Analysis and Synthesis of Digital Dyadic Sequences

ABDALLA G. M. AHMED, KAUST, KSA
MIKHAIL SKOPENKOV, KAUST, KSA
MARKUS HADWIGER, KAUST, KSA
PETER WONKA, KAUST, KSA

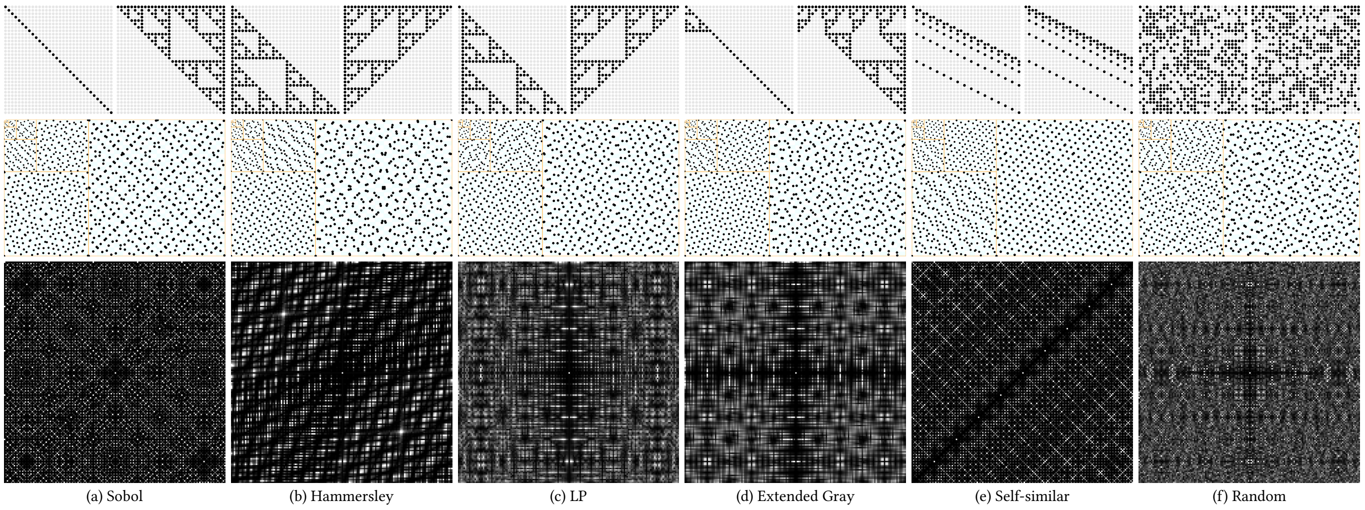


Fig. 1. Generating matrices, points, and periodograms of various digital dyadic sequences. The first 2^m points of the sequences are shown for $m = 0, \ldots, 9$ in separate squares of increasing sizes. The digital dyadic sequences in (b) and (c) are obtained by reordering the known digital dyadic nets with the same names; such reordering is one of the main discoveries of the paper. The sequence in (d) is obtained from a 256-point Gray net by first reordering it into a sequence and then extending the sequence in one of the possible ways. The sequence in (e) represents a new class of self-similar sequences introduced in the paper.

We explore the space of matrix-generated $(0, m, 2)$-nets and $(0, 2)$-sequences in base 2, also known as digital dyadic nets and sequences. In computer graphics, they are arguably leading the competition for use in rendering. We provide a complete characterization of the design space and count the possible number of constructions with and without considering possible reorderings of the point set. Based on this analysis, we then show that every digital dyadic net can be reordered into a sequence, together with a corresponding algorithm. Finally, we present a novel family of self-similar digital dyadic sequences, to be named ξ-sequences, that spans a subspace with fewer degrees of freedom. Those ξ-sequences are extremely efficient to sample and compute, and we demonstrate their advantages over the classic Sobol $(0, 2)$-sequence.

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

Authors' addresses: Abdalla G. M. Ahmed, KAUST, KSA, abdalla_gafar@hotmail.com; Mikhail Skopenkov, KAUST, KSA, mikhail.skopenkov@gmail.com; Markus Hadwiger, KAUST, KSA, markus.hadwiger@kaust.edu.sa; Peter Wonka, KAUST, KSA, pwonka@gmail.com.

Additional Key Words and Phrases: sampling, nets, digital nets, dyadic nets, Sobol sequence, Faure sequence, quasi-Monte Carlo, low-discrepancy sequences, self-similar

ACM Reference Format:
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 (December 2023), 17 pages. https://doi.org/10.1145/3618308

1 INTRODUCTION

Sampling is a fundamental process in computer graphics, underlying Monte Carlo integration in rendering, halftoning, stippling, generative modeling, and object distributions. A wide range of sampling strategies have been developed, and none is conclusively considered the best so far. Low-discrepancy patterns, however, introduced by Shirley [1991] to graphics and popularized mostly by Keller and collaborators, are arguably leading the competition for use in rendering. Of special interest are so-called dyadic nets and sequences, mainly the Hammersley net and the Sobol sequence. These combine exceptionally high production rates with excellent convergence behavior when used in Monte Carlo integration, for which they are specifically designed. They are very versatile, as described in the discussion by Keller [2013]. Furthermore, they are easy to understand thanks to their modular geometric "multi-stratified" structure

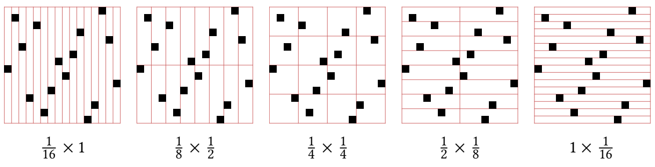

$\frac{1}{16} \times 1$
$\frac{1}{8} \times \frac{1}{2}$
$\frac{1}{4} \times \frac{1}{4}$
$\frac{1}{2} \times \frac{1}{8}$
$1 \times \frac{1}{16}$

Fig. 2. An example 4-bit 16-point dyadic net (i.e., $m = 4$, $N = 16$) showing all $m + 1 = 5$ possible stratifications: $\frac{1}{2^4} \times 1$, $\frac{1}{2^3} \times \frac{1}{2}$, $\frac{1}{2^2} \times \frac{1}{2^2}$, $\frac{1}{2} \times \frac{1}{2^3}$, $1 \times \frac{1}{2^4}$.

[Pharr et al. 2016]. In a nutshell, a dyadic net is concerned with a “fair” 2D distribution of $N = 2^m$ points, so that specifically constructed rectangles of equal area contain exactly one point. These rectangles are obtained by partitioning the unit square $[0, 1)^2$ into rectangular cells of sizes $\frac{1}{2^m} \times 1$, $\frac{1}{2^{m-1}} \times \frac{1}{2}$, . . ., $1 \times \frac{1}{2^m}$ (see Fig. 2). A dyadic sequence is a sequence of sample points such that all leading sub-sequences of points of size 2^m, for all m, are dyadic nets as well as all subsequent sub-sequences of size 2^m (see Fig. 3). We would like to remark that dyadic nets and sequences fulfill the weaker n-rooks and Latin hypercube conditions so that they are proper subsets. In computer graphics, the research effort in the area of dyadic nets and sequences was spearheaded by Keller and collaborators, who early on used and analyzed dyadic distributions in rendering [Grünschloß et al. 2008; Grünschloß and Keller 2009; Keller 2006; Kollig and Keller 2002].

In this paper, we set out to analyze the design space of matrix-generated, or *digital*, dyadic nets and sequences. One possible reason why this design space is still not fully explored is that the focus of the Monte Carlo community might have been on proving discrepancy bounds and not the actual construction of dyadic sequences and dyadic nets. Therefore, important contributions in the area of sample construction were made even recently. Examples include a compact parametrization of the whole dyadic space by Ahmed and Wonka [2021], a compact parametrization of a sub-space comprising dyadic sequences, along with a very efficient generation algorithm, by Helmer et al. [2021], and an all-new construction of a high-dimensional distribution that has pair-wise two-dimensional projections by Paulin et al. [2021].

Our paper makes the following contributions:

- a comprehensive analysis of digital dyadic nets and sequences. We introduce the missing ingredient (Thm. 3.2) that enables us to map the analysis of matrices generating digital dyadic sequences to the language of linear algebra.
- the computation of the exact dimension of the design space for both digital dyadic nets and sequences.
- an algorithm that reorders any digital dyadic net into a digital dyadic sequence. For example, we can reorder the Hammersley net or the Larcher-Pillichshammer net into digital dyadic sequences.
- the discovery of self-similar dyadic sequences, also called ξ-sequences. They are more efficient to construct than the Sobol sequence, and they are easily invertible. The subspace of ξ-sequences is large enough to contain examples with low discrepancy and a large minimal distance between two sample points.

The rest of the paper is organized as follows. In Section 2, we highlight the most relevant literature. In Section 3, we define and analyze digital dyadic nets and sequences, recall some known and state our new theoretical results: compute the exact dimension of the design space for both and give an algorithm that reorders any digital dyadic net into a digital dyadic sequence. In Section 4, we apply our algorithm for several examples: we reorder the Hammersley, the Larcher–Pillichshammer, and the Gray nets into digital dyadic sequences. In Section 5, we introduce and study new self-similar dyadic sequences and give an explicit parametrization of their design space.

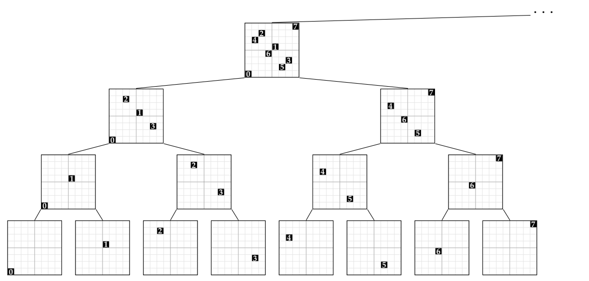


Fig. 3. The hierarchical structure of a dyadic sequence is visualized as a tree. Note that the individual points also represent (0, 0, 2)-nets (i.e., 1-point dyadic nets). All sub-sequences corresponding to an internal (or leaf) node in the tree are required to be dyadic nets. Here the gray midlines are shown for better appearance.

2 RELATED WORK

Our work has interesting connections to different fields of study in computer graphics and mathematics. In this section, we try to highlight the most relevant literature, organized into three categories.

2.1 Low-Discrepancy Sampling

The study of uniform sample distributions is much older than computer graphics. An early 1D sequence was presented by van der Corput [1935], who established the digit-reversal principle that underlies almost all Low-Discrepancy (LD) constructions. Roth [1954] and Hammersley [1960], respectively, presented techniques for extending the van der Corput principle to two and higher dimensions.

In a parallel line of study, Sobol [1967] pioneered the *digital* construction of LD sets and sequences, generalized by Niederreiter [1987; 1992], who coined the terms (t, m, s)-nets and (t, s)-sequences, and initiated a new field of study of these constructions. We defer definitions to Section 3, but point out that nets are finite LD sets, while sequences are extensible sets that admit more points while maintaining a low discrepancy. Most of these constructions are *digital*, i.e. sample coordinates are derived from the sequence number via binary matrix vector multiplication and vector addition modulo 2. A thorough study of common construction methods is given by Dick and Pillichshammer [2010]. Not all nets and sequences, however, are necessarily digital, and there are known algorithms for constructing non-digital nets [Ahmed and Wonka 2021; Grünschloß and Keller 2009].

Aside from direct construction techniques of nets and sequences, Owen [1995; 1998] presented a powerful technique to randomize (t, m, s)-nets and (t, s)-sequences while preserving their favorable properties; in fact improving their quality. The original concept by Owen, while quite simple, is very memory intensive, consuming as many random bits—per dimension—as the number of samples. Much work followed Owen's, mainly aimed at simplifications for more efficient implementation [Burley 2020; Friedel and Keller 2002; Kollig and Keller 2002], at the cost of different compromises. However, there are also variants that are conceptually different; most notably the techniques of Faure and Tezuka [2002] aimed at shuffling the order of the samples rather than scrambling their locations.

Owen's scrambling works only on existing nets and sequences, so a method for generating the underlying sets is still required. Ahmed and Wonka [2021] presented a technique for the direct construction of arbitrary 2D nets, but their method can only produce nets and consumes even more random bits than Owen's scrambling.

2.2 Recursive Tiling Techniques

Ostromoukhov [2007] is credited for importing the digit-reversal principle of LD constructions (Section 2.1) and using it to distribute blue-noise samples over a self-similar set of tiles. The idea was followed up by Wachtel et al. [2014], and matured in Ahmed et al. [2017], whose tile set, ART, has a low granularity and tuneable properties, presenting a comparable blue-noise competitor to LD sequences [Ahmed and Wonka 2021; Christensen et al. 2018; Kopf et al. 2006].

2.3 Low-Discrepancy Blue Noise

A recent trend of research tries to inject blue noise properties into LD sets and/or sequences. Early ideas along this line of research may be traced back to Keller [2006], who speculated on the superiority of LD sets that bear a large minimum distance between points, showcasing the very-low-discrepancy family of nets by Larcher and Pillichshammer [2003] that also attain a large minimum distance between points. Grünschloß and colleagues subsequently presented examples of small-sized digital nets with maximized minimum distance between points obtained via a search algorithm [2008], as well as non-digital constructions developed heuristically [2009]. The recent work of Christensen et al. [2018], along with [Helmer et al. 2021; Pharr et al. 2016], may be seen as an extension of this search approach that targets sequences instead of nets.

A different line of research in this category was that of Ahmed et al. [2016], with an algorithm for proactively imposing spectral control over LD nets, at the cost of compromising the quality of the net. Perrier et al. [2018] extended this idea to sequences. See a survey by Singh et al. [2019] for details. Ahmed and Wonka [2021] demonstrated the possibility of incorporating spectral control into nets without compromising their quality but highlighted the difficulty of extending the idea to sequences.

3 ANALYSIS OF DIGITAL DYADIC NETS AND SEQUENCES

In this section, we define and analyze digital dyadic nets and sequences, and state our new theoretical results: computation of the exact dimension of the design space for both and an algorithm that reorders any digital dyadic net into a digital dyadic sequence.

3.1 Definitions

There is a general notion of (t, m, s)-nets in base b. In this paper, we are focused on the particular case of 2D dyadic nets. They are known as $(0, m, 2)$-nets in base 2 (in short, m-nets, or 2^m-point nets). Further, we specifically study dyadic sequences, also known as $(0, 2)$-sequences in base 2.

Definition 3.1. *By a* rectangle $a \times b$ *we mean the set of points*

$$\{ (x, y) \in \mathbb{R}^2 : x_0 \le x < x_0 + a,\ y_0 \le y < y_0 + b \} \quad \text{for some } x_0, y_0.$$

Consider $m + 1$ possible stratifications of the unit square $[0, 1)^2$ into equal rectangles $\frac{1}{2^m} \times 1$, $\frac{1}{2^{m-1}} \times \frac{1}{2}, \dots, 1 \times \frac{1}{2^m}$ (strata)*. See Fig. 2. A 2^m-point set in $[0, 1)^2$ is a* dyadic net *if it contains a single point per stratum in each of the $m + 1$ stratifications.*

A dyadic sequence *is a finite or infinite sequence of points in $[0, 1)^2$ such that the first 2^m points, the next 2^m points, and all subsequent blocks of 2^m points are dyadic nets for all $m = 0, 1, \dots$.*

Thus, infinite dyadic sequences comprise sequences of dyadic nets of all sizes. Any dyadic sequence can be visualized as a binary tree, where each node corresponds to a dyadic net. See Fig. 3.

Now we turn to *digital dyadic nets* [Niederreiter 1987]. Hereafter all the computations use linear algebra over the Galois field $GF(2)$, also denoted by $\mathbb{Z}/2\mathbb{Z}$ or $\mathbb{Z}_2$: all vector and matrix entries $\in \{0, 1\}$ and $1 + 1 = 0$. Computation in $GF(2)$ is different from computing with binary matrices over the real numbers $(1 + 1 = 2)$, or Boolean algebra $(1 + 1 = 1)$. In what follows, a *matrix* refers to an $m \times m$ matrix with the entries in $GF(2)$ unless otherwise indicated.

Definition 3.2. *For two bit vectors $X = (x_1, \dots, x_m)$ and $Y = (y_1, \dots, y_m)$, denote $\rho(X, Y) := (0.x_1 \dots x_m, 0.y_1 \dots y_m) \in [0, 1)^2$. An ordered collection of 2^m distinct points (X_1, Y_1), (X_2, Y_2), … is called* digital *if*

$$(X_i, Y_i) = \rho(C_x S_i, C_y S_i), \tag{1}$$

where (C_x, C_y) is a fixed pair of binary $m \times m$ matrices, S_1, S_2, … enumerate all the binary m-bit column vectors in the order $(0, \dots, 0)$, $(1, 0, \dots, 0)$, $(0, 1, \dots, 0), \dots, (1, \dots, 1)$, and the products are modulo 2.

A digital collection of points that is also a dyadic net (sequence) is called a digital dyadic net (sequence).

Equivalently, based on the analysis from [Ahmed and Wonka 2021], a digital collection of points is a *digital dyadic net*, if it contains a unique point $\rho(X, Y)$ with any given last r bits of X and last $m - r$ bits of Y, for each $r = 0, 1, \dots, m$. Likewise, a digital collection of points is a *digital dyadic sequence*, if the first and all the subsequent blocks of 2^k points form a digital dyadic net after removing the last $m - k$ bits from both X and Y, for each $k < m$.

This notation enables us to express well-known sequences and nets in succinct form; the map "ρ" is usually omitted. Table 1 summarizes explicit parametrizations of digital dyadic nets and sequences, and the analysis of how many different constructions are possible by each parametrization. The GFaure construction introduced by Tezuka [1994] describes a family of sequences obtained by setting $C_x = L_x$ and $C_y = L_y P$, with L_x and L_y arbitrary lower unitriangular matrices and P the binary Pascal matrix. By a *unitriangular* matrix we mean an upper or lower triangular matrix with all ones

Table 1. **Overview of constructions** of different digital dyadic nets and sequences. L_x, L_y are lower unitriangular matrices, U_x, U_y are upper unitriangular matrices, J is the anti-diagonal matrix with all ones on the anti-diagonal, P is the binary Pascal matrix, M is any invertible matrix, X_0, Y_0 are binary bit vectors, and S is the index of the generated point (X, Y) in binary bit vector form. See Section 3 for details.

Name of construction	Explicit construction	# constructions (counting permutations)	# constructions (not counting permutations)
Digital dyadic nets	$(X, Y) = (M, L_y U_y JM)S$	$2^{3m(m-1)/2}(2^1-1)(2^2-1)\cdots(2^m-1)$	$2^{m(m-1)}$
GS-nets	$(X, Y) = (L_x M, L_y PM)S$	$2^{3m(m-1)/2}(2^1-1)(2^2-1)\cdots(2^m-1)$	$2^{m(m-1)}$
Digital dyadic sequences	$(X, Y) = (L_x U_x, L_y PU_x)S$	$2^{3m(m-1)/2}$	$2^{m(m-1)}$
GFaure sequences	$(X, Y) = (L_x, L_y P)S$	$2^{m(m-1)}$	$2^{m(m-1)}$
Hammersley-like nets	$(X, Y) = (J, L_y U_y)S$	$2^{m(m-1)}$	$2^{m(m-1)}$
Affine digital dyadic nets	$(X, Y) = (M, L_y U_y JM)S + (X_0, Y_0)$	$2^{m(3m+1)/2}(2^1-1)(2^2-1)\cdots(2^m-1)$	2^{m^2}
Affine digital dyadic sequences	$(X, Y) = (L_x U_x, L_y PU_x)S + (X_0, Y_0)$	$2^{m(3m+1)/2}$	2^{m^2}

Table 2. **Overview of constructions** of particular digital dyadic nets and sequences. I and J are the identity and anti-diagonal matrices, respectively. P is the binary Pascal matrix. U_{LP} and L_{LP} are given by (3) and (11). S is the index of the generated point (X, Y) in binary bit vector form.

Name of construction	Explicit construction
Sobol (Faure) sequence	$(X, Y) = (I, P)S$
Hammersley net	$(X, Y) = (J, I)S$
Hammersley sequence	$(X, Y) = (JPJ, PJ)S$
LP net	$(X, Y) = (J, U_{\mathrm{LP}})S$
LP sequence	$(X, Y) = (L_{\mathrm{LP}}, PJ)S$

on the diagonal. The entries of the *Pascal matrix* P are $P_{ij} = \binom{j-1}{i-1}$ mod 2, with i row and j column index, respectively:

$$P = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \cdots \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \cdots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \cdots \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{2}$$

Hereafter a zero entry is in gray for readability.

Table 2 summarizes our newly introduced (see Section 4) as well as known constructions for reference, e.g. the Hammersley net is obtained by $C_x = J, C_y = I$, with J the anti-diagonal matrix with all ones on the anti-diagonal, and the Larcher–Pillichshammer (LP) net uses $C_x = J, C_y = U_{\mathrm{LP}}$ with

$$U_{\mathrm{LP}} = \begin{pmatrix} 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}. \tag{3}$$

Here $C_x = J$ ensures that the points are ordered so that their x-coordinates are increasing. A clever construction of digital dyadic sequences has been suggested by [Sobol' 1967]; see a nice exposition by [Bratley and Fox 1988]. The simplest and the most popular 2D Sobol sequence uses $C_x = I, C_y = P$. It was studied by [Faure 1982] and is often called simply the *Sobol sequence*; see [Pharr et al. 2016].

In the following, we first discuss digital dyadic nets and then digital dyadic sequences. In order to determine if two pairs of matrices (C_x, C_y) and (C'_x, C'_y) generate the same set of points in a different order, we use the concept of the *characteristic matrix.*

Definition 3.3. *For a pair of invertible matrices (C_x, C_y), the matrix $C = C_y C_x^{-1}$ is called the* characteristic matrix *of the pair.*

If we want to check if two pairs of matrices generate the same set of points, we simply compare their characteristic matrices. The characteristic matrix directly associates the x-coordinate of a point with its y-coordinate, i.e $Y = CX$ when X and Y are written in appropriate binary form. We also note that changing a pair of matrices (C_x, C_y) to another pair $(C_x M, C_y M)$ with an invertible matrix M preserves the characteristic matrix C and enumerates all possible transformations that change the order of points but keep the final net the same. This transformation is equivalent to replacing S by MS in (1), i.e., to a reordering of the bit vectors S.

In particular, if M is upper unitriangular, then for each $k = 1, \ldots, m-1$ this reordering preserves the decomposition of the sequence of 2^m possible bit vectors S into blocks of 2^k consecutive vectors: only the vectors within each block and the whole blocks are reordered. Such reordering preserves the dyadic sequence property: a digital dyadic sequence is reordered to another digital dyadic sequence. Conversely, one can show that a matrix M having this property must be upper unitriangular.

We will also need the following definition for a type of matrix that is integral to understanding the digital construction.

Definition 3.4. *A* progressive matrix *is a square matrix whose leading principal sub-matrices are invertible.*

In this context, we should also comment on an essential fact of linear algebra in $GF(2)$. A matrix is progressive if and only if it can be factored into the product of a lower unitriangular matrix L and an upper unitriangular matrix U [Horn and Johnson 1985, Corollary 3.5.5]. The factorization is also unique. So we only have $m(m-1)$ bits to determine a progressive matrix. This is m bits less than required to determine an arbitrary $m \times m$ matrix. Finally, there are some interesting facts about the Pascal matrix that will be helpful: the Pascal matrix is upper unitriangular; P is its own inverse; mirroring both rows and columns by multiplying with J on each side gives JPJ, which is a lower unitriangular matrix; and $JPJ = PJP$ [Kajiura et al. 2018, Lemma 2.5]. See Appendix B for short proofs.

Also some comments on (numerical) linear algebra in $GF(2)$. Matrix inversion generally works by adapting the Gauss–Seidel algorithm. The LU-factorization algorithm can also be adapted. See Algorithm 5 in the appendix. The QR-factorization using Gram–Schmidt does not work, because vectors can be self-orthogonal. We make heavy use of determinant calculations, in particular, the Laplace expansion is employed in many proofs. As mathematics books treat linear algebra over general fields, we believe that a specific introduction to linear algebra over $GF(2)$ is more likely to be found in engineering textbooks, e.g. [Bard 2009]. We also found packages for the numerical implementations in Python and C++, but ultimately reimplemented all algorithms from scratch.

3.2 Digital Dyadic Nets

We first would like to establish the conditions that need to be satisfied for a pair of matrices (C_x, C_y) to form a digital dyadic net. We introduce the term *dyadic pair of matrices* for the following discussion.

Definition 3.5. *A* dyadic pair of matrices (C_x, C_y) *is a pair of matrices such that the* hybrid *matrices*

$$H_r = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-r,1} & a_{m-r,2} & \cdots & a_{m-r,m} \\ b_{1,1} & b_{1,2} & \cdots & b_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{r,1} & b_{r,2} & \cdots & b_{r,m} \end{pmatrix}, \tag{4}$$

comprising the first $m-r$ rows from $C_x = (a_{ij})$ and the first r rows from $C_y = (b_{ij})$, are invertible for all $r \le m$.

It is well established [Grünschloß et al. 2008; Larcher and Pillichshammer 2001] that two binary matrices (C_x, C_y) form a dyadic pair if and only if they produce a digital dyadic net with Eq. (1). However, even though the conditions are known, they are not sufficient to derive an explicit construction or a complete description of the design space which we will now recall.

Theorem 3.1. *(See [Hofer and Suzuki 2019, Lemma 2.6]) A pair of matrices (C_x, C_y) is dyadic if and only if C_x is invertible, and $C_y = LUJC_x$ for some lower unitriangular matrix L and some upper unitriangular matrix U.*

This explicit parametrization enables the possibility to enumerate all possible constructions directly, and to sample from possible constructions without having to check that the condition for a dyadic pair is satisfied. While it is also possible to construct and sample from general invertible matrices C_x over $GF(2)$, this introduces unwanted complexity that will be eliminated shortly. Luckily, for nets we do not need to distinguish between constructions that create the same points in a different order. Therefore, nets can be conveniently constructed in this canonical order, i.e. by ignoring C_x or by using the parametrization of Hammersley-like nets.

We remark that the parametrization in Theorem 3.1 is actually symmetric in C_x and C_y: if $C_y = LUJC_x$, then $C_x = L'U'JC_y$ with $L' = JU^{-1}J$ and $U' = JL^{-1}J$. Here L' and U' are lower- and upper-unitriangular respectively because the left- and right-multiplication by J reverses the order of rows and columns respectively.

Now we ask the question of *how many constructions are possible by the original construction.* We can determine that there are

$$2^{m(m-1)} \cdot 2^{m(m-1)/2} \cdot (2^1-1)\cdots(2^m-1)$$

possible constructions. The first factor of the product refers to the choices due to enumerating the matrices L and U, and the rest to enumerating all invertible matrices C_x. Such enumeration is given by the following known construction algorithm for invertible matrices over $GF(2)$: As the first column, pick any of the 2^m-1 nonzero m-bit vectors. As the second column, pick any of the 2^m-2 binary vectors not proportional to the first column. As the third column, pick any of the 2^m-2^2 vectors not contained in the linear span of the first two columns, etc.

Next, we wish to answer the question of *how many unique sets of points can be generated.* This means we are interested in the number of constructions without considering (counting) permutations. We can determine that there are

$$2^{m(m-1)} \tag{5}$$

possible constructions by observing that the characteristic matrix has the form $C = LUJ$. The number of constructions directly follows from the number of bits we can choose for designing arbitrary unitriangular matrices L and U. These results are included in Table 1. We would like to relegate the detailed proofs to Appendix A. While the proofs are not necessary to understand the main results in the paper, the ones for dyadic nets provide important insights for the case of dyadic sequences (the more difficult case and main focus of the paper).

3.3 Digital Dyadic Sequences

To discuss digital dyadic sequences we first need the following definition.

Definition 3.6. *A* progressive pair *of matrices is a pair of matrices such that the $k \times k$ leading principal sub-matrices form dyadic pairs for all k. In other words, a pair $(C_x, C_y) = (a_{ij}, b_{ij})$ is* progressive, *if the* hybrid *matrices*

$$H_{k,r} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k-r,1} & a_{k-r,2} & \cdots & a_{k-r,k} \\ b_{1,1} & b_{1,2} & \cdots & b_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{r,1} & b_{r,2} & \cdots & b_{r,k} \end{pmatrix}, \tag{6}$$

are invertible for all $r \le k \le m$.

Note that this definition implies that each of the two matrices is progressive. The concept of a progressive pair of matrices is the missing link that is required to fully map the analysis of digital dyadic sequences to the language of linear algebra. We establish the link as follows.

Theorem 3.2. *A pair of matrices (C_x, C_y) creates a digital dyadic sequence with* (1) *if and only if (C_x, C_y) is a progressive pair.*

This new theorem follows from our definitions: the last constraint in the paragraph after Definition 3.2 is a system of linear equations with the matrix being exactly hybrid matrix (6) from Definition 3.6. However, the resulting assertion is nontrivial; let us illustrate the point. We first start by analyzing the generation of a 1D digital dyadic sequence by a single matrix C_x. This matrix is then multiplied by a matrix S_{all} that consists of all possible inputs, i.e. all m bit integers in sequence as binary numbers $[0, 1, 2, 3, 4, 5, \ldots]$. Now we

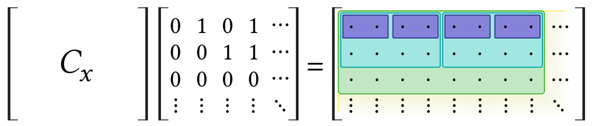


Fig. 4. A visualization to support the proof of Theorem 3.2. If an $m \times m$ generating matrix for a 1D digital dyadic sequence is multiplied with the matrix that encodes the sequence of integers $[0, 1, 2, 3, \dots]$ in binary, the output matrix must ensure that specific blocks in the output form $2^1, 2^2, \dots, 2^m$-point nets. However, to establish the net property for a 2^k-point net, we only need to look at the first k bits (k rows) of the output. In the output we visualize the blocks of bits that need to be checked to verify the 2^1-point (blue color), 2^2-point (cyan color), and 2^3-point (green color) net property.

analyze the output matrix shown in Fig. 4. In order for the matrix C_x to generate a valid 1D dyadic sequence, the blue boxes need to contain permutations of 0, 1, the cyan permutations of 00, 10, 01, 11, the green boxes permutations of 000, 100, 010, . . ., and so on for all k bit integers up to m. This analysis can be extended to a pair of generating matrices (C_x, C_y) by observing that the first 2^k points need to form a 2^k-point net. All subsequent blocks of 2^k points are also dyadic nets; this is a property of XOR-scrambling, which is a bit harder to see. We can now analyze the design space by analyzing the properties of progressive pairs of matrices.

The known explicit parametrization of the design space of digital dyadic sequences can be elegantly stated in terms of progressive pairs as follows.

Theorem 3.3. *(See [Hofer and Suzuki 2019, Theorem 1.2].) A pair of matrices (C_x, C_y) is progressive if and only if $C_x = L_x U$ and $C_y = L_y P U$ for some upper unitriangular matrix U and some lower unitriangular matrices L_x, L_y.*

Equivalently, $(C_x, C_y) = (L_x U_x, L_y U_y)$ is progressive, if and only if $U_y U_x^{-1} = P$, where we LU-decompose each matrix. The latter is equivalent to $U_x U_y^{-1} = P$ because $P = P^{-1}$, thus the condition is symmetric in C_x and C_y.

A new short proof of the theorem is given in Appendix B. Just like in the case of dyadic nets, this explicit parametrization enables the possibility to enumerate all possible constructions directly and to sample from possible constructions. Again, we ask the question of *how many constructions are possible.* We can determine that there are

$$2^{3m(m-1)/2}$$

possible constructions. The number of constructions directly follows from the number of bits we can choose for designing arbitrary unitriangular matrices L_x, L_y, and U. Next, we wish to answer the question of how many unique sets of points can be generated. This means we are interested in *the number of constructions without considering permutations.* We can determine that there are

$$2^{m(m-1)} \tag{7}$$

possible constructions by observing that the characteristic matrix has the form $C = L_y P L_x^{-1}$. The number of constructions follows from the number of bits we can choose for designing arbitrary unitriangular matrices L_x and L_y. We therefore propose to use the GFaure construction to explore the complete design space of digital dyadic sequences in case the order in which the points are generated is not important (the GFaure construction ensures that the canonical order is a sequence order, though). While the construction was proposed before, it was not clear however that this particular construction was able to enumerate all possible sequences up to the order of the points. In addition, for the reduced parameterization to be valid, we need to ensure that distinct choices of the pair (L_x, L_y) lead to distinct characteristic matrices $C = L_y P L_x^{-1}$. This is a somewhat nontrivial property proved in Appendix B, along with the other results of this subsection.

3.4 Converting Digital Dyadic Nets to Digital Dyadic Sequences

An important discovery of our work is that all digital nets can be reordered to become digital sequences. We think this is quite striking, since digital nets like the Hammersley net have generally been considered to be non-extensible [Keller 2013; Pharr et al. 2016], whereas they actually are. Also, important digital nets like the LP-net can actually be reordered to become a sequence, a much more useful order for sampling applications.

When comparing the number of unique point sets that can be generated by the digital dyadic net construction (5) and the digital dyadic sequence construction (7), we notice that they can generate the same number of unique point sets. Therefore, we can conclude that every digital net can be reordered into a digital sequence. We would like to note that this conclusion may be a bit more complex than it seems. The conclusion is only possible because our enumeration is proven to be exhaustive and proven not to contain any duplicates. This justifies the proofs given in the appendix and also requires the joint treatment of digital nets and sequences, even though the original goal was just to understand sequences.

We first describe the process in the form of linear algebra, and later comment more explicitly on an algorithmic implementation.

Theorem 3.4. *For each dyadic pair (C_x, C_y) there is an invertible matrix M such that $(C_x M, C_y M)$ is a progressive pair.*

In other words, the dyadic and progressive pairs have the same sets of characteristic matrices, and each digital dyadic net can be ordered to provide a digital dyadic sequence.

The matrix M is explicitly constructed from the decomposition $C_y = LUJC_x$ provided by Theorem 3.1. A possible choice is

$$M = C_x^{-1} J U^{-1} P J.$$

The resulting pair has the form

$$(C_x M, C_y M) \quad = \quad (JU^{-1}PJ, LPJ) \quad = \quad (\underbrace{JU^{-1}PJ}_{L_x}, \underbrace{LJPJ}_{L_y} \cdot P)$$

by the identity $PJ = JPJP$. Here $L_x = JU^{-1}PJ$ and $L_y = LJPJ$ are lower unitriangular matrices because for any upper unitriangular matrix U' the matrix $JU'J$ is lower unitriangular. So we arrive at the decomposition $(C_x M, C_y M) = (L_x, L_y P)$ from Theorem 3.3 and therefore the pair is progressive.

Based on this result, we propose the following construction for dyadic pairs, called *GS-net*:

$$(C_x, C_y) = (L_x M, L_y P M), \tag{8}$$

Algorithm 1: Converting a digital dyadic net into a digital dyadic sequence

```
Input: a pair of matrices $(C_x, C_y)$ that are known to create
    a digital dyadic net with (1);
Output: a pair of matrices $(C_{xnew}, C_{ynew})$ that creates a
    digital dyadic sequence with the same points.
1 Compute $C_x^{-1}$ and the characteristic matrix $C = C_y C_x^{-1}$;
2 Compute $C' = CJ$, flipping the columns of $C$;
3 Compute the LU-factorization of $C'$ yielding $L$ and $U$
    unitriangular matrices using Algorithm 5 in the appendix;
4 Return the two matrices $C_{xnew} = JU^{-1}PJ$ and $C_{ynew} = LPJ$.
```

where L_x, L_y are arbitrary lower unitriangular matrices and M is an arbitrary invertible matrix. This makes it clear that every digital dyadic net is simply a digital dyadic sequence reordered by an arbitrary invertible matrix.

We make the reordering explicit in Algorithm 1. It requires the computation of an inverse matrix and the computation of the LU-factorization in $GF(2)$. We implemented this ourselves, but libraries are available. Finally, we would like to recall that each dyadic sequence can be reordered by an upper unitriangular matrix U. Therefore, the reordering of a net into a sequence is not unique and the matrices returned by our proposed algorithm can still be right-multiplied by an arbitrary upper triangular matrix U.

3.5 Affine Digital Dyadic Nets and Sequences

Seeking to enlarge the space of digital dyadic nets and sequences, we apply XOR scrambling introduced by Kolleg and Keller [2002]. Namely, instead of (1), we use equation

$$(X, Y) = (C_x S, C_y S) + (X_0, Y_0) \tag{9}$$

with some fixed bit vectors X_0, Y_0. This gives $2m$ additional degrees of freedom if we take the order of points into account.

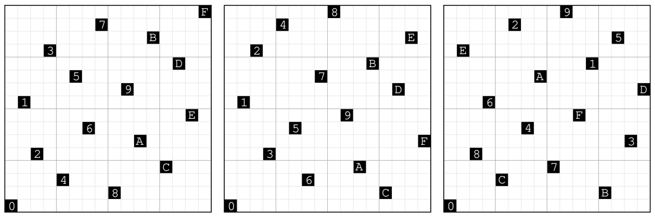
Then we rewrite (9) as

$$Y = C_y C_x^{-1}(X + X_0) + Y_0 = C_y C_x^{-1} X + Y_0'$$

for some constant bit-vector Y_0'. This means that for any pair (X_0, Y_0) of constant bit-vectors used to XOR-scramble a digital dyadic sequence, there exists a single bit vector Y_0' that would produce the same set of points by scrambling only along the y axis. This means that the actual dimension of xor-scrambling is only m. Note that this Y_0' vector may not be absorbed within any matrix multiplication, so it represents a new degree of freedom. Incidentally, it exactly compensates the m diagonal bits we lost earlier. This makes the dimension of affine digital dyadic sequences exactly m^2, allocated as $m(m-1)/2$ bits below the diagonal of each of the C_x and C_y matrices, and m bits for XOR scrambling of y.

4 SYNTHETIC SEQUENCES

In this section, we demonstrate examples of applying the knowledge gained through Section 3 to synthesizing dyadic sequences with favorable properties, and in the following section we give a third demonstration of a whole class of sequences.



(a) Hammersley (b) LP (c) Gray

Fig. 5. The first 16-point nets of 256-point (a) Hammersley (b) LP (c) Gray sequences, with possible indexings (in hexadecimal) of the points.

4.1 Designing Dyadic Nets and Sequences by Search and Optimization

For applications of Monte Carlo integration, such as rendering, one often searches for digital dyadic nets and sequences that are optimal in a sense. Common optimization targets include minimum distance between points and star discrepancy [Keller 2006]. *Star discrepancy* D^* measures how the fraction $n(x, y)/2^m$ of points of an m-net in a rectangle $[0, x) \times [0, y)$ deviates from the area of the rectangle; formally, $D^* = \sup |xy - n(x, y)/2^m|$, where the supremum is over all x, y in $[0, 1]$. There are multiple simple ways to search for dyadic nets and sequences with specific properties. For example, for a small bit depth ($m = 4, N = 16$), it is feasible to perform an exhaustive search over all digital dyadic nets or sequences. For larger bit depths, one can use heuristic search, e.g. hill climbing. At each iteration, all possible one bit mutations to an existing net or sequence can be evaluated and the optimum mutation can be chosen until no mutation brings an improvement in the objective. One interesting possibility is a hierarchical search. Due to the incremental nature of nets and sequences we may optimize progressively, e.g. first searching for a k bit solution and then optimizing for additional bits while retaining the leading samples (bits) in the net or sequence.

The important aspect of our work is that any of these explorations can be done efficiently. The complete characterization of the design space for nets and sequences enables us to only visit candidates, e.g. generator matrices for a digital construction, that are viable. This improves over previous work that explored a much larger set of constructions and then filtered out invalid candidates. E.g. Grünschloß et al. [2008] use heuristic search for alternative C_x matrices to produce Hammersley-like nets. Since the percentage of viable candidates can be small, a search using our explicit construction can bring orders of magnitude speed-up.

Instead of searching for, e.g. new pairs (C_x, C_y), different dyadic nets are typically derived from these, especially Sobol, by applying the dyadic-preserving Owen's scrambling [1998]. Faithful application of Owen's scrambling is computationally challenging, and common implementations offer different trade-offs between speed, quality, and flexibility [Burley 2020; Helmer et al. 2021; Kollig and Keller 2002]. Our explicit characterization and understanding of sequences provide multiple advantages. It is hard to predict the properties of an Owen scrambled sequence. The above-mentioned hierarchical search would not be possible. Also, there are advantages when creating nets from sequences and distributing them over pixels, see e.g. PBRT [Pharr et al. 2016] or Ahmed and Wonka [2020]. In Fig. 1, we show example digital dyadic sequences.

4.2 Hammersley Sequences

A Hammersley net is generated by a pair of matrices (J, I). Executing Algorithm 1 by manual computation, we arrive at the progressive pair of matrices (JPJ, PJ). For example, the 256-point Hammersley net can be sequenced using the matrix pair

$$\left(\begin{pmatrix} 1&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 1&0&1&0&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&0&0&0&1&0&0&0\\ 1&1&0&0&1&1&0&0\\ 1&0&1&0&1&0&1&0\\ 1&1&1&1&1&1&1&1 \end{pmatrix}, \begin{pmatrix} 1&1&1&1&1&1&1&1\\ 1&0&1&0&1&0&1&0\\ 1&1&0&0&1&1&0&0\\ 1&0&0&0&1&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&0&1&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0 \end{pmatrix}\right). \tag{10}$$

See Fig. 5(a) for the layout of the first 16-point net in the 256-point sequence with possible indexing.

4.3 LP Sequences

While the Hammersley nets have the worst known star discrepancy among digital dyadic nets, the LP nets of Larcher and Pillichshammer [2003] are believed to have the best one according to numerical experiments measuring the star discrepancy. We use 2^{2^k}-point LP nets as a second example of converting a net into a sequence.

An LP net is generated by the pair of matrices (J, U_{LP}) given by (3). The corresponding sequence is generated by the pair (L_{LP}, PJ), where $L_{\mathrm{LP}} = L$ is the lower-unitriangular matrix with the entries

$$L_{11} = 1, L_{i1} = L_{1j} = 0, L_{ij} = \binom{i-2}{j-2} \mod 2 \quad \text{for } 2 \le i, j \le m. \tag{11}$$

Indeed, the pair (L_{LP}, PJ) is progressive by Theorem 3.3 because $PJ = JPJ \cdot P =: L_y \cdot P$. The pair (L_{LP}, PJ) generates the same net because the characteristic matrices are the same: $PJL_{\mathrm{LP}}^{-1} = U_{\mathrm{LP}}J$ if m is a power of 2. This identity is proved at the end of Appendix B. For example, the following pair of matrices creates a 256-point LP net as a digital dyadic sequence:

$$\left(\begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0\\ 0&1&0&1&0&0&0&0\\ 0&1&1&1&1&0&0&0\\ 0&1&0&0&0&1&0&0\\ 0&1&1&0&0&1&1&0\\ 0&1&0&1&0&1&0&1 \end{pmatrix}, \begin{pmatrix} 1&1&1&1&1&1&1&1\\ 1&0&1&0&1&0&1&0\\ 1&1&0&0&1&1&0&0\\ 1&0&0&0&1&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&0&1&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0 \end{pmatrix}\right). \tag{12}$$

See Fig. 5(b). The analysis of the star discrepancy of all possible sub-sequences is an interesting open question.

4.4 Converting Point Sets to Digital Dyadic Sequences

Algorithm 2 can be used to test if an arbitrary set of points (including sets of points that are known dyadic nets) allows for a digital construction. If it is possible, the algorithm produces the pair of matrices that generate the given set of points as a digital dyadic sequence.

4.5 Gray Sequences

We applied the conversion algorithm above to the Gray Code family of nets introduced by Ahmed and Wonka [2021], obtained by using a Gray Code-ordered van der Corput sequence to supplement the x-offsets of the points along the columns and the y-offsets along the rows. These nets turned out to be digital.

Algorithm 2: Converting a point set into a digital dyadic sequence

```
Input: an arbitrary $2^m$-point set in $[0, 1)^2$;
Output: a pair of matrices that creates a digital dyadic
        sequence with the same points using (1), if it exists.
1 Order the points sequentially along the $x$-axis and set $C_x = J$;
2 Use the $y$-coordinates of the points at power-of-two indices
    as columns of the prospected $C_y$;
3 Conduct one pass through the other points to validate if they
    coincide with the ones given by (1), and check if $C_y$ is
    progressive;
4 If this step works, we have a digital dyadic net that can
    subsequently be converted to a sequence using Algorithm 1.
```

A 256-point Gray net (from now on, we will drop the word "Code") is generated by the pair of matrices (J, C'), where

$$C' = \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 1&1&1&0&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ 0&0&0&0&1&1&0&0\\ 0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&1 \end{pmatrix}. \tag{13}$$

The corresponding sequence is generated by

$$\left(\begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 1&1&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 1&0&0&0&1&0&0&0\\ 1&1&0&0&1&1&0&0\\ 1&0&1&0&1&0&1&0\\ 1&1&1&1&1&1&1&1 \end{pmatrix}, \begin{pmatrix} 1&1&1&1&1&1&1&1\\ 0&1&0&1&0&1&0&1\\ 1&0&0&1&1&0&0&1\\ 0&0&0&1&0&0&0&1\\ 1&1&1&1&0&0&0&0\\ 1&0&1&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0 \end{pmatrix}\right). \tag{14}$$

See Fig. 5(c). Note that matrix (13) generating the Gray net has the same structure for all even m we tried, whereas matrices (14) look different for different sizes of the sequence.

4.6 Validation

Let us report the following experiments that clearly demonstrate the utility of sequences over nets for progressive sampling: We considered the 256-point LP net. First, we took a random ordering of the points and computed the star discrepancies of the first 16 points, the first 32 points, the first 48 points, etc. Then we took the sequence ordering given by our algorithm (see Table 2) and computed similar star discrepancies. Finally, we computed the ratios of the 16 numbers for the random ordering to the 16 numbers for the sequence ordering, as depicted in Table 3. We obtained that the first number has become 2.22 times smaller, the second one 2.25 times smaller, etc. Note that the original order would have worse results than the randomly reordered one.

Table 3. The ratio of the star discrepancy of the first N points of the 256-point LP net for the random ordering to the one for the sequence ordering

N	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256
Ratio	2.22	2.25	2.23	2.60	2.59	2.71	3.04	3.94	2.98	3.16	2.65	2.25	1.85	1.54	1.25	1.00

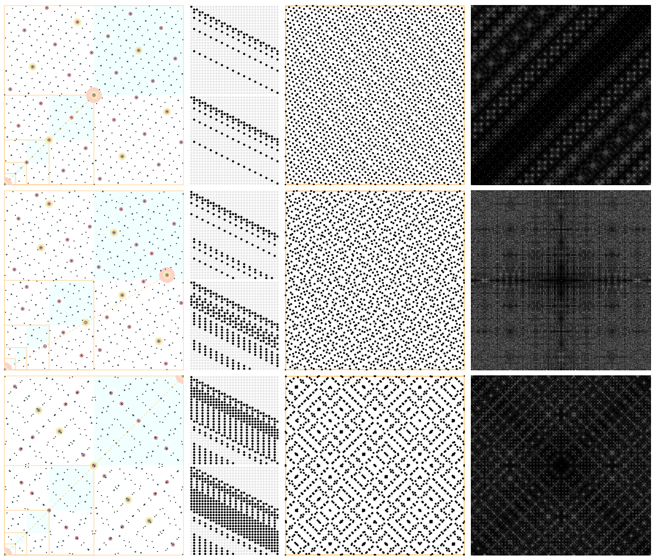


Fig. 6. Structure, generating matrices, points, and periodograms of various self-similar ξ-sequences. The plots to the left depict the first few points; larger circles show the points closer to the beginning of the sequence. The top plots show the ξ_0-sequence generated by $X = Y = (1, 0, \ldots, 0)$.

5 SELF-SIMILAR DYADIC SEQUENCES

In this section, we propose a family of digital dyadic sequences that is efficient to compute and invert. At the same time, the family is large enough to allow for a meaningful exploration of the design space in order to select high-quality representatives. Our construction is inspired by recursive tiling techniques for distributing blue-noise samples [Ahmed et al. 2017; Ostromoukhov 2007; Wachtel et al. 2014].

We start with the geometric design requirements. After extensive experiments in trying to design a self-similar sequence, we settled for the following idea. We are looking for an infinite dyadic sequence such that every fourth point belongs to the bottom-left quadrant $[0, 1/2)^2$ and for each $k = 0, 1, 2, \ldots$ the first 2^k points appearing in this quadrant form a scaled version of the first 2^k points of the whole sequence in $[0, 1)^2$, with the same order. See Fig. 6 to the top-left. One can formulate this design requirement as

$$p_{4i} = p_i/2 \qquad \text{for } i = 0, 1, 2, \ldots . \tag{15}$$

This means that a point with four times the sequence number is scaled by the factor 1/2. In practice, we restrict ourselves to finite digital dyadic sequences. A digital dyadic sequence $p_0, p_1, \ldots, p_{2^m-1}$ is called *self-similar* if for each $i < 2^{m-2}$ we have (15) with the right side truncated to m digits (i.e., if the $(m+1)$-th digit after the point in the dyadic decomposition of one of the coordinates of $p_i/2$ is 1 then it is replaced by 0). This simple equation is all that we need to realize the sequence digitally.

We break it down into three elements. First, we note that multiplying the sequence number by 4 means shifting the bit vector S down two steps, inserting two leading zeros that effectively wipe out two columns of the progressive matrix C_x. Second, (15) means that the subsequent columns of the matrix should bear similar information to the wiped ones. Finally, dividing the coordinates by 2 is equivalent to pushing all these subsequent columns one row down, inserting zeros. This leads to the following self-similar matrix structure

$$C_x = \begin{pmatrix} a_0 & b_0 & 0 & 0 & 0 & \cdots \\ a_1 & b_1 & a_0 & b_0 & 0 & \cdots \\ a_2 & b_2 & a_1 & b_1 & a_0 & \cdots \\ a_3 & b_3 & a_2 & b_2 & a_1 & \cdots \\ a_4 & b_4 & a_3 & b_3 & a_2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \tag{16}$$

for some column bit vectors $A = (a_0, a_1, \cdots)$ and $B = (b_0, b_1, \cdots)$ that, by definition, represent the x-coordinates of the second and third point in the sequence (equal to $0.a_0a_1\ldots$ and $0.b_0b_1\ldots$ respectively). We take A as a design parameter that can be freely chosen by the user, except for a_0 which has to be 1. It then remains to set or compute the bits of B so as to ensure a progressive matrix. We can iteratively expand the matrix. The key insight is that each subsequent expansion includes only one new unknown bit from B, except for the first step where we have two bits when expanding from a 1×1 to a 2×2 matrix. Formally, we have the following statement:

Lemma 5.1. *For each bit vector $A = (1, a_1, a_2, \cdots, a_{m-1})$ starting with bit 1 there are exactly two bit vectors $B = (b_0, b_1, \cdots, b_{m-1})$ such that matrix (16) is progressive.*

Proof. Take an arbitrary bit b_0 and let us determine bits $b_1, b_2, \ldots$ inductively.

Assume that $b_0, \ldots, b_{k-1}$ have already been determined so that the first k leading principal minors of the matrix in Eq. (16) equal 1. Let us show that then there is a unique b_k such that the $(k+1)$-th leading principal minor Δ equals 1; for clarity, we illustrate the argument for $k = 3$. Take the Laplace expansion along the last row:

$$\Delta = \begin{vmatrix} 1 & b_0 & 0 & 0 \\ a_1 & b_1 & 1 & b_0 \\ a_2 & b_2 & a_1 & b_1 \\ a_3 & b_3 & a_2 & b_2 \end{vmatrix} = a_3 \begin{vmatrix} b_0 & 0 & 0 \\ b_1 & 1 & b_0 \\ b_2 & a_1 & b_1 \end{vmatrix} +$$

$$+ b_3 \begin{vmatrix} 1 & 0 & 0 \\ a_1 & 1 & b_0 \\ a_2 & a_1 & b_1 \end{vmatrix} + a_2 \begin{vmatrix} 1 & b_0 & 0 \\ a_1 & b_1 & b_0 \\ a_2 & b_2 & b_1 \end{vmatrix} + b_2 \begin{vmatrix} 1 & b_0 & 0 \\ a_1 & b_1 & 1 \\ a_2 & b_2 & a_1 \end{vmatrix}.$$

By the structure of the matrix in Eq. (16), the coefficient at b_k here

$$\begin{vmatrix} 1 & 0 & 0 \\ a_1 & 1 & b_0 \\ a_2 & a_1 & b_1 \end{vmatrix} = \begin{vmatrix} 1 & b_0 \\ a_1 & b_1 \end{vmatrix}$$

is exactly the $(k-1) \times (k-1)$ leading principal minor, equal to 1 by the inductive hypothesis. Thus there is a unique b_k such that $\Delta = 1$. Note that we compute the determinant of a $(k+1) \times (k+1)$ matrix using the assumption for the $(k-1) \times (k-1)$ matrix. Depending on k being even or odd the matrix structure is slightly different, and we only illustrate the case for k being odd here, but the even case is analogous. □

Similarly to the case of arbitrary progressive matrices in Section 3, there is a closed-form solution for the self-similar case. Starting with the simpler case $A = (1, 0, 0, \dots)$, i.e. setting the x-coordinate of the second point in the sequence to be $1/2 = 0.100\dots$, we get the following assertion: if $A = (1, 0, 0, \dots)$ in progressive matrix (16), then B is one of the vectors

$$(0, \xi_1, \xi_2, \dots) = (0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, \dots), \tag{17}$$

$$(1, \xi_1, \xi_2, \dots) = (1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, \dots), \tag{18}$$

where $\xi_i = 1$ if and only if i is a power of 2. In other words, the third point of the sequence has one of the (truncated) x-coordinates

$$\xi = 0.01101000100000001\dots = \frac{1}{2} \sum_{k=0}^{\infty} 2^{-2^k}, \tag{19}$$

$$\xi^+ = 0.11101000100000001\dots = \frac{1}{2} + \frac{1}{2} \sum_{k=0}^{\infty} 2^{-2^k}. \tag{20}$$

This is proved by induction similarly to Lemma 5.1. This gives us the following self-similar progressive matrix:

$$C_\xi = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & \cdots \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & \cdots \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & \cdots \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}, \tag{21}$$

and a similarly looking matrix for C_{ξ^+}. Note that this matrix looks quite different from the conventional upper-triangular form of Sobol matrices. Note also that ξ appears also along the diagonal.

Now we move to the general case, where we have

Lemma 5.2. *For arbitrary $A = (1, a_1, a_2, \dots)$ the two vectors*

$$B = \xi(A) := (0,\ \xi_1,\ a_1\xi_1 + \xi_2,\ a_2\xi_1 + a_1\xi_2 + \xi_3,\ \dots) \tag{22}$$

$$B^+ = \xi^+(A) := \xi(A) + A \tag{23}$$

are all the vectors making matrix (16) progressive.

In other words, the x-coordinate of the third point in the sequence is obtained by the carryless multiplication of the x-coordinate of the second point by one of numbers (19)–(20), followed by usual multiplication by 2.

Proof. Multiply C_ξ (or C_{ξ^+}) by the lower unitriangular matrix

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots \\ a_1 & 1 & 0 & \cdots \\ a_2 & a_1 & 1 & \cdots \\ a_3 & a_2 & a_1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{24}$$

from the left. This keeps the matrix progressive (LU-decomposable), preserves structure (16), and places A and $\xi(A)$ (or $\xi^+(A)$) in the first two columns. Then the proof is concluded by Lemma 5.1. □

Denoting the corresponding matrix by $C_{\xi(A)} := LC_\xi$ and $C_{\xi^+(A)} := LC_{\xi^+}$, we have the following statement to put it together:

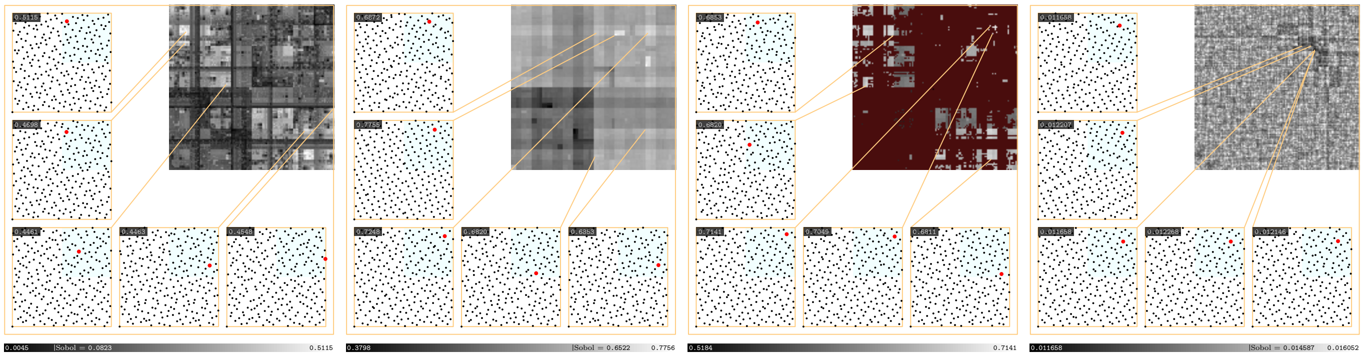
Theorem 5.3. *For any pair of bit vectors X and Y starting from 1 in the first position, the pairs $(C_{\xi(X)}, C_{\xi^+(Y)})$ and $(C_{\xi^+(X)}, C_{\xi(Y)})$ are progressive. Conversely, each progressive pair of matrices of form (16) equals either $(C_{\xi(X)}, C_{\xi^+(Y)})$ or $(C_{\xi^+(X)}, C_{\xi(Y)})$ for some X and Y starting from 1.*

Proof. By construction, the matrices $C_{\xi(X)}, C_{\xi^+(Y)}$ have form (16) with $(a_0, b_0) = (1, 0)$ and $(1, 1)$ respectively; i.e., our pair has form

$$\left(\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots \\ a_1 & b_1 & 1 & 0 & \cdots \\ a_2 & b_2 & a_1 & b_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 & 0 & \cdots \\ c_1 & d_1 & 1 & 1 & \cdots \\ c_2 & d_2 & c_1 & d_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \right)$$

for some a_i, b_i, c_i, d_i. The two matrices are themselves progressive by Lemma 5.2. It remains to prove that hybrid matrix (6) has determinant 1 for $r = 1, \dots, k-1$. We compute the determinant recursively. Since the first row of (6) is $(1\,0\,\dots\,0)$ in our case, it follows that removing the first row and the first column preserves the determinant:

$$\begin{vmatrix} 1 & 0 & 0 & 0 & \cdots \\ a_1 & b_1 & 1 & 0 & \cdots \\ a_2 & b_2 & a_1 & b_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & 1 & 0 & 0 & \cdots \\ c_1 & d_1 & 1 & 1 & \cdots \\ c_2 & d_2 & c_1 & d_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{vmatrix} = \begin{vmatrix} b_1 & 1 & 0 & \cdots \\ b_2 & a_1 & b_1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ 1 & 0 & 0 & \cdots \\ d_1 & 1 & 1 & \cdots \\ d_2 & c_1 & d_1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{vmatrix}.$$



(a) Minimum distance between points (b) Average distance to the closest neighbor (c) The same for the minimum distance > 0.3 (d) Star discrepancy

Fig. 7. An exhaustive exploration of ξ-sequences containing 256 points. Each pixel in the top-right square $[1/2; 1)^2$ represents a possible position of the second point of the sequence, which uniquely determines the whole ξ-sequence. There are 128×128 pixels in total and 16K possible ξ-sequences. The color depicts one of the (suitably normalized) quality measures (a)–(d) of the resulting ξ-sequence. In (c), the average distance to the closest neighbor is shown only for the sequences such that the minimum distance is greater than 0.3; the remaining pixels are brown. A few best-quality representatives are in the insets; each red bold point depicts the second point in the sequence. For comparison, the same quality measures for the Sobol sequence are shown in the legends.

In the resulting matrix, the $(k - r)$-th row (highlighted) becomes $(1\,0\,\ldots\,0)$. Now removing this row and the first column, we get

$$\begin{vmatrix} b_1 & 1 & 0 & \cdots \\ b_2 & a_1 & b_1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ 1 & 0 & 0 & \cdots \\ d_1 & 1 & 1 & \cdots \\ d_2 & c_1 & d_1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{vmatrix} = \begin{vmatrix} 1 & 0 & \cdots \\ a_1 & b_1 & \cdots \\ \vdots & \vdots & \ddots \\ 1 & 1 & \cdots \\ c_1 & d_1 & \cdots \\ \vdots & \vdots & \ddots \end{vmatrix}.$$

The resulting matrix has the same structure as initial matrix (6), only r and k are reduced by 1 and 2 respectively. We now repeat the procedure, each time removing the row $(1\,0\,\ldots\,0)$ and the first column in the current matrix until we get $r = 0$ or $r = k$. The resulting determinant will be a leading principal minor of $C_{\xi(X)}$ or $C_{\xi^+(Y)}$. It equals 1 because $C_{\xi(X)}$ or $C_{\xi^+(Y)}$ are themselves progressive.

The converse assertion follows from Lemma 5.2 because the pairs $(C_{\xi(X)}, C_{\xi(Y)})$ and $(C_{\xi^+(X)}, C_{\xi^+(Y)})$ are never progressive. □

We find these results striking, since this constant ξ handles all the effort of matrix derivation and pairing we encountered in Section 3. Thus, the resulting self-similar sequences generated by $(C_{\xi(X)}, C_{\xi^+(Y)})$ for various X and Y are deservedly called ξ-*sequences*. See Fig. 6 and 7. (We do not need to consider the pairs $(C_{\xi^+(X)}, C_{\xi(Y)})$ because this leads just to the swap of x- and y-coordinates.) The sequence generated by (C_ξ, C_{ξ^+}) is referred as ξ_0-sequence. On the practical side, we truncate the coordinates of the 2^m points not to m bits as required by the digital construction, but to the machine precision of $M = 32$ (or $M = 64$) bits. In other words, we always consider the first 2^m points of a 2^M-point ξ-sequence. We note that the special structure of ξ reduces the carryless multiplication to $O(\log M)$ from $O(M)$. In 32-bit precision, it takes precisely 5 shift-and-xor operations. This makes setting up a ξ-sequence, given (X, Y), incredibly low-cost, even cheaper than generating a random sample for (X, Y). Algorithm 3 summarizes the preceding derivations into a set of steps for creating a ξ-sequence.

Algorithm 3: Constructing the first 4 points of a ξ-sequence using 32-bit precision from two bit-vectors $X = (x_0, \ldots)$ and $Y = (y_0, \ldots)$ with $x_0 = y_0 = 1$. All additions and multiplications are carryless.

```
Input  : Two bit-vectors $X$ and $Y$ (the coordinates of $p_1$);
Output : The first 4 points $p_0, \ldots, p_3$ of the sequence.
1 $\xi(A) \leftarrow (2^{-1} + 2^{-2} + 2^{-4} + 2^{-8} + 2^{-16})A$;
2 $B \leftarrow \xi(X)$;
3 $B^+ \leftarrow \xi(Y) + Y$;
4 $p_0 \leftarrow (0, 0)$;
5 $p_1 \leftarrow (X, Y)$;
6 $p_2 \leftarrow (B, B^+)$;
7 $p_3 \leftarrow (X, Y) + (B, B^+)$.
```

Algorithm 4: C code for retrieving a sample from a ξ-sequence. The sequence is encoded by the first 4 points $p[0], \ldots, p[3]$. The input is the number seqNo of the sample.

```
Point getSample(uint32_t seqNo) {
    uint32_t x(0), y(0);
    for (int depth = 0; depth < 16; depth++) {
        x ^= p[seqNo & 3].x >> depth;
        y ^= p[seqNo & 3].y >> depth;
        seqNo >>= 2;
    }
    return {x, y};
}
```

To our knowledge, ξ-sequences are the fastest way to construct a dyadic sequence and draw different dyadic nets. They enable constructing a pseudo-random sequence for each pixel. However, the fast construction turns out to be not the only advantage: the self-similar structure actually embodies a wealth of versatility that we are going to discuss in the following subsections.

Table 4. Speed performance, in million points per second, of ξ-sequences and other state-of-the-art sampler generators.

$m = \log_2(N)$	8	12	16	20	24	28
Random	18	25	24	56	68	71
Sobol	32	35	28	53	57	51
Burley	11	16	13	31	39	39
ξ	17	26	24	57	74	73
ξ_{256}	32	113	146	185	411	410

5.1 Retrieving the Samples

Points from ξ-sequences may be generated from the matrices as any standard digital sequence. The self-similar structure of the matrices, however, offers a short-hand, as outlined in Algorithm 4.

Thus, besides fast construction, drawing the samples is an aspect where ξ-sequences significantly stand out compared to all alternatives we are aware of. The simple loop structure of ξ-sequences, and the few parameters, make it a good candidate for low-level optimization, using registers, for example, but our plain C code already delivers high performance, as depicted in Table 4.

While the straightforward implementation of ξ-sequences shown in Algorithm 4 already leads the competition, the self-similar structure of the sequence permits ever further speed up through a small lookup table. In the basic implementation we derive the basis vectors from the first four points, but in fact any power-of-four number of points may be used in the same way as basis for retrieving subsequent points, as may be found in our supplementary code. The ξ_{256} entry in Table 4 refers to an implementation that uses a 256-vector look-up table, and it leads to 2-6× speed up.

5.2 Memory Footprint

In addition to their speed performance, ξ-sequences consume very little memory, and offer a flexible trade-off between memory and speed. The sequence may be compressed to a single 2D bit vector p_1, a pair of vectors p_1, p_2, or the standard four vectors $p_0, \ldots, p_3$, and may be expanded to a 16, 256, or even 64K lookup table for two-step retrieval. This flexibility with memory makes ξ-sequences quite GPU friendly. Indeed, providing random numbers in a GPU context is a standing challenge, and ξ-sequences help in this direction by making it possible to run a unique random sequence in each thread, passing a table of parameters that identify the sequences. We implemented a test of this, and generated over 17G points per second, roughly 1T per minute, on a TITAN Xp GPU.

5.3 Morton Ordering

Instead of generating the x and y coordinates separately, they can be obtained jointly as a Morton-ordering [Morton 1966] index

$$\mathbf{z} = 0.y_0 x_0 y_1 x_1 \cdots , \tag{25}$$

and then we can split (*unshuffle* [Warren 2012]) the bits later. This can be done easily by

$$\mathbf{z}^{(i+1)} \leftarrow \mathbf{z}^{(i)} + p_z[s^{(i)} \bmod 4] \cdot 2^{-2i}, \quad \mathbf{s}^{(i+1)} \leftarrow \lfloor \mathbf{s}^{(i)}/4 \rfloor \tag{26}$$

where $i = 0, \ldots, 15$, $\mathbf{z}^{(0)} = 0$, $\mathbf{s}^{(0)}$ is the point index in the sequence, $p_z[q]$ is obtained by interleaving (*shuffling* [Warren 2012]) the bits of y- and x-coordinates of the q-th point $p[q]$ for $q = 0, \ldots, 3$, and addition is carryless. This may not be very useful in generating the samples, because, in addition to the cost of unshuffle, it requires 64-bit processing to obtain the full 32-bit resolution of each axis. Mapping to Morton indices, however, is useful in the reverse direction, where we are given the coordinates of a stratum, and asked to retrieve the exact location of the (first) sample in that stratum, and/or its sequence number, as encountered in stippling and importance sampling. Note that the coordinates of the stratum prefix the coordinates of points inside it. Thus, obtaining a point sequence number, given the stratum, is a partial inversion of the sequence, which we discuss in more detail next.

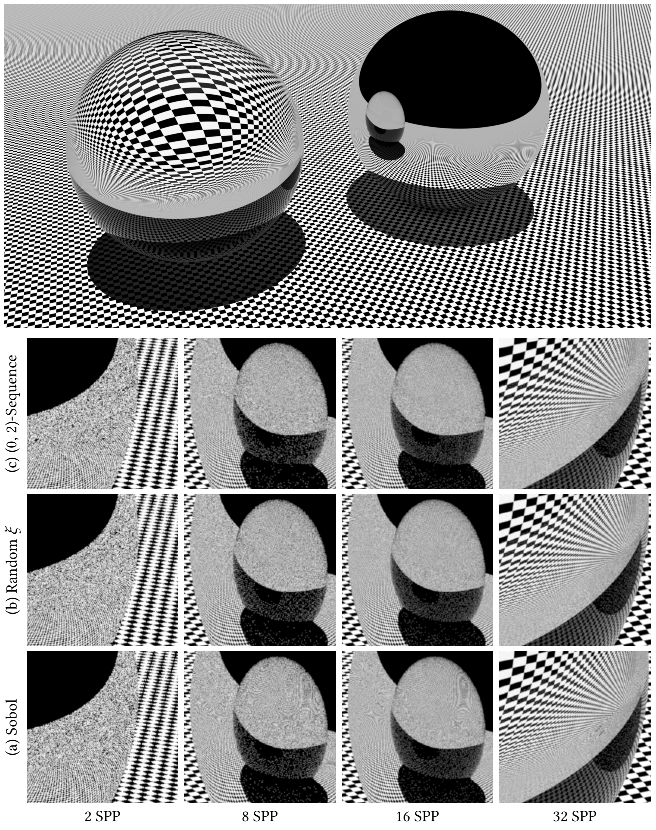


Fig. 8. Physically-based renderings using various digital dyadic sequences: (a) The Global Sobol sampler based on the Sobol sequence; (b) our adapted global sampler using a random ξ-sequence; and (c) low-discrepancy $(0, 2)$-sequence sampler, which uses independently shuffled Sobol 2D samples for each pixel and pair of dimensions. We placed our sampler in the middle for easier comparison. The figure shows that the ξ-based Global sampler is less aliased than Sobol and less noisy than the pixel-based sampler. The insets show selected parts of the scene at different sampling rates.

5.4 Inverting the Sequence

A straightforward approach to inverting a ξ-sequence, mapping a stratum index $\mathbf{z}$ into a sequence number $\mathbf{s}$, is to *undo* the iterations

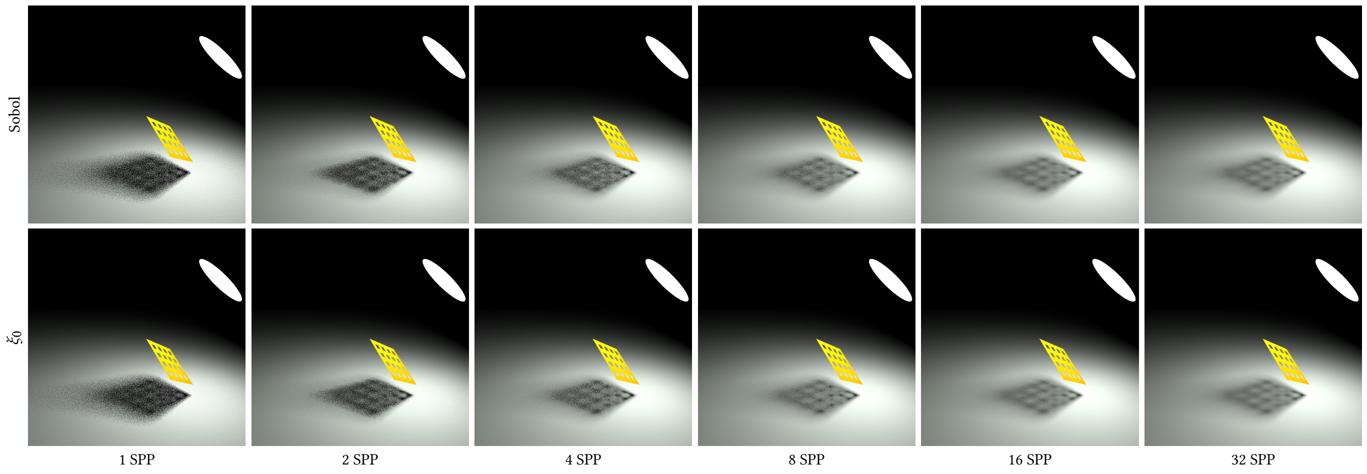


Fig. 9. Renderings obtained with the Z-sampler from Ahmed and Wonka [2020] using Sobol and ξ_0 sequences for the generation of the samples at difference sampling rates (samples per pixel). The figure shows that the ξ-based Z-Sampler is less noisy than the Sobol-based one for the shown scene.

in Eq. (26). Let

$$\mathbf{s} = \cdots s_{54}s_{32}s_{10}\,, \tag{27}$$

$$\mathbf{z} = 0.z_{01}z_{23}\cdots\,, \tag{28}$$

$$p_z[q] = 0.p_{z01}[q]p_{z23}[q]\cdots \tag{29}$$

encode the sequence index of a point, its Morton index, and the first 4 points of the sequence written in base 4. Note the reversed indexing of the digits of the sequence number. Given $\mathbf{z}^{(0)} = \mathbf{z}$, we may find $\mathbf{s}$ iteratively starting from $\mathbf{s}^{(0)} = 0$ as follows:

$$q \leftarrow q : z_{01}^{(i)} = p_{z01}[q]\,, \tag{30}$$

$$\mathbf{s}^{(i+1)} \leftarrow \mathbf{s}^{(i)} + q \cdot 2^{2i}\,, \tag{31}$$

$$\mathbf{z}^{(i+1)} \leftarrow (\mathbf{z}^{(i)} + p_z[q]) \cdot 4\,, \tag{32}$$

in the i-th iteration. The idea is that, if we examine Eq. (26), we see that only one vector contributes to the most significant two bits of $\mathbf{z}$. We can therefore deduce which of the four vectors was added in the first iteration of computing $\mathbf{z}$, and insert its index as a digit in the sequence number, as in Eq. (31). We then subtract, or equivalently add, the originally added vector to $\mathbf{z}$. Now, only one vector is responsible for the following pair of bits. By shifting the bits up we can recursively apply the undo process. Note that this process iteratively zeros $\mathbf{z}$ while building $\mathbf{s}$.

We implemented this and obtained the inversion rate of ~38M points per second (cf. Table 4) that, while faster than any sampling technique we are aware of, is significantly slower than the forward production rate. A much faster approach takes advantage of Morton ordering discussed in Section 5.3, noting that the inversion matrix bears a similar staggered pair-of-columns structure. This gives a speed up of up to 70% over the forward rate, since only one value is evaluated rather than two. Implementation details may be found in the supplementary code and interactive demos.

5.5 Coding Complexity

Again, this is an aspect where ξ-sequences have an advantage. Not only the fact that their implementation follows straightforward steps, as listed in Algorithm 3, but the intuition of these steps has a clear geometric interpretation, as we discussed earlier. In the supplementary materials we provide a JavaScript-based interactive demonstration of these sequences that we encourage the reader to experiment with.

5.6 Atlas

The relatively small two-dimensional space of ξ-sequences, along with their easy implementation on GPUs, makes it possible to scan them exhaustively and build an atlas for exploring these patterns and finding ones with favorable properties. Fig. 7 shows example maps for the star discrepancy, minimum distance between points, and average distance to the nearest neighbor.

5.7 Rendering Examples

We show the impact of our work in two rendering examples.

The first example (Fig. 8) demonstrates that self-similar sequences provide better anti-aliasing than the standard Sobol sequence. To make a meaningful comparison we integrated the sequences into the Global Sobol Sampler coming with PBRT. The ξ-sequences offer: (1) 3× speedup in sample generation and 5× in inversion, leading to up to 37% observed reduction in rendering times; (2) no inversion lookup tables, cf.[Grünschloß et al. 2012]; (3) notable anti-aliasing with randomized ξ-sequences, and (4) they are free; emphasizing the value of our in-depth study.

Let us make a few comments on the implementation. The sampler uses a *higher-dimensional* Sobol sequence, generated by certain matrices $(I, P, C_2, C_3, \ldots)$; the first two are used to sample the image plane. We transform the matrices so that the first two become generating matrices $(C_{\xi(X)}, C_{\xi^+(Y)})$ of the desired ξ-sequence. For that purpose, we decompose $C_{\xi(X)} = L_xU$ and $C_{\xi^+(Y)} = L_yPU$ as in Theorem 3.3. Then we right-multiply all the original matrices $(I, P, C_2, C_3, \ldots)$ by U (which means just reordering of the sequence) and left-multiply the first two matrices by L_x and L_y respectively (which means scrambling). The resulting sequence is used for sampling.

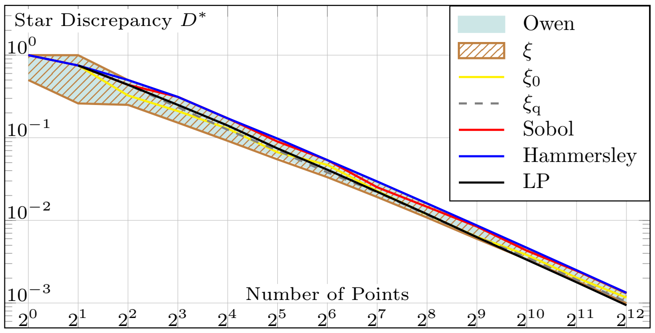


Fig. 10. Discrepancy comparison of XOR-scrambled ξ-sequences to unscrambled Sobol sequence. Hammersley and Larcher-Pillichshammer (LP) nets are shown for comparison. For Owen and ξ we measured the discrepancy over a randomly drawn 64K sample. The yellow line refers to the ξ_0-sequence. The dashed line ξ_q refers to the lowest measured discrepancy of the sampled sequences when the coordinates of the points are truncated to $\log_2$(Number of Points) bits.

The second example (Fig. 9) also shows slightly better anti-aliasing. Here we integrated the sequences into the Z-Sampler [Ahmed and Wonka 2020] that uses the samples as is, without any scrambling. Full-resolution images may be found in the supplementary materials.

5.8 Discrepancy and Spectral Evaluation

The new class of self-similar sequences introduced in the paper has several advantages, that we have evaluated in the preceding subsections, such as speed, memory usage, anti-aliasing, and versatility. Towards that, we show discrepancy and spectral comparison in Fig. 10 and 11, both demonstrating that ξ-sequences are a good representative sample of the space of digital dyadic sequences. Note that all digital dyadic sequences exhibit some structure when looking at a single instance and the perceived structure changes with m (see Fig. 1).

6 DISCUSSION AND CONCLUSION

In this paper, we presented new insights on digital dyadic sequences. Our work provides a comprehensive understanding of the design space and how to navigate it.

We believe our work will lay the foundation for future work in systematically exploring the design space of higher-dimensional sequences and their impact on rendering. The importance of 2D sequences is that they often serve as building blocks for synthesizing high-dimensional sequences. Throughout the paper, we already discussed multiple implications of our work on rendering applications, and we are hopeful that the proposed ξ-sequences and Gray sequences will find their way into existing rendering frameworks.

ACKNOWLEDGMENTS

We are grateful to F. Pillichshammer for bringing earlier proofs of Theorems 3.1 and 3.3 (which we conceived independently) to our attention. Thanks to Mohanad Ahmed for his insightful discussions.

REFERENCES

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

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

Abdalla G. M. Ahmed, Jing Ren, and Peter Wonka. 2022. Gaussian Blue Noise. *ACM Trans. Graph.* 41, 6, Article 260 (nov 2022), 15 pages. https://doi.org/10.1145/3550454.3555519

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

Gregory V. Bard. 2009. *Some Basic Facts about Linear Algebra over GF(2).* Springer US, Boston, MA, 81–88. https://doi.org/10.1007/978-0-387-88757-9_6

Paul Bratley and Bennett L. Fox. 1988. Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator. *ACM Trans. Math. Softw.* 14, 1 (mar 1988), 88–100. https://doi.org/10.1145/42288.214372

Brent Burley. 2020. Practical Hash-based Owen Scrambling. *Journal of Computer Graphics Techniques (JCGT)* 10, 4 (29 December 2020), 1–20. http://jcgt.org/published/0009/04/01/

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

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms* (2nd ed.). The MIT Press, Cambridge, Massachusetts.

Josef Dick and Friedrich Pillichshammer. 2010. *Digital Nets and Sequences: Discrepancy Theory and Quasi–Monte Carlo Integration.* Cambridge University Press. https://doi.org/10.1017/CBO9780511761188

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*, Kai-Tai Fang, Harald Niederreiter, and Fred J. Hickernell (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 242–256.

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

Ira Gessel and Gérard Viennot. 1985. Binomial Determinants, Paths, and Hook Length Formulae. *Advances in Mathematics* 58, 3 (1985), 300–321. https://doi.org/10.1016/0001-8708(85)90121-5

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. 1989. *Concrete Mathematics: A Foundation for Computer Science.* Addison-Wesley, Reading.

Leonhard Grünschloß, Johannes Hanika, Ronnie Schwede, and Alexander Keller. 2008. (t, m, s)-Nets and Maximized Minimum Distance. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, Alexander Keller, Stefan Heinrich, and Harald Niederreiter (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 397–412.

Leonhard Grünschloß and Alexander Keller. 2009. (t, m, s)-Nets and Maximized Minimum Distance, Part II. In *Monte Carlo and Quasi-Monte Carlo Methods 2008.* Springer, 395–409.

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

J. M. Hammersley. 1960. Monte Carlo Methods for Solving Multivariable Problems. *Annals of the New York Academy of Sciences* 86, 3 (1960), 844–874. https://doi.org/10.1111/j.1749-6632.1960.tb42846.x

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

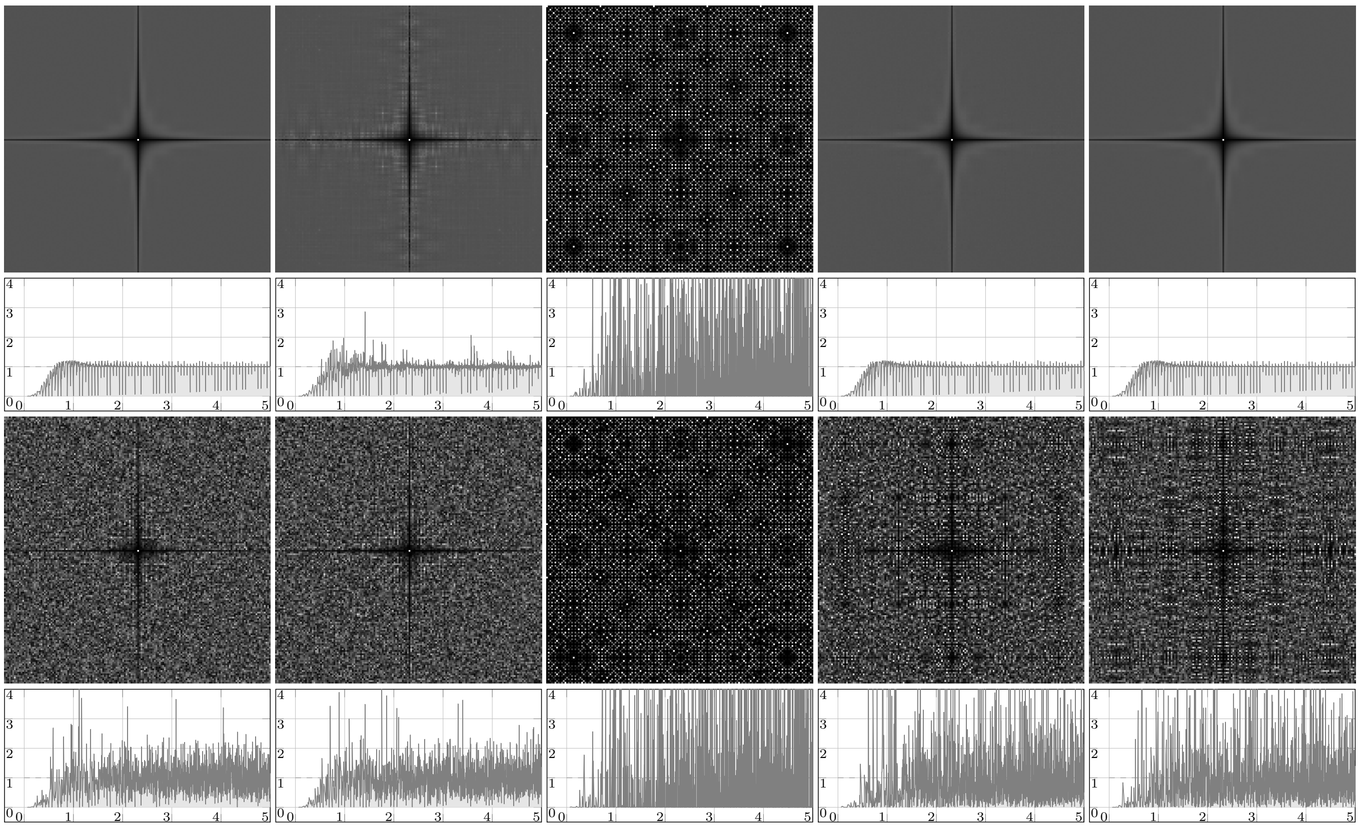
Roswitha Hofer and Gerhard Larcher. 2010. On existence and discrepancy of certain digital Niederreiter–Halton sequences. *Acta Arithmetica* 141 (2010), 369–394.

Roswitha Hofer and Kosuke Suzuki. 2019. A Complete Classification of Digital (0, 3)-Nets and Digital (0, 2)-Sequences in Base 2. *Uniform distribution theory* 14, 1 (2019), 43–52. https://doi.org/doi:10.2478/udt-2019-0004

Roger A. Horn and Charles R. Johnson. 1985. *Matrix Analysis.* Cambridge University Press, Cambridge.

Hiroki Kajiura, Makoto Matsumoto, and Kosuke Suzuki. 2018. Characterization of matrices B such that (I, B, B2) generates a digital net with t-value zero. *Finite Fields and Their Applications* 52 (2018), 289–300.

Alexander Keller. 2006. Myths of Computer Graphics. In *Monte Carlo and Quasi-Monte Carlo Methods 2004.* Springer, 217–243.



(a) Owen (b) Burley (c) XOR-Scrambled Sobol (d) Random Digital (e) ξ

Fig. 11. Two-dimensional and radial average spectral profiles of different dyadic sequences, showing (top) the frequency power spectrum of the point process, obtained by averaging 16k periodograms of 256 points, and (bottom) a periodogram of a randomly selected realization. We use radial profiles advocated by [Ahmed et al. 2022], where we average values for every distinct radius. The set for ξ-sequences includes all distinct sequences (up to XOR-scrambling) at 8-bit resolution. A randomly selected digital or ξ-sequence has a smoother periodogram than the XOR-Scrambled Sobol sequence but not Burley's implementation of Owen-scrambling. However, the average of sufficiently many periodograms for ξ-sequence is smoother than both, and visually as smooth as Owen's scrambling.

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

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

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang Tiles for Real-Time Blue Noise. In *ACM SIGGRAPH 2006 Papers* (Boston, Massachusetts) *(SIGGRAPH '06)*. Association for Computing Machinery, New York, NY, USA, 509–518. https://doi.org/10.1145/1179352.1141916

Gerhard Larcher and Friedrich Pillichshammer. 2001. Walsh Series Analysis of the L_2-Discrepancy of Symmetrisized Point Sets. *Monatshefte für Mathematik* 132, 1 (2001), 1–18.

G Larcher and F Pillichshammer. 2003. Sums of Distances to the Nearest Integer and the Discrepancy of Digital Nets. *Acta Arithmetica* 106 (2003), 379–408.

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

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

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

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

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

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

K. F. Roth. 1954. On Irregularities of Distribution. *Mathematika* 1, 2 (1954), 73–79. https://doi.org/10.1112/S0025579300000541

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

Gurprit Singh, Cengiz Öztireli, Abdalla G.M. Ahmed, David Coeurjolly, Kartic Subr, Oliver Deussen, Victor Ostromoukhov, Ravi Ramamoorthi, and Wojciech Jarosz. 2019. Analysis of Sample Correlations for Monte Carlo Rendering. *Computer Graphics Forum* 38, 2 (2019), 473–491. https://doi.org/10.1111/cgf.13653

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.

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

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

A PROOF OF THEOREM 3.1 (ON THE DYADIC PAIRS)

We present the known proof in detail to provide insight into the more complicated argument in the next appendix. We use two natural properties of dyadic pairs stated as lemmas below.

Lemma A.1. *If a pair of matrices (C_x, C_y) is dyadic and M is any invertible matrix, then (C_xM, C_yM) is also dyadic.*

Proof. Recall that the right multiplication by an invertible binary matrix is equivalent to a sequence of *elementary transformations*, each being a swap of two columns or adding a column to another one. The transformation $(C_x, C_y) \mapsto (C_xM, C_yM)$ is equivalent to applying those simultaneously to the columns of C_x and C_y. This leads to elementary transformations of any hybrid matrix (4) formed by the first $m-r$ rows of C_x and the first r rows of C_y. Since elementary transformations preserve the determinant, the hybrid matrix remains invertible, hence the pair remains dyadic. □

Applying this lemma for $M = C_x^{-1}$, we can make the first matrix in the pair the identity matrix. In the latter case, we have the following characterization of dyadic pairs [Kajiura et al. 2018, Lemma 3.1].

Lemma A.2. *A pair of matrices (I, C_y) is dyadic if and only if all leading principal minors of C_yJ are nonzero.*

Proof. Let $C_y = (b_{ij})$. The determinants of all the possible hybrid matrices (formed by r rows of I and $m-r$ rows of C_y)

$$\begin{vmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \\ b_{11} & \dots & b_{1r} & b_{1,r+1} & \dots & b_{1,m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_{m-r,1} & \dots & b_{m-r,r} & b_{m-r,r+1} & \dots & b_{m-r,m} \end{vmatrix} = \\ = \begin{vmatrix} b_{1,r+1} & \dots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{m-r,r+1} & \dots & b_{m-r,m} \end{vmatrix}$$

are exactly the leading principal minors of C_yJ, for $r \neq m$. Here we applied the Laplace expansion along the first r rows and used that C_yJ is obtained from C_y by reversing the order of the columns. □

Proof of Theorem 3.1. By Lemma A.1, (C_x, C_y) is dyadic if and only if $(I, C_yC_x^{-1})$ is dyadic. By Lemma A.2, the latter is dyadic if and only if $C_yC_x^{-1}J$ is progressive. This means that $C_yC_x^{-1}J = LU$ for some upper unitriangular matrix U and lower unitriangular matrix L. The latter is equivalent to $C_y = LUJC_x$ with C_x invertible. □

Algorithm 5: LU-decomposition [Cormen et al. 2001]

```
Input  : an $m \times m$ progressive matrix $A = (a_{ij})$;
Output : lower and upper unitriangular matrices $L = (l_{ij})$
         and $U = (u_{ij})$ such that $A = LU$;
1 for $k \leftarrow 1$ to $m$
2     do for $i \leftarrow k$ to $m$
3         do $l_{ik} \leftarrow a_{ik}$
4            $u_{ki} \leftarrow a_{ki}$
5     for $i \leftarrow k+1$ to $m$
6         do for $j \leftarrow k+1$ to $m$
7             do $a_{ij} \leftarrow a_{ij} - l_{ik}u_{kj}$
8 Return $L$ and $U$.
```

B PROOF OF THEOREM 3.3 (ON THE PROGRESSIVE PAIRS)

We give a new short elementary proof of Theorem 3.3. In contrast to the one by [Hofer and Suzuki 2019], it does not involve 3D nets and works entirely in 2D. It is based on properties of progressive pairs stated as lemmas below. Again, we start with the invariance under certain transformations, discovered by [Faure and Tezuka 2002].

Lemma B.1. *If a pair of matrices (C_x, C_y) is progressive, U is an upper unitriangular matrix, and L_x, L_y are lower unitriangular matrices, then (L_xC_xU, L_yC_yU) is also progressive.*

Proof. We use the same idea as in the proof of Lemma A.1. The right multiplication by an upper unitriangular matrix is equivalent to a sequence of elementary transformations, each being adding a column to another one located to the right from it. The transformation $(C_x, C_y) \mapsto (C_xU, C_yU)$ is equivalent to applying those simultaneously to the columns of C_x and C_y. Since a column is always added to a one to the right, this leads to elementary transformations of any hybrid matrix formed by the top-left corner $r \times k$ submatrix of C_x and the top-left corner $(k-r) \times k$ submatrix of C_y. Since elementary transformations preserve the determinant, the hybrid matrix remains invertible, hence the pair remains progressive.

Similarly, the left multiplication by a lower unitriangular matrix is equivalent to a sequence of elementary transformations, each being adding a row to another one located below it. The transformation $(C_x, C_y) \mapsto (L_xC_x, L_yC_y)$ now means *different* elementary transformations of C_x and C_y. But since a row is always added to a one below, this still leads to elementary transformations of any hybrid matrix. Hence the pair remains progressive. □

Applying this lemma, we can make the first matrix in the pair the identity matrix and the second one an upper unitriangular matrix. In the latter case, we have the following characterization of progressive pairs in terms of minors. A *shifted leading minor* is a minor formed by the first r rows and some r consecutive columns for some r.

Lemma B.2. *A pair of matrices (I, C_y) is progressive if and only if all shifted leading minors of C_y are nonzero.*

Proof. This is similar to Lemma A.2, but now the numbers k and r of rows taken from the matrices I and C_y need not sum up to m.

Let $C_y = (b_{ij})$. The determinants of all the possible hybrid matrices

$$\begin{vmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \\ b_{11} & \dots & b_{1k} & b_{1,k+1} & \dots & b_{1,k+r} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_{r1} & \dots & b_{rk} & b_{r,k+1} & \dots & b_{r,k+r} \end{vmatrix} = \begin{vmatrix} b_{1,k+1} & \dots & b_{1,k+r} \\ \vdots & \ddots & \vdots \\ b_{r,k+1} & \dots & b_{r,k+r} \end{vmatrix}$$

are exactly the shifted leading minors of C_y. □

We now show that the condition that all shifted leading minors are non-zero is very restrictive [Hofer and Larcher 2010, Proposition 4]. This allows us to simplify the original proof of Theorem 3.3.

Lemma B.3. *There is a unique upper unitriangular matrix U such that the pair (I, U) is progressive.*

Proof of Lemma B.3. The proof is by induction on the matrix size m. The base $m = 1$ is obvious.

To prove the inductive step, use the characterization of progressive pairs (I, U) from Lemma B.2. Assume there is a unique upper unitriangular $(m-1) \times (m-1)$ matrix $U = (u_{ij})$ with all shifted leading minors nonzero (this is the inductive hypothesis). It suffices to prove that there is a unique way to add the entries $u_{k,m}$ for $k = 1, \dots, m$ so that the resulting new shifted leading minors are also nonzero (the other added entries $u_{m,1} = \dots = u_{m,m-1} = 0$).

We do it by induction on k. The base $k = 1$ holds because $u_{1,m}$ must be 1 as a shifted leading minor of size 1×1. Assume that $u_{1,m}, \dots, u_{k,m}$ have already been determined. Consider the shifted leading minor Δ formed by the rows $1, \dots, k+1$ and columns $m - k, \dots, m$. Take the Laplace expansion along the last column:

$$\Delta = \Delta_1 u_{1,m} + \dots + \Delta_{k+1} u_{k+1,m},$$

where the minor Δ_j is obtained from Δ by removing the j-th row and the last column. Here $\Delta = \Delta_{k+1} = 1$ as shifted leading minors. Thus

$$u_{k+1,m} = \Delta_1 u_{1,m} + \dots + \Delta_k u_{k,m} + 1.$$

We have expressed $u_{k+1,m}$ in terms of the entries $u_{1,m}, \dots, u_{k,m}$ and the entries of the top-left $(m-1)\times(m-1)$ submatrix. By the inductive hypothesis, all those entries are uniquely determined. Then $u_{k+1,m}$ is also determined, and the lemma follows by induction. □

This argument does not construct the unique matrix U explicitly. But we actually already know the answer.

Corollary B.1. *The Pascal matrix P is the unique upper unitriangular matrix such that the pair (I, P) is progressive.*

Proof. The uniqueness has just been proved in Lemma B.3. The pair (I, P) is progressive by Theorem 3.2 and Table 2 but let us give a direct proof here. Consider the integer matrix $p_{ij} = \binom{j-1}{i-1}$. Applying [Gessel and Viennot 1985, Lemma 9] repeatedly, we conclude that each shifted leading minor of the matrix (p_{ij}) equals 1. Hence by Lemma B.2 the pair (I, P) is progressive. □

Proof of Theorem 3.3. Assume that (C_x, C_y) is a progressive pair. Then both C_x and C_y are progressive matrices. Hence $C_x = L_x U_x$ and $C_y = L_y U_y$ for some upper unitriangular matrices U_x, U_y and lower unitriangular matrices L_x, L_y. By Lemma B.1, the pair $(L_x^{-1} C_x U_x^{-1}, L_y^{-1} C_y U_x^{-1}) = (I, U_y U_x^{-1})$ is progressive. By Corollary B.1, $U_y U_x^{-1} = P$. We get $C_x = L_x U_x$ and $C_y = L_y P U_x$, as required.

Conversely, each pair $(L_x U_x, L_y P U_x)$ is progressive by Corollary B.1 and Lemma B.1. □

To count the number of possible characteristic matrices $C = L_y P L_x^{-1}$, we need the following results.

Lemma B.4. *We have $P^2 = (PJ)^3 = I$ and $PJP = JPJ$.*

Proof. The identity $P^2 = I$ by [Faure 1982] is a compact form of the well-known identity [Graham et al. 1989, (5.21) and (5.12)]

$$\sum_{j=1}^{m} \binom{j-1}{i-1}\binom{k-1}{j-1} = 2^{k-i}\binom{k-1}{i-1} \overset{\text{mod } 2}{=} I_{ik}.$$

The identity $PJP = JPJ$ by [Kajiura et al. 2018] is a compact form of the well-known identity [Graham et al. 1989, (5.14) and (5.25)]

$$\sum_{j=1}^{m} \binom{m-j}{i-1}\binom{k-1}{j-1}(-1)^{j-1} = \binom{m-k}{m-i}.$$

The identity $(PJ)^3 = I$ is a consequence of the previous two. □

Lemma B.5. *For distinct pairs (L_x, L_y) of lower unitriangular matrices, the matrices $L_y P L_x^{-1}$ are pairwise distinct.*

Proof. The matrix $C = L_y P L_x^{-1}$ uniquely determines L_x and L_y because

$$CJ = L_y P L_x^{-1} J = \underbrace{L_y JPJ}_{L} \cdot \underbrace{PJL_x^{-1}J}_{U}$$

is an LU-decomposition of CJ, which is unique. Here we used the identity $P = JPJPJ$ (see Lemma B.4) and observed that for any upper unitriangular matrix U' the matrix $JU'J$ is lower unitriangular. □

Corollary B.2. *If m is a power of 2, then under notation (3) and (11) we have $PJL_{\mathrm{LP}}^{-1} = U_{\mathrm{LP}}J$.*

Proof. First, analogously to the proof of Lemma B.4, we get $L_{\mathrm{LP}}^{-1} = L_{\mathrm{LP}}$ by the identity [Graham et al. 1989, (5.21) and (5.12)]

$$\sum_{j=2}^{m} \binom{i-2}{j-2}\binom{j-2}{k-2} = 2^{i-k}\binom{i-2}{k-2} \overset{\text{mod } 2}{=} I_{ik}.$$

Second, $PJL_{\mathrm{LP}} = U_{\mathrm{LP}}J$ by the identity [Graham et al. 1989, (5.26)]

$$\sum_{j=2}^{m} \binom{m-j}{i-1}\binom{j-2}{k-2} = \binom{m-1}{i+k-2} \overset{\text{mod } 2}{=} (U_{\mathrm{LP}})_{i,m-k+1},$$

where the latter holds since m is a power of 2. So $PJL_{\mathrm{LP}}^{-1} = U_{\mathrm{LP}}J$. □