infinite sequences of s-dimensional points such that, for all integer m, the first and all subsequent blocks of b^m points form (t, m, s)-nets in base b. While these definitions are independent of the construction method, the model possibly existed thanks to the existence of algebraic recipes for construction. Specifically, the definition of (t, m, s)-nets translates directly into a simple condition on the digital framework: if we build a hybrid $m \times m$ matrix

$$H_{m_1,m_2,\ldots,m_s} = \left(\begin{array}{ccc} c_{0,0}^{(1)} & \cdots & c_{0,m-1}^{(1)} \\ \vdots & \vdots & \vdots \\ c_{m_1-1,0}^{(1)} & \cdots & c_{m_1-1,m-1}^{(1)} \\ \hline \vdots & \vdots & \vdots \\ \hline c_{0,0}^{(s)} & \cdots & c_{0,m-1}^{(s)} \\ \vdots & \vdots & \vdots \\ c_{m_s-1,0}^{(s)} & \cdots & c_{m_s-1,m-1}^{(s)} \end{array} \right), \tag{7}$$

by combining the first m_k rows of the kth generator matrix in Eq. (3), with $\sum m_k = m$, then each such a hybrid matrix, multiplied by V, computes the allocation of the points to strata in a specific stratification. The $(0, m, s)$-net condition, for example, then corresponds to requiring that each hybrid matrix is invertible, thereby ensuring a one-to-one correspondence of points to strata. A $(0, s)$-sequence requires a set of generator matrices that progressively maintains this hybrid-matrix condition over their top-left corners.

These definitions suggest that it is desirable to keep both the base b and the parameter t small, although different applications may require different configurations. For example, the van der Corput construction is a $(0, 1)$-sequence in base 2. Faure sequences are $(0, b)$-sequences in their respective prime bases, while Niederreiter's extension to Faure creates $(0, b^q)$-sequences in power-of-prime bases b^q.The first two dimensions of the Sobol sequence comprise a $(0, 2)$-sequence in base 2, but the t value increases as more dimensions are taken, which gives higher quality to lower dimensions, as noted in Sec. 2.2. Halton sequences, in contrast, do not readily fit within the (t, s)-sequences model unless it is adapted to accept

different bases. However, they still exhibit a multistratified structure. Thus, the common ground among all three families of constructions is that they attain optimal quality at powers of the base, or the product of bases in the case of Halton. This explains the superiority of Sobol and Halton over Faure for graphics applications, where lower dimensions are usually more important.

In this paper, we develop a novel construction of $(0, 2^q)$-sequences in the corresponding 2^q bases, using blocks of binary matrices. That is, instead of using base-4 digits directly, for example, we use 2×2 binary matrices as building blocks to emulate base-4 matrices. Please note that $b = s$ is implied in the following if no base is mentioned.

2.4 Scrambling and Shuffling of LD Constructions

Beyond the algebraic recipes for constructing LD sets and sequences, the design space is significantly enriched by *scrambling* techniques that can derive new valid nets or sequences from existing ones. Tezuka [1994] showed that multiplying arbitrary lower-triangular matrices on the left of the generator matrices produces valid generators, since doing so preserves the ranks of all hybrid matrices. Owen [1995], in contrast, introduced a post-processing technique, "Owen scrambling", that preserves the net/sequence structure. In contrast to scrambling that applies to sample locations, the term *shuffling* refers to reordering (permuting) the sample indices [Faure and Tezuka 2002]. As $(0, s)$-sequences, all these are applicable to our sequences, and we will employ them in our development.

2.5 LD Sampling in Computer Graphics

Interest in LD constructions has grown in the graphics community over the past decade, largely focused on finding ways to impose control over these distributions, as commonly required in graphics applications. hmed et al. [2016] presented an original 2D low-discrepancy construction that enabled imposing a blue-noise spectrum, and Perrier et al. [2018] subsequently imposed a blue-noise spectrum onto selected pairs of dimensions of a Sobol sequence. While both methods compromised the discrepancy of the underlying distribution, they provided a proof of concept that encouraged further research.

n a different line of work, Christensen et al. [2018] searched for dyadic (i.e., base-2) $(0, 2)$-sequences with good spatial and spectral properties, and Pharr [2019] presented a more efficient search. Ahmed and Wonka subsequently presented a theorem [2021, Theorem 4.2] implying that the search space is actually confined to Owen-scrambled 2D Sobol. Soon after, the first group of authors, led by Helmer [2021], developed an extremely fast implementation of Owen scrambling. We regard this as an excellent example of the community's fast-paced learning cycles in analyzing and manipulating the inherently difficult-to-understand LD constructions. In this paper, we present an almost complete cycle from experimentation to the development of theoretical elements.

n more recent years, research on LD in CG has enriched the quasi-Monte Carlo (QMC) literature with significant findings and novel constructions, including an efficient algorithm that spans the entire space of dyadic $(0, m, 2)$-nets [Ahmed and Wonka 2021], a cascade of dyadic $(0, m, 2)$-nets over successive pairs of dimensions [Paulin et al. 2021], a family of self-similar dyadic $(0, 2)$-sequences [Ahmed et al. 2023], a gradient-descent optimization of discrete Owen scrambling [Doignies et al. 2024], and a practical study of a base-3 analogue of Sobol sequences [Ostromoukhov et al. 2024]. Our current work is partially inspired by this last work of Ostromoukhov et al., where we thought of using base 4 instead to gain the advantage of binary computation.

3 EXPLORATION

In this section, we report our pilot search for binary matrices that generate $(0, 2^q)$-sequences, beginning with experimental exploration and gradually developing an analytical framework distilled from our empirical findings. Starting with four dimensions, our initial goal is to construct a $(0, 4)$-sequence by searching for a set of generator matrices that linearly map a reverse-ordered vector of sample-index digits into vectors of digits representing coordinates along the four axes, as in Eq. (3). It is already established and well known that plain matrices and straightforward arithmetic in a composite base like 4 would not serve the purpose. An intuitive way to understand this deficiency is to note that a factor of the base, e.g., 2 in base 4, does not preserve all the ranks of a multiplied sequence number digit modulo the base, which drastically limits the chance of finding a viable set of matrices. Rather than using GF(4) symbolic arithmetic like Niederreiter [1987], our idea is to use plain GF(2) matrices and vectors, interpreting 2×2 blocks of the generator matrices and pairs of bits of the multiplied V vector as base-4 digits. Our first key insight is that the set of 2×2 invertible binary matrix blocks can resolve all six permutations of the non-zero numbers in a base-4 digit. While this does not prove that building matrices from such blocks would yield a $(0, 4)$-sequence, it is at least encouraging to explore. Please note that all arithmetic hereafter is in GF(2), which corresponds to using bitwise AND/XOR for multiplication/addition, respectively.

We begin with an exhaustive search over a small range to validate the concept. Without loss of generality, we assume the identity matrix I for the first dimension, since its matrix can be factored out on the right of the four matrices to permute the sequence number. Our plan is to progressively search for three binary square matrices that expand by two rows and columns at each step and satisfy the hybrid-matrix condition, Eq. (7). That is, at each step and for each dimension, we expand each matrix,

$$A_{m+2} \leftarrow \left(\begin{array}{c|c} A_m & C \\ \hline R & X \end{array}\right), \tag{8}$$

by adding an m×2 column C, a 2×m row R, and a 2×2 corner block X. We will drop the suffix hereinafter where it is unambiguous. We can always assume $R = 0$ by factoring out a lower-triangular Tezuka-scrambling matrix:

$$\left(\begin{array}{c|c} A & C \\ \hline R & X \end{array}\right) = \left(\begin{array}{c|c} I & 0 \\ \hline RA^{-1} & I \end{array}\right) \left(\begin{array}{c|c} A & C \\ \hline 0 & RA^{-1}C + X \end{array}\right), \tag{9}$$

where I is an appropriately sized identity matrix. Then we use the factoring

$$\left(\begin{array}{c|c} A & C \\ \hline 0 & X \end{array}\right) = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & X \end{array}\right) \left(\begin{array}{c|c} A & C \\ \hline 0 & I \end{array}\right) \tag{10}$$

to reveal that X must be invertible. This reduces the set of possible corner-extension matrices from 16 to the following six:

$$\{\text{⠑}, \text{⠝}, \text{⠕}, \text{⠎}, \text{⠮}, \text{⠵}\} \ . \tag{11}$$

We can further exclude the third and fifth by factoring out a lower-triangular Tezuka-scrambling matrix, reducing the set to only four:

$$\{\text{⠑}, \text{⠝}, \text{⠎}, \text{⠵}\} \ , \tag{12}$$

which is more favorable as a power of 2.

We built an efficient exhaustive search algorithm by packing entire matrices into 64-bit words, enabling fast bit manipulation and the construction of lookup tables for invertible matrices. This approach enabled searching up to 8×8 matrices, equivalent to four extension steps. The search confirmed the existence of multiple working sets of matrices, which further motivated us to conduct two additional key experiments. We first considered *nesting* a $(0, 2)$-sequence, namely 2D Sobol, within our $(0, 4)$ set. To this end, we enforced the extension of the second-dimension matrix to reproduce the Pascal matrix and searched for a complementary pair of matrices. This approach worked successfully all the way to 8×8 matrices, with multiple choices at each extension step. Next, we considered *ensembling* two $(0, 2)$-sequences into a $(0, 4)$-sequence. To achieve this, we used the formula recently introduced by Ahmed et al. [2023], which states that for any pair U_x, U_y of upper-triangular generator matrices to form a $(0, 2)$-sequence, they must satisfy

$$U_y U_x^{-1} = U_x U_y^{-1} = P \, . \tag{13}$$

Thus, we conducted the search as with nesting and, at each extension, filtered the results that satisfied this condition. Once again, the experiment worked successfully all the way to the 8×8 matrices.

3.1 Initial Findings

For all combinations of valid initial 2×2 blocks of the three matrices, we consistently obtained 768 distinct valid sets for the first extension to 4×4, and 384 distinct valid extensions on each branch for subsequent extensions to 6×6 and 8×8, suggesting an analytical construction model. The fixed number of alternatives in each extension step indicates that the degrees of freedom are likely limited to the top and bottom blocks of the extension columns. Following this hint, we note that the corner extension block X in Eq. (8) is not involved in any of the hybrid matrices and therefore, for each of the three matrices, represents a free choice among the four options in Eq. (12). This accounts for a factor of $4^3 = 64$, and we verified that fixing the extension corner blocks X to identity reduces the list to 12 sets in the first and 6 in subsequent extension steps. This leaves us with the degrees of freedom in the top row, which we analyze next.

Let

$$\begin{pmatrix} A_0 \, A_1 \, \ldots \, A_{j-1} \, A_j \end{pmatrix} \tag{14}$$

denote the first block row in one matrix A of the three at the jth extension, where each A_i is a 2×2 block. By constructing a hybrid matrix comprising j rows of the first identity matrix together with this row of A, we find that

$$\left| \begin{array}{ccc|c} \multicolumn{3}{c|}{I} & \\ \hline A_0 & \ldots & A_{j-1} & A_j \end{array} \right| = 1 \implies \left| A_j \right| = 1 \, , \tag{15}$$

leading to the conclusion that, for each of the three matrices, all 2×2 blocks of the first block row must be invertible.

Now consider taking a hybrid of the first $j-1$ rows of I with the first row of each of two other matrices A and B, which yields

$$\left| \begin{array}{ccc|cc} \multicolumn{3}{c|}{I} & & \\ \hline A_0 & \ldots & A_{j-2} & A_{j-1} & A_j \\ B_0 & \ldots & B_{j-2} & A_{j-1} & A_j \end{array} \right| = 1 \implies \begin{vmatrix} A_{j-1} & A_j \\ B_{j-1} & B_j \end{vmatrix} = 1 \, . \tag{16}$$

Since all four elements are invertible by virtue of Eq. (15), we can use our earlier "$RA^{-1}C + X$" reduction of Eq. (9) to obtain

$$\left| B_{j-1} A_{j-1}^{-1} A_j + B_j \right| = 1 \, , \tag{17}$$

or equivalently

$$\left| A_{j-1}^{-1} A_j + B_{j-1}^{-1} B_j \right| = 1 \, . \tag{18}$$

Let us define a symbol

$$a_j = A_{j-1}^{-1} A_j \, , \tag{19}$$

to designate the product of the inverse of the $j-1$st by the jth entry in the first block-row of matrix A. Then Eq (18), rewritten as

$$\left| a_j + b_j \right| = 1 \, , \tag{20}$$

states that the sum of corresponding symbols in the first row of each pair A, B of generator matrices must be invertible. This also implies that the symbols at corresponding slots must be distinct for the three matrices. Thus, for each slot in the first block-row, the inserted block-matrix symbols must be

(1) invertible,
(2) distinct for each matrix, and
(3) have invertible sums.

The population of six invertible 2×2 matrices comprises two disjoint sets,

$$(S, Z) = \left(\left\{ \underbrace{\text{⠑}}_{s_1}, \underbrace{\text{⠝}}_{s_2}, \underbrace{\text{⠕}}_{s_3} \right\}, \left\{ \underbrace{\text{⠊}}_{z_1}, \underbrace{\text{⠎}}_{z_2}, \underbrace{\text{⠮}}_{z_3} \right\} \right) , \tag{21}$$

that satisfy these requirements. Each set comprises two diagonally-mirrored triangular elements, and a counter-diagonal element that switches between them via addition, hence we call them S (think ⠎) and Z (think ⠵) to make it easier to recall which is which; hence the name SZ for these sequences. We will also hereinafter use this "S/Z" convention to distinguish left/right-hand elements and operations, respectively.

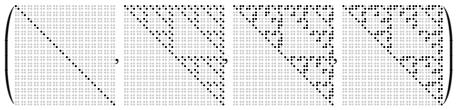
We conclude this subsection by noting that these three simple rules cover all the requirements needed to satisfy the hybrid matrix condition for the first 2×2 block matrices. Furthermore, they extend analogously to other powers of 2. Thus, constructing binary matrices for generating a "$(0, 2, 2^q)$-net in base 2^q" reduces to finding an alphabet of $q \times q$ block matrices that satisfy these rules. This name in Niederreiter's notation just means a set of Latinized 2^{2q} points in 2^q dimensions that is stratified across all $2^q(2^q - 1)/2$ pairs of 2D projections; cf. Jarosz et. al. [2019]. Next, we show how to extend the matrices to arbitrary sizes for $(0, 2^q)$-sequences in base 2^q.

3.2 Refinement

Instead of defining the symbols in Eq. (19) in terms of adjacent entries, it is more constructive to populate the matrices directly with symbols. Recognizing that all first-row blocks must be invertible enables us to achieve this through the following subtle factoring

$$\left(\begin{pmatrix} I & & \\ & \ddots & \\ & & I \end{pmatrix}, \begin{pmatrix} I & \cdots & A_j \\ & \ddots & \vdots \\ & & I \end{pmatrix}, \begin{pmatrix} I & \cdots & B_j \\ & \ddots & \vdots \\ & & I \end{pmatrix}, \begin{pmatrix} I & \cdots & C_j \\ & \ddots & \vdots \\ & & I \end{pmatrix}\right) = \begin{pmatrix} I & & \\ & \ddots & \\ & & A_j^{-1} \end{pmatrix}\left(\begin{pmatrix} I & & \\ & \ddots & \\ & & I \end{pmatrix}, \begin{pmatrix} I & \cdots & I \\ & \ddots & \vdots \\ & & I \end{pmatrix}, \begin{pmatrix} I & \cdots & B_jA_j^{-1} \\ & \ddots & \vdots \\ & & I \end{pmatrix}, \begin{pmatrix} I & \cdots & C_jA_j^{-1} \\ & \ddots & \vdots \\ & & I \end{pmatrix}\right)\begin{pmatrix} I & & \\ & \ddots & \\ & & A_j \end{pmatrix}. \tag{22}$$

The common S (left-hand-side) factor represents a Tezuka scrambling and can therefore be dropped. Thus, without loss of generality, we can populate all blocks in the first row of the second-dimension matrix (the first of the three) with identity matrix blocks ⁘. Once this final factoring is implemented, the exhaustive search reduces to this single quadruple of generator matrices:



(23)

here shown to 32-bit depth. Based on our exhaustive search, this quadruple serves as the template for all binary-matrix-constructed $(0, 4)$-sequences, and all other possible variants can be derived by reintroducing the degrees of freedom factored out earlier, as we will discuss later.

Careful inspection of these matrices reveals that they are built exclusively from the S block symbols in Eq. (21), plus empty (0) blocks. Further inspection, guided by the preliminary assumption that these sequences are related to Faure's, reveals that they are in fact Pascal matrices

$$\mathcal{P}(a) = \left(p_{i,j} : \binom{j}{i} a^{j-i}\right), \tag{24}$$

constructed from these four block symbols. For example,

$$\mathcal{P}(⁘) = \left(p_{\text{row,col}} : \underbrace{\binom{\text{col}}{\text{row}}}_{\text{mod } 2} (⁘)^{\text{col}-\text{row}}\right) = \begin{matrix} ⁘^0 & ⁘^1 & ⁘^2 & ⁘^3 & ⁘^4 & ⁘^5 & ⁘^6 & ⁘^7 & \cdots \\ ⁘^0 & & ⁘^2 & & ⁘^4 & & ⁘^6 & & \cdots \\ & ⁘^0 & ⁘^1 & & & ⁘^4 & ⁘^5 & & \cdots \\ & & ⁘^0 & & & & ⁘^4 & & \cdots \\ & & & ⁘^0 & ⁘^1 & ⁘^2 & ⁘^3 & & \cdots \\ & & & & ⁘^0 & & ⁘^2 & & \cdots \\ & & & & & ⁘^0 & ⁘^1 & & \cdots \\ & & & & & & ⁘^0 & & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{matrix}. \tag{25}$$

4 SZ SEQUENCES

Based on our exhaustive search for binary $(0, 4)$-sequence generators and similar partial searches for $(0, 8)$ and $(0, 16)$ sequences, we find that what characterizes the S set of 2×2 block symbols, compared to the complementing Z set in Eq. (21), is that it is not only closed under addition, but also multiplication, and hence constitutes a *finite field*, as dictated by Wedderburn's little theorem, stating that "every finite division ring is a field". In a nutshell, finite fields are sets of elements with operations analogous to addition and multiplication, closed under both operations, and equipped with neutral elements analogous to 0 and 1, as well as additive and multiplicative inverses. The only missing element in our S alphabet to make a field is zero, but in fact this element has existed and been used throughout our development: it is the symbol used to build the identity matrix in our set. This can be slightly confusing, since we have a "grand" identity matrix for the final set of generator matrices, built from the 0 symbol, and a "Pascal-ed" identity symbol used to build the second-dimension generator matrix.

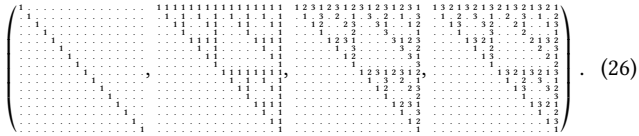
Algorithm 1: Constructing an SZ $(0, 2^q)$-sequence.

Input : (1) index q of required power-of-2 dimension $s = 2^q$.
Output : A set of s binary generator matrices to produce a $(0, s)$-sequence.

1 Search for a set (alphabet) $\{x_i\}_{i=1}^{s-1}$ of $s - 1$ invertible $q \times q$ matrices that is closed under addition, multiplication, and inversion;
2 Build a Pascal matrix $\mathcal{P}(x)$ of the desired numerical resolution for each symbol in the alphabet;
3 Along with an equally sized identity matrix, these constitute the desired set of generator matrices.

Once we arrive at this conclusion for 4D, we can readily envision analogous constructions for higher powers of 2 dimensions, as summarized in Algorithm 1. Using this algorithm, we constructed SZ sequences for different values of q, and empirically verified their $(0, s)$-sequence property by building and validating all the sets of hybrid matrices. It is worth noting that, as a special case of our model, the conventional Pascal matrix in 2D Sobol/Faure constructions is the Pascal matrix of the identity of a 2D alphabet comprising granular 0 and 1 as its block matrices. Thus, we hereinafter treat an all-zero q×q block matrix as an intrinsic symbol in our alphabets.

While we did not proactively choose to use fields, they automatically emerged, bringing us closer to Faure's sequences and their Niederreiter's extension. Let us briefly abstract our empirical search results in Section 3 to understand what happened: We constructed the following four base-4 matrices:



(26)

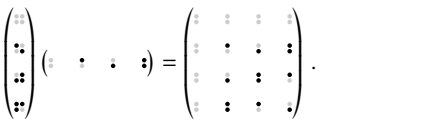
These are exactly the ones in Eq. (23), just encoding each distinct 2×2 block as a base-4 digit, namely indexed after 0 and the three S symbols in Eq. (21). When multiplied by a base-4 digit-reversed column vector representation of a sequence number, these matrices generate a $(0, 4)$-sequence in base 4, i.e., a sequence of 4D points fully multi-stratified for powers of 4 numbers of points. However, multiplication does not follow the standard modulo-4 arithmetic but instead uses a special table

S \ Z	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

(27)

that amends the degenerate modulo-4 multiplication table, preparing it for Faure construction. Here S refers to the left multiplicand digit

from the matrix, and Z to the right one from the vector. Finally, *this arithmetic table is neither built arbitrarily nor looked up; it is intrinsic to the matrix-block alphabet and is evaluated automatically via GF(2) matrix-vector multiplication:*



(28)

Please mind the bit-reversed ordering: the top row is least significant bit, so • is 1 and ⁚ is 2.

Although we have arrived at this construction experimentally and independently, as detailed in the preceding Section 3, once abstracted this way we observe a close similarity to Niederreiter's analytically developed construction [1987; 1992] that extends Faure's sequences to power-of-prime bases. Moreover, the uniqueness of finite fields strongly suggests that our construction coincides with Niederreiter's. This remains a hypothesis, however, awaiting analytical validation. The multiple degrees of freedom in both constructions can produce different realizations, making empirical verification difficult. In any case, both constructions span the same universe of $(0, 2^q)$-sequences in base 2^q. It is the emphasized text in the preceding description, however, that is the essential difference between our construction and Niederrieter's: Instead of using symbolic GF(2^q) arithmetic with lookup tables, as in standard implementations of Niederreiter sequences [Bratley et al. 1992], our field elements are embedded as block matrices in GF(2) generator matrices, which handle both the field arithmetic and the bijection operators in Niederreiter's construction. This makes our binary-based construction far more efficient and scalable. Another important advantage is that our experimental approach led us to discover nesting and ensembling, which make a significant difference in graphics applications, as we will see later in Section 5.2.

4.1 Alphabets

Identifying alphabets as finite fields brings many insights and implications. The most relevant property for our construction is that in fields like our alphabets there exists a symbol (actually multiple) that multiplicatively cycles through all other non-zero symbols; that is, it can generate all the non-zero elements as its powers. Such a symbol is called "primitive" and is commonly denoted by α. The alphabet can then be written as

$$\Sigma = \{0, I, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}\,. \tag{29}$$

Aside from analytical insights and proofs, alpha elements are also highly useful in practice. The first advantage they offer is a straightforward search plan for alphabets, as summarized in Algorithm 2, replacing the complex and time-consuming stack-based searches we used in our exploration phase. Secondly, noting that an alpha element belongs to a unique alphabet, the search algorithm can easily be modified to enumerate all alphabets by dividing the number of alpha elements in the universe of invertible matrices by the number of alpha elements in a single alphabet. We performed such a search over the feasible range and obtained

$$\left(\left|\{\Sigma(q)\}_{q=1}^{6}\right|\right) = (1, 1, 8, 336, 64512, 53329920)\,. \tag{30}$$

Algorithm 2: AlphaSearch: Searching for an alphabet.

Input : (1) A power of 2 size of elements $s = 2^q$,
(2) A timeout number of attempts T.
Output : An alphabet of s block matrices.

```
for i ← 0 to T − 1 do
    Construct a random invertible q × q matrix block x;
    Repeatedly multiply x by itself until you get I, and record
      the needed number of multiplications: its "root index" n;
    if n = s − 1 then
        return {0, {x^i}_{i=0}^{s−2}};
Return a timeout message.
```

A brief Internet search identified this sequence as OEISA258745 [2015], described as "Order of general affine group AGL(n,2) (=A028365(n)) divided by (n+1)". This suggests the possibility of constructing alphabets algorithmically, which we leave for future research.

4.2 Randomization

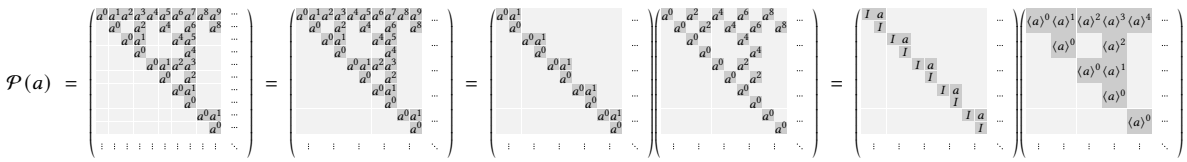
With the template set of generator matrices built exclusively from alphabet building blocks, we can now randomize the set by reintroducing the degrees of freedom we previously factored out. The extension rows removed in Eq. (9), along with the X diagonal blocks factored out later, correspond exactly to Tezuka's [1994] scrambling mentioned in Section 2.4. Beyond that, our model readily supports Owen's [1995] scrambling, Faure–Tezuka [2002] shuffling in both 2 and 2^q bases, as well as XOR-scrambling [Kollig and Keller 2002].

4.3 Nesting

The key idea of nesting, introduced in the exploration phase, is to find an alphabet over 2^{2q} whose symbols embed those of a given alphabet over 2^q, in such a way that the generators of the nested sequence can be reproduced by sequence-preserving manipulations of the corresponding generators of the nesting sequence. As illustrated in Fig. 2, it is always possible to refactor a generator Pascal matrix of a q×q block symbol a into a scrambled Pascal matrix of a corresponding $2q$×$2q$ nesting symbol

$$\langle a \rangle = \begin{pmatrix} a^2 & \\ & a^2 \end{pmatrix}. \tag{31}$$

Hence, the task of nesting reduces to finding a 2^{2q} alphabet that includes the set $\{\langle a_i \rangle\}_{i=1}^{2^q-1}$ of all nesting symbols. That is, finding a $2q$×$2q$ alpha matrix whose powers include this set. A brute-force approach to such a search is far more difficult than before, since we are seeking highly specific alphabets. We therefore generated and analyzed all nesting alphabets within a feasible range and, fortunately, found that all nesting alphabets are built exclusively from the nested alphabet symbols. This observation dramatically reduces the search space from 2^{4q^2} to 2^{4q}, making it feasible to reach 64K dimensions. This reduction not only enables us to find working alphabets but also to sample them uniformly.



(a) Granular (b) 2×2 Grouping (c) Factoring (d) Reinterpreting

Fig. 2. A step-by-step refactoring illustration of a Pascal matrix of a given $q\times q$-block-matrix symbol a into a scrambled Pascal matrix of a $2q\times 2q$ nesting symbol $\langle a\rangle = \begin{pmatrix} a^2 & \\ & a^2 \end{pmatrix}$. The process is generic, requiring neither symbol to belong to a valid alphabet.

Algorithm 3: NestSort: order alphabets to enable ensembling.

Input : An alphabet Σ of $s = 2^q$ block-matrix symbols.
Output: An ordered alphabet with a nested substructure.

Ensure the placement of the 0 and I symbols in the first and second slots, respectively;
$k \leftarrow 2$;
while $k < s$ **do**
  for $i \leftarrow 1$ **to** $k-1$ **do**
    $x \leftarrow \Sigma[i] + \Sigma[k]$;
    Locate the slot j containing x;
    Swap $\Sigma[i+k]$ with $\Sigma[j]$;
  $k \leftarrow 2\cdot k$;

Once the alphabet is found, the generator matrices are built according to Algorithm 1, and the generator matrices of each embedded symbol a are then multiplied by the factor

$$\mathcal{P}(a) = \begin{pmatrix} I & a \\ & I \end{pmatrix} \mathcal{P}(\langle a\rangle), \tag{32}$$

to restore the original matrices, as illustrated in Fig. 2. Note that for the nesting set, this multiplication corresponds to Tezuka scrambling and therefore preserves the structure of the sequence.

We conclude this section by noting that nesting, as described above, works only by doubling q, squaring the dimension at each upsampling step of the sequence space, rather than simply doubling the dimension as one might hope.

4.4 Ensembling

Ensembling is less obvious than nesting, since it depends on a non-obvious property. We start with a nested sequence, and the key idea is to try to make subsequent bands of dimensions of the nesting sequence, with respect to each other, look identical to their counterparts in the nested band. This process begins by arranging the symbols in a suitable order, as outlined in Algorithm 3, which arranges the symbols in a literally nested algebraic structure of bands

$$B_1 \leftarrow (0, I),\ B_2 \leftarrow (B_1, B_1 + x_1),\ B_3 \leftarrow (B_2, B_2 + x_2),\ \ldots \tag{33}$$

This idea rests on an interesting property

$$\mathcal{P}(a+x) = \mathcal{P}(a)\mathcal{P}(x), \tag{34}$$

of Pascal matrices that summing the symbols is equivalent to multiplying the corresponding Pascal matrices. A sketch of the proof begins by noting that each entry

$$\binom{j}{i}(a+x)^{j-i}, \tag{35}$$

of $\mathcal{P}(a+x)$ on the right-hand side embodies a binomial expansion with the same range of powers as the corresponding row–column product on the left-hand side; it then remains only to equate the binomial coefficients. Applying this property to the nested structure in Eq. (33), we observe that adding a symbol to each subset is equivalent to multiplying its generator matrix by those of the subset, with the net effect of reordering the points in the subset while preserving their geometry. In other words, the sub-sequence of each band is merely a reordering of the reference band's sequence and can therefore be transformed into a nested sequence using the same restoration in Eq. (32). In fact, it is the identical sequence, but in a different order. This can be observed in Fig. 1(b), where all the red $(0, 2)$ pairs and cyan $(0, 4)$ quadruples appear identical. In Appendix A, we present a step-by-step demonstration of nesting and ensembling used to generate the sequence shown in Fig. 1(b).

5 RESULTS AND APPLICATIONS

To evaluate the performance of SZ sequences, we tested them both on the numerical integration of synthetic functions and on rendering tasks. We also include empirical measurements of discrepancy. Our implementation is provided in the supplemental material.

5.1 Numerical Integration

To compare the performance of SZ points with other point sets for integration, we measured the mean relative squared error (MRSE) when integrating a variety of simple analytic functions. We follow the approach developed by Jarosz et al. [2019, Section 5.2], summarized below. Three 1D functions

$$g^0(r) = 1 - \text{binaryStep}(r, r_{\text{end}}), \tag{36}$$

$$g^1(r) = 1 - \text{linearStep}(r, r_{\text{start}}, r_{\text{end}}), \tag{37}$$

$$g^\infty(r) = \mathrm{e}^{-r^2/(2\sigma^2)}, \tag{38}$$

serve as building blocks, with $r_{\text{end}} = 3/\pi$, $r_{\text{start}} = r_{\text{end}} - 0.2$, and $\sigma = 1/3$. Projection of an n-dimensional point p to a set of dimensions d_i is denoted by

$$p_{d_1,\ldots,d_n} = (p_{d_1}, \ldots, p_{d_n}), \tag{39}$$

n-dimensional functions are then defined by taking the vector norm of projected points

$$f^D_{d_1,\ldots,d_n}(p) = g^D(\|p_{d_1}, \ldots, p_{d_n}\|). \tag{40}$$

These are plots of associated 2D functions:

$f^0_{1,2}$ $f^1_{1,2}$ $f^\infty_{1,2}$

We used four forms of functions for our tests:

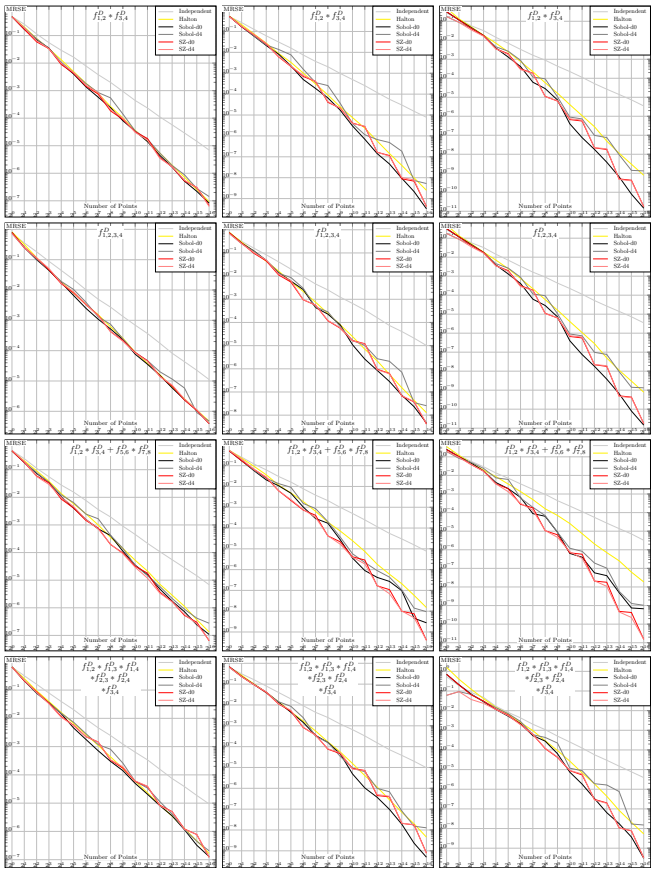
- $f^D_{1,2} * f^D_{3,4}$: the product of two 2D functions.
- $f^D_{1,2,3,4}$: a fully 4D function.
- $f^D_{1,2} * f^D_{3,4} + f^D_{5,6} * f^D_{7,8}$: the sum of the product of two 2D functions.
- $f^D_{1,2} * f^D_{1,3} * f^D_{1,4} * f^D_{2,3} * f^D_{2,4} * f^D_{3,4}$: the product of functions of all 2D projections of a 4D point.

For each of these and for each of the three g functions defined above, we computed a reference value and then ran 1,024 independent trials, each taking up to 2^{16} samples. Owen scrambling was used to randomize the low-discrepancy sequences. In addition to our SZ sequence and the Sobol sequence, we also measured results with independent uniform random numbers and with the Halton sequence. For SZ and Sobol, we also measured results using samples starting at the 4th dimension, to reflect the case in rendering where such integrands might be encountered after a few dimensions of the sequence had already been consumed. We will use "SZ-d0" and "SZ-d4" to distinguish these, and similarly for Sobol. We measured MRSE at all power-of-two number of samples

The resulting error plots are shown in Fig. 4. For all functions, we observe that the performance of all samplers other than the independent sampler is similar for g^0; therefore, in the following we focus on g^1 and g^∞. For both the first and second function forms, SZ-d0 matches Sobol-d0's performance at power-of-4 sample counts, although it exhibits higher error at intermediate powers of two However, SZ-d4 attains error levels comparable to SZ-d0, whereas Sobol-d4 is significantly worse. We would expect this trend to continue at higher dimensions, as SZ maintains its $(0, 2)$- and $(0, 4)$-sequences, while the quality of lower-dimensional projections of Sobol points worsens. For the third function form, the sum of the product of two 2D functions, both SZ-d0 and SZ-d4 significantly outperform the Sobol sampler. We believe that this is due both to it providing $(0, 2)$-sequences for each of the constituent 2D functions but also proving $(0, 4)$-sequences for each term. Finally, in the product of all of the 2D projections of a 4D point, we again see SZ-d0 and SZ-d4 matching Sobol at power-of-4 sample counts, and SZ-d4 having much lower error than Sobol-d4.

5.2 Rendering

To quickly assess the potential of SZ sequences in rendering, we implemented a special scene setup that uses only 2D constituents to build the light-transport path. Specifically, we modified *pbrt* so that the first two dimensions of the sequence are used to select a point inside each pixel area, the next two are used to select a point



(a) Binary Step (g^0) (b) Linear (g^1) (c) Gaussian (g^∞)

Fig. 4. Mean relative squared error when integrating of a variety of analytic 4- and 8-dimensional functions. In these plots, "Sobol-d0" and "Sobol-d4" denote Sobol points starting at dimension 0 and 4, respectively, and similarly for SZ. SZ-0 performs as well as Sobol at power-of-4 sample counts, though is sometimes worse at intermediate power-of-2 counts. In some cases (e.g., the third row of g^∞), it gives significantly lower error. SZ-4 generally performs as well as SZ-0, while Sobol-4 often has higher error than Sobol-0.

on the lens, the next two to importance-sample a direction from the environment map's distribution, and then two more are used to sample the BSDF. Note that this differs from *pbrt*'s default assignment of dimensions for integration which does not consider the possibility of $(0, 2^q)$-sequences and so consumes additional 1D samples such as for time and sampling which light source to sample from early dimensions of samplers. With this setup, we evaluated global sampling with Sobol and SZ points for rendering direct illumination from an environment map, using two error metrics: MRSE and the perceptually-based ꟻLIP metric [Andersson et al. 2020, 2021]. We chose the *Sportscar* scene for the evaluation since it includes a variety of complex measured BSDFs, ranging from nearly diffuse to highly specular. Fig. 5 shows results with a high-dynamic range environment map, including crops of representative parts of the image rendered at 64spp. The SZ sampler achieves a meaningful reduction of 1.93× in MRSE, and this error reduction is visually evident in

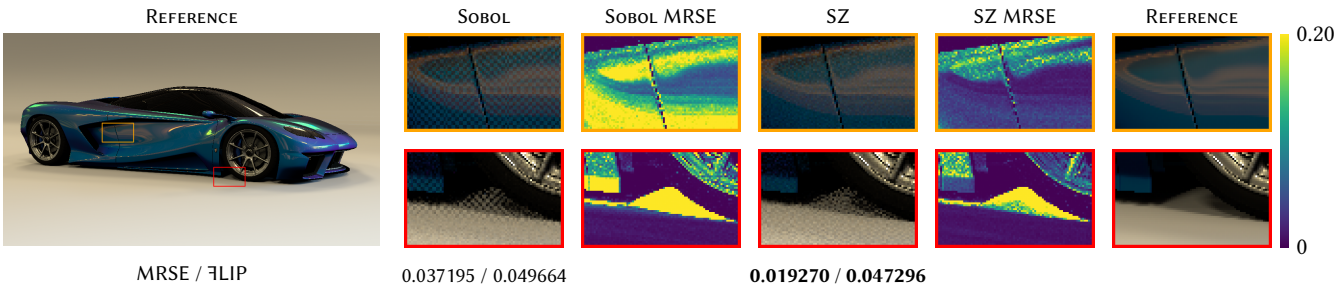


Fig. 5. Comparisons of rendering using the Sobol sequence and using our SZ sequence with the *Hangar Interior* environment map. Rendering was at 64spp with direct lighting only. Crops show representative results; full images are available in our supplemental material.

images. In this case, we can see that SZ does not suffer from the characteristic checkerboard pattern of unconverged Sobol sampling. See Section **??** for additional results with this scene, including a variety of different environment maps, visualizations of error across entire images, and error measurements at additional sampling rates.

At the other extreme, we experimented with a minimal-change, drop-in implementation in *pbrt*-3 by only replacing the Sobol matrices with SZ ones for global sampling and modifying the 2D-based Z-indexed `Sampler` [Ahmed and Wonka 2020] to sample all dimensions at once using SZ sequences. With this setup, we obtained promising results that surpassed Halton and were comparable to Sobol.

Guided by these preliminary tests, we adopt the Z-indexed sampling framework as our preferred model for distributing SZ samples. Our implementation is included in the supplementary materials. In Fig. 6, we compare our SZ sampler with state-of-the-art *pbrt* samplers across various common test scenes. We use the recently introduced Q-ART Owen Scrambling [Ahmed 2025] to shuffle the pixel Z-indices in these examples, but have obtained similar results with hashing [Ahmed and Wonka 2020]. Overall, SZ generally performs favorably, offering a balanced compromise between aliasing reduction and noise suppression at low and high sampling rates, respectively, with only a few exceptions. In particular, it tends to outperform global samplers at low sampling rates and the original 2D-based Z sampler at higher rates. It is also worth noting that the integrator plays a significant role in differentiating the samplers, with variations ranging from substantial differences in direct lighting to barely noticeable differences with BDPT.

Despite the generally strong performance of the SZ-based sampler, we occasionally observe failure cases, for instance, in Fig. 6(H), where a coarse noise artifact appears at relatively low sampling rates, most likely due to a pair of strongly correlated dimensions caused by, for example, splitting a 3D sample between two (0, 4) bands. Avoiding such situations may require more control over the sampler–integrator interaction, which we leave for future research.

5.3 Discrepancy

The essence of Niederreiter's framework [1992, Chapter 4] is that identifying a sequence such as SZ as a $(0, 2^q)$-sequences in base 2^q is sufficient to qualify it as a low-discrepancy sequence. Nevertheless, we ran a test within the feasible range to empirically validate our framework, and obtained the results shown in Fig. 7 that confirm the theoretical predictions. Note that we skipped the first two dimensions, as they are identical to those of Sobol, Faure, and Niederreiter.

5.4 Frequency Spectra

In Fig. 1 we see a comparison between frequency spectra of corresponding pairs of dimensions between SZ and Sobol sequences. Our construction evidently admits better pairwise projections at powers of 2^q octaves, though the discrepancy plots in Fig. 7 suggest that Sobol behaves better at a more granular powers-of-2 octaves. Further, the fact that all consecutive pairs of dimensions of ensembled SZ sequences are dyadic $(0, 2)$-sequences makes them ripe for different optimizations of such sequences like those suggested by Ahmed et al. [2024; 2023] or Doignies et. al. [2024].

5.5 Time and Space Complexity

As a binary-matrix-based construction, the baseline for time and space complexity is that of Sobol sequences, and migrating from Sobol in existing systems is as simple as replacing the matrices. Notably, the matrices are identical for the first two dimensions, thereby preserving any special code for inversion [Grünschloß et al. 2012] or fast execution [Ahmed 2024].

Understanding the anatomy of SZ matrices, however, offers multiple opportunities to optimize space and/or time complexity. For example, in the code listing of Fig. 8(a), we exploit the alpha property to generate a complete set of 256 pairwise-stratified points in 16D, as visualized in Fig. 8(b). In this example, the space complexity is reduced to a single integer `0xB6C8` representing the alpha symbol of the alphabet in Fig. 1(b), while other symbols are evaluated implicitly by computing subsequent dimensions from preceding ones. Combined with Owen- or XOR-scrambling, this could represent a viable GPU-friendly sampling solution for some applications, or a self-contained sample generator for hardware integration and/or embedded systems. Although the example is limited to two octaves and does not use nesting, similar specialized solutions could be tailored for more advanced requirement

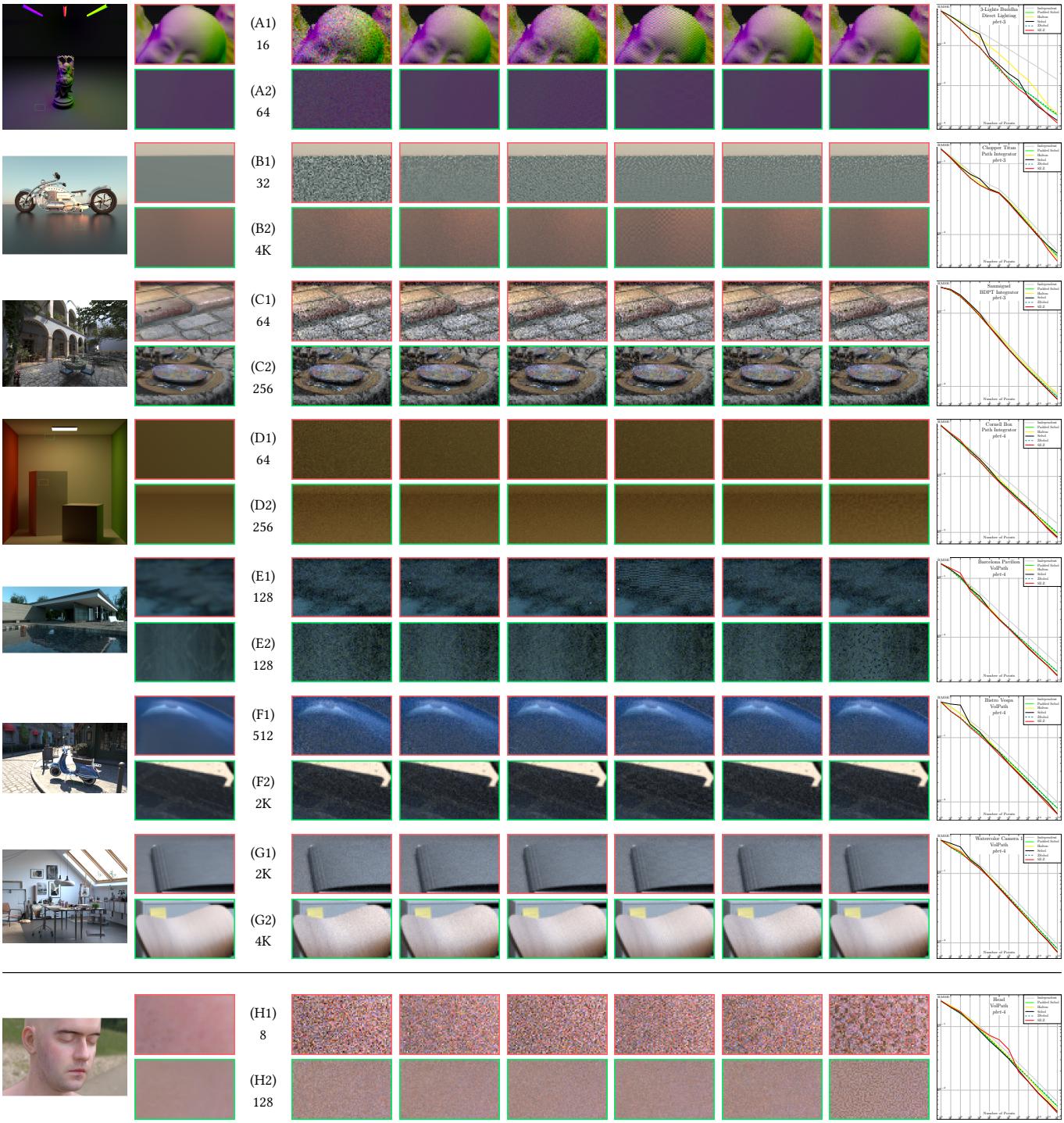
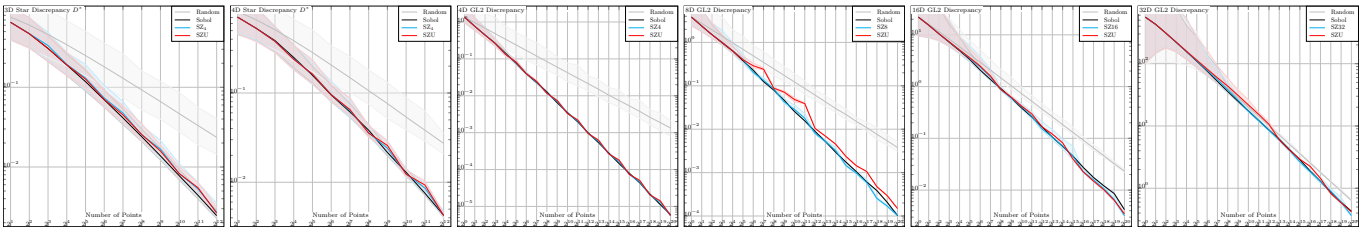


Fig. 6. Example renderings comparing sampling with Z-indexed SZ sequence (SZ-Z) to different state-of-the-art samplers. Specifically, it combines the dithering functionality of ZSobol at low sampling rates with the faster convergence of high-dimensional samplers like Sobol and Halton. We could, however, handpick a few of occasional instances, e.g., (D2) and (E2), where SZ exhibits some noisy artifacts, as well as an example complete scene of "a failure case", (H), where SZ-Z is notably inferior, although it still takes over eventually. The complete scenes are available in the supplementary materials, along with a few more examples.

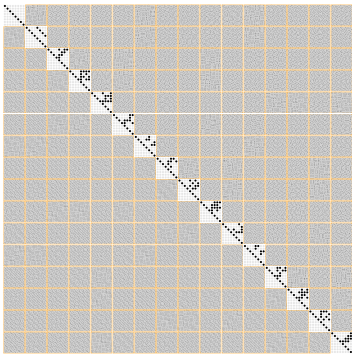


(a) (b) (c) (d) (e) (f)

Fig. 7. (a, b) Star D^* and (c–f) Generalized L2 discrepancy plots comparing different SZ sequences to Sobol, each averaged over Owen-scrambled realizations. The 4D L2 plots align well with the D^* ones, suggesting L2 as a reliable measure of discrepancy. While SZ sequences seem to outperform Sobol in discrepancy measure beyond 4D, it has to be noted that Sobol has the advantage of being "universal"; that is, it is one and the same Sobol sequence compared to different sequences from the SZ family, one for each power-of-two dimensions. The "$\ldots\langle\langle\langle\langle 2\rangle 4\rangle 16\rangle 256\rangle\ldots$"-nested/ensembled is possibly the closest SZ construction to a universal sequence, hence we include it as SZU (SZ-Universal) in these comparisons.

```c
#include <stdio.h>

const int alpha = 0xB6C8;

inline int MxV(int M, int V) {
    return (
        ( V &       1) * ( M       & 0xf) ^
        ((V >> 1) & 1) * ((M >>  4) & 0xf) ^
        ((V >> 2) & 1) * ((M >>  8) & 0xf) ^
        ((V >> 3) & 1) * ((M >> 12) & 0xf)
    );
}

inline int I(int i) { return MxV(0x1248, i); }

void sz16(int alpha, int i, int *p) {
    p[0] = (I(i & 0xf) << 4) | I(i >> 4);
    for (int d = 1, x = I(i >> 4); d < 16; d++) {
        p[d] = p[0] ^ (x << 4);
        x = MxV(alpha, x);
    }
}

int main(int argc,char **argv) {
    int p[16];
    for (int i = 0; i < 256; i++) {
        sz16(alpha, i, p);
        for (int d = 0; d < 16; d++) {
            printf("%3d ", p[d]);
        }
        printf("\n");
    }
}
```

(a) Code



(b) plots of generated points

Fig. 8. (a) A self-contained code example to compute 256 16D points that are pair-wise stratified in all 2D projections. (b) Plots to visualize the generated points without\with Owen scrambling. The diagonal shows the implicit matrices used to compute the points in the dimension of respective rows and columns.

Table 1. Actual rendering times of the (top/bottom) *pbrt*-3/4 scenes in Fig. 6 for different samplers, relative to the independent sampler showing time in seconds. The SZ-Z sampler consistently performs favorably, coming close to the simple quasi-random sampler, but the relative speedup depends on scene complexity, path depth, and integrator, which all determine the proportion of time spent on sampling. For ZSobol in *pbrt*-3 we use the original implementation of Ahmed and Wonka [2020] based on the `PixelSampler` class, which differs slightly from the standard *pbrt*-4 implementation.

Scene	Indpt	Halton	PdSbl	Sobol	ZSbl	SZ-Z
Bdhha	88.5	1.14×	1.06×	1.70×	1.07×	**1.06**×
Chppr Ttn	138.5	1.09×	1.09×	1.14×	1.12×	**1.03**×
Sanmiguel	932.6	1.03×	1.03×	1.04×	1.04×	**1.01**×
Crnll Box	196.4	1.18×	1.19×	1.20×	1.49×	**1.01**×
Brclna Pvln	211.4	1.08×	1.07×	1.09×	1.18×	**1.01**×
Bistro Vspa	373.9	1.05×	1.04×	1.06×	1.11×	**1.01**×
Watercolor	1146.8	1.02×	1.02×	1.03×	1.06×	**1.00**×
Head	158.2	1.12×	1.10×	1.12×	1.26×	**1.01**×

Apart from such specialized hacks and tricks, SZ sequences admit a generic optimization via an "S–P–Z decomposition":

$$\mathcal{P}(a) = \begin{pmatrix} I & & & \\ & a^{-1} & & \\ & & a^{-2} & \\ & & & \ddots \end{pmatrix} \mathcal{P}(I) \begin{pmatrix} I & & & \\ & a & & \\ & & a^{2} & \\ & & & \ddots \end{pmatrix}. \tag{41}$$

The middle Pascal factor can be evaluated using the fast diagonal factoring introduced by Ahmed [2024], while the block-diagonal factors are simply treated as sums of diagonal matrices, which translate naturally to bit-shift-and-mask operations. This reduces the time and space complexity from $O(m)$ for m-bit precision to $O(\log(m))$ for the middle P factor and $O(2q-1)$ for the S and Z ffactors—independent of numeric precision—which is especially significant for 64-bit computations. An implementation is available in our supplementary materials. We achieve 89%/77% memory savings and 20×/15× speedups for the first 16/256 dimensions, respectively, leading to up to 60% speedup over Sobol depending on scene specifications, as shown in Table 1. Further, although evaluating 2D Sobol samples is indeed faster than high-dimensional SZ, our SZ-Z sampler remains faster than ZSobol because ZSobol must shuffle the Z-indices of pixels for each 1D or 2D constituent separately, whereas SZ requires this shuffling only once.

5.6 Coding Complexity

Even though the final implementation amounts to only a few lines of code, implementing a low-discrepancy construction is relatively challenging and prone to mistakes, and SZ sequences are no exception. Acknowledging this, we will make our code publicly available, and aim to maintain libraries for C, Python, and other languages to encourage adoption of these sequences. That said, we still encourage readers to try implementing the sequences themselves, as doing so can deepen their understanding of the underlying properties.

6 CONCLUSION

In this paper, we have enriched the sampling library with a novel construction of binary-based low-discrepancy sequences that scale flexibly with dimension, offering multiple degrees of freedom and control to enable sampling solution architects to tailor sequences to their needs. Originating as an experimental effort, our empirical modeling led us close to the well-established constructions of Faure

and Niederreiter, with the primary difference that we replace elementary fields over (power-of-) prime bases with synthetic fields built over binary matrices, enabling highly efficient computation. More importantly, our model uniquely offers the capability of ensembling a high-dimensional LD sequence from low-dimensional sub-sequences, a feature particularly desirable in graphics applications such as rendering, where the integration domain is largely composed of 2D surfaces.

Our sequences are evidently superior to Halton's for CG applications, and through multiple abstract and rendering tests, we demonstrated that they benchmark competitively with respect to the well-established Sobol sequences. We believe, however, that a deeper analysis of the integrator-sampler interaction is needed before arriving at a final conclusion about the performance of these sequences. We therefore expect this work to spark future research aiming at analyzing, improving, and utilizing these sequences. Finally, we relied mostly on empirical validation, which is sufficient for specific instances, but an analytical proof is still missing for the generic concept, which we hope to see in the near future.

ACKNOWLEDGMENTS

We thank all the anonymous reviewers for their insightful comments. We are also grateful to Art Owen for the insightful discussion, and particularly for highlighting the potential connection to Niederreiter's construction. Beyond the Σ|Z convention for distinguishing left and right elements, the name 'SZ' was selected by the first author as a tribute to the city of Shenzhen, which he was visiting at the time the idea was conceived.

This work was supported in parts by Guangdong S&T Program (2024B01015004), NSFC (U21B2023, 62402323), ICFCRT (W2441020), Shenzhen Science and Technology Program (KJZD20240903100022028, KQTD20210811090044003, RCJC20200714114435012), and Scientific Development Funds from Shenzhen University.

REFERENCES

Abdalla G. M. Ahmed. 2024. An Implementation Algorithm of 2D Sobol Sequence Fast, Elegant, and Compact. In *Eurographics Symposium on Rendering*. The Eurographics Association. https://doi.org/10.2312/sr.20241147

Abdalla G. M. Ahmed. 2025. Q-ART Owen Scrambling. In *Computer Graphics and Visual Computing (CGVC)*. The Eurographics Association. https://doi.org/10.2312/cgvc.20251215

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

Abdalla G. M. Ahmed, Mikhail Skopenkov, Markus Hadwiger, and Peter Wonka. 2023. Analysis and Synthesis of Digital Dyadic Sequences. *ACM Trans. Graph.* 42, 6, Article 218 (Dec. 2023), 17 pages. https://doi.org/10.1145/3618308

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 (Nov. 2020), 15 pages. https://doi.org/10.1145/3414685.3417881

Abdalla G. M. Ahmed and Peter Wonka. 2021. Optimizing Dyadic Nets. *ACM Trans. Graph.* 40, 4, Article 141 (jul 2021), 17 pages. https://doi.org/10.1145/3450626.3459880

Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. 2020. ꟻLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 2 (2020), 15:1–15:23. https://doi.org/10.1145/3406183

Pontus Andersson, Jim Nilsson, Peter Shirley, and Tomas Akenine-Möller. 2021. Visualizing Errors in Rendered High Dynamic Range Images. In *Eurographics Short Papers*. https://doi.org/10.2312/egs.20211015

Paul Bratley, Bennett L. Fox, and Harald Niederreiter. 1992. Implementation and Tests of Low-Discrepancy Sequences. *ACM Trans. Model. Comput. Simul.* 2, 3 (July 1992), 195–213. https://doi.org/10.1145/146382.146385

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

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, 137–145. https://doi.org/10.1145/800031.808590

Bastien Doignies, David Coeurjolly, Nicolas Bonneel, Julie Digne, Jean-Claude Iehl, and Victor Ostromoukhov. 2024. Differentiable Owen Scrambling. *ACM Trans. Graph.* 43, 6, Article 255 (Nov. 2024), 12 pages. https://doi.org/10.1145/3687764

Henri Faure. 1982. Discrépance de Suites Associées à un Système de Numération (en Dimension s). *Acta Arithmetica* 41, 4 (1982), 337–351. http://eudml.org/doc/205851

Henri Faure and Shu Tezuka. 2002. Another Random Scrambling of Digital (t,s)-Sequences. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*. Springer Berlin Heidelberg, 242–256.

Andrew S. Glassner. 1994. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc.

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*. Springer Berlin Heidelberg, 399–408.

John H Halton. 1960. On The Efficiency Of Certain Quasi-Random Sequences Of Points In Evaluating Multi-Dimensional Integrals. *Numer. Math.* 2 (1960), 84–90.

Andrew Helmer, Per Christensen, and Andrew Kensler. 2021. Stochastic Generation of (t, s) Sample Sequences. In *Eurographics Symposium on Rendering - DL-only Track*. The Eurographics Association. https://doi.org/10.2312/sr.20211287

OEIS Foundation Inc. 2015. Sequence A258745: Order of the general affine group AGL(n,2). The Online Encyclopedia of Integer Sequences. https://oeis.org/A258745 Accessed: 2025-01-18.

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.

Stephen Joe and Frances Y. Kuo. 2008. Constructing Sobol Sequences with Better Two-Dimensional Projections. *SIAM J. Sci. Comput.* 30, 5 (Aug. 2008), 2635–2654. https://doi.org/10.1137/070709359

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

Lauwerens Kuipers and Harald Niederreiter. 1974. *Uniform Distribution of Sequences*. John Wiley & Sons. http://opac.inria.fr/record=b1083239 A Wiley-Interscience publication..

Don Mitchell. 1992. Ray Tracing and Irregularities of Distribution. In *Third Eurographics Workshop on Rendering Proceedings*. 61–69.

Harald Niederreiter. 1987. Point Sets and Sequences with Small Discrepancy. *Monatshefte für Mathematik* 104, 4 (1987), 273–337.

Harald Niederreiter. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM.

Victor Ostromoukhov, Nicolas Bonneel, David Coeurjolly, and Jean-Claude Iehl. 2024. Quad-Optimized Low-Discrepancy Sequences. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. ACM, Article 72, 9 pages. https://doi.org/10.1145/3641519.3657431

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*. Springer New York, 299–317.

Loïs Paulin, David Coeurjolly, Jean-Claude Iehl, Nicolas Bonneel, Alexander Keller, and Victor Ostromoukhov. 2021. Cascaded Sobol' Sampling. *ACM Trans. Graph.* 40, 6, Article 275 (dec 2021), 13 pages. https://doi.org/10.1145/3478513.3480482

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. 2019. Efficient Generation of Points that Satisfy Two-Dimensional Elementary Intervals. *Journal of Computer Graphics Techniques (JCGT)* 8, 1 (27 February 2019), 56–68. http://jcgt.org/published/0008/01/04/

Matt Pharr and Greg Humphreys. 2004. *Physically-Based Rendering: from Theory to Implementation* (1st ed.). Morgan Kaufmann Publishers Inc.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2023. *Physically Based Rendering: From Theory to Implementation* (4th ed.). MIT Press.

Il'ya Meerovich Sobol'. 1967. On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* 7, 4 (1967), 784–802.

Shu Tezuka. 1994. A Generalization of Faure Sequences and its Efficient Implementation. *Technical Report, IBM Research, Tokyo Research Laboratory* (1994). https://doi.org/10.13140/RG.2.2.16748.16003

J.G. van der Corput. 1935. Verteilungsfunktionen. *Proceedings of the Nederlandse Akademie van Wetenschappen* 38 (1935), 813–821.

Henry S. Warren. 2012. *Hacker's Delight* (2nd ed.). Addison-Wesley Professional.

A NESTING AND ENSEMBLING WALKTHROUGH

(1) Starting with initial base-4 alphabet:

$$\Sigma_2 = (s_0, s_1, s_2, s_3) = (, , ,) \tag{A.1}$$

(2) Embed each symbol in a 4×4 base-16 one by placing the squared symbol diagonally on a 2×2-block matrix:

$$\Sigma_{\langle 2\rangle} = (\langle s_i\rangle)_{i=0}^{3} = \left(( )^2 , ( )^2 , ( )^2 , ( )^2 \right) = (, , ,) \tag{A.2}$$

Note that $\left(s_2^2, s_3^2\right) = (s_3, s_2)$, hence $\langle s_2\rangle$ contains s_3 blocks and vice versa.

(3) Find a 4×4 "alpha" matrix block whose 15 distinct powers include all non-zero elements of $\Sigma_{\langle 2\rangle}$:

$$\left(( )^i \right)_{i=0}^{14} = (, , , , , , , , , , , , , ,) \ . \tag{A.3}$$

(4) Add the zero symbol to the set of powers, and group the completed alphabet by offset from the embedded $\Sigma_{\langle 2\rangle}$ symbols:

$$\Sigma_4 = \begin{pmatrix} (, , ,) , \\ (, , ,) , \\ (, , ,) , \\ (, , ,) \end{pmatrix} = \begin{pmatrix} (, , ,) + , \\ (, , ,) + , \\ (, , ,) + , \\ (, , ,) + \end{pmatrix} . \tag{A.4}$$

This grouping is unique, up to ordering. That is, the four groups are disjoint. Algorithm 3 achieves this grouping in logarithmic $O(q)$ steps instead of linear $O(2^q)$.

(5) Generate the Pascal matrix for each block symbol in Eq. (A.4), for example:

$$\mathcal{P}( ) = \left(p_{\text{row,col}} : \underbrace{\binom{\text{col}}{\text{row}}}_{\text{mod } 2} ( )^{\text{col}-\text{row}} \right) = \ . \tag{A.5}$$

As per Eq. (A.3), all blocks are powers of $\alpha =$, hence their powers cycle through the same alphabet; for example,

$$( )^{\text{col}-\text{row}} = \left(( )^6 \right)^{\text{col}-\text{row}} = ( )^{6(\text{col}-\text{row})} \ . \tag{A.6}$$

The cycling order, however, varies by the block exponent in Eq. (A.3), leading to different Pascal matrices.

(6) Addition of symbols translates into multiplication of their Pascal matrices, Eq. (34), and multiplication by an upper-triangular matrix on the right generates the same points, only shuffling their order. Therefore, Eq. (A.4) leads to four bands

$$\mathcal{P}(\Sigma_4) = \left(\begin{pmatrix} , , , \\ , , , \end{pmatrix} \begin{matrix} \} \text{generate} \\ \} \text{shuffle} \end{matrix} \right) \tag{A.7}$$

of identical 4D sequences shuffled in different orders, as reflected in the replicated four red-and-cyan 4D blocks in Fig 1(b).

(7) For each band, multiply the four generator matrices from left respectively by $\left(\begin{pmatrix} I & s_i \\ 0 & I \end{pmatrix} \right)_{i=0}^{3} = (, , ,)$:

$$\left(\begin{pmatrix} , , , \\ , , , \end{pmatrix} \right) , \tag{A.8}$$

to restore the respective base-4 Pascal matrices:

$$\left(\begin{pmatrix} , , , \\ , , , \end{pmatrix} \right) , \tag{A.9}$$

which are identical to those in Eq (23). The multiplied factors expand into 4×4-block-diagonal matrices, with all-invertible blocks, hence constitute base-16-rank-preserving Tezuka scramblings.

(8) By a similar reasoning we can transform the four base-4 Pascal matrices into shuffled two-band base-2 Pascals. The initial alphabet is

$$\Sigma_1 = (,) , \tag{A.10}$$

which is already embedded in $(s_0 = ( )^2 , s_1 = ( )^2)$, hence we may proceed to group the base-4 alphabet as

$$\Sigma_2 = ((,) + , (,) +) \tag{A.11}$$

from which follows that the four base-4 Pascals in Eq. (A.9) may, in turn, be refactored into two bands

$$\underbrace{\begin{pmatrix} , \\ , \end{pmatrix}}_{\text{generate}} \times \underbrace{\begin{pmatrix} , \\ , \end{pmatrix}}_{\text{shuffle}} \tag{A.12}$$

of the same 2D sequence shuffled differently. We then multiply the two generators from left respectively by $(,)$ to restore the classic dyadic $(0, 2)$-sequence of Sobol and Faure. Hence, the final set of 16D generator matrices becomes

$$\left(\begin{pmatrix} \begin{pmatrix} , \\ , \end{pmatrix} \times \begin{pmatrix} , \\ , \end{pmatrix} \\ , , , \end{pmatrix} \right) , \tag{A.13}$$

which expands identically to the set of matrices in Fig 1(b).

(9) The process extends analogously to arbitrary large dimensions. In each extension, the existing 2^q matrices are multiplied, in a Cartesian-product sense, by 2^q base-2^{2q} Pascal matrices that shuffle the same 2^q-dimensional sequence in different orders.