Practical Hash-based Owen Scrambling

Brent Burley
Walt Disney Animation Studios

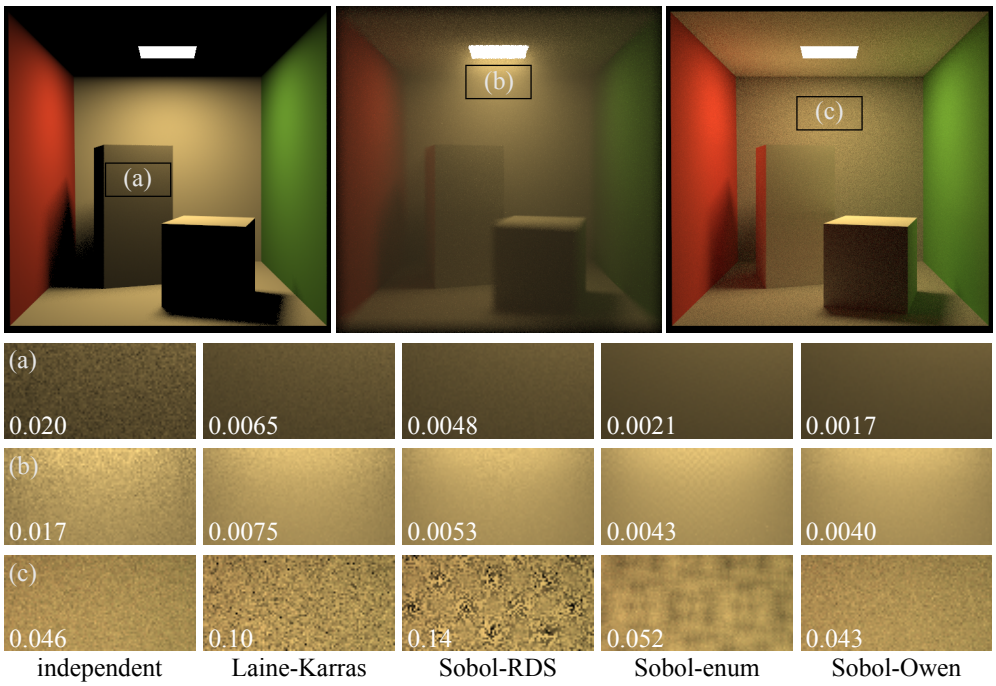

Figure 1. Top: three configurations of the Cornell Box rendered with Owen-scrambled Sobol sampling: (a) direct lighting at 16 spp, (b) global illumination with participating media and depth-of-field at 16 spp, (c) global illumination with a subpixel checkerboard texture on the back wall and floor at 64 spp. Bottom: detail images with RMSE values indicated comparing (from left to right) independent sampling, Laine-Karras sampling, Sobol sampling with random digit scrambling (RDS), Sobol-RDS sampling enumerated over the image plane, and Owen-scrambled Sobol sampling. With favorable integrands (a) and (b), Owen scrambling provides the lowest error, comparable to enumerated Sobol but without the structured artifacts faintly visible in that method for integrand (b). With an unfavorable integrand (c), unlike the other methods, Owen scrambling provides error comparable to independent sampling and avoids the structured artifacts visible in the other Sobol methods.

Abstract

Owen's nested uniform scrambling maximally randomizes low-discrepancy sequences while preserving multidimensional stratification. This enables advantageous convergence for favorable integrands and bounded error for unfavorable ones, and makes it less prone to structured artifacts than other scrambling methods. The Owen-scrambled Sobol sequence in particular has been gaining popularity recently in computer graphics. However, implementations typically use a precomputed table of samples which imposes limits on sequence length and dimension.

In this paper, we adapt the Laine-Karras hash function to achieve an implementation of Owen scrambling for the Sobol sequence that is simple and efficient enough to perform on-the-fly evaluation for sequences of indeterminate length and dimension and in arbitrary sample order, readily permitting parallel, progressive, or adaptive integration. We combine this with nested uniform shuffling to enable decorrelated reuse of the sequence for padding to higher dimensions. We discuss practical-use considerations, and we outline how hash-based Owen scrambling can be extended to arbitrary base for use with non-base-two sequences.

1. Introduction

Quasi-Monte Carlo (QMC) integration offers faster convergence for functions with bounded variation by using low-discrepancy samples rather than independent ones. In *randomized* quasi-Monte Carlo (RQMC), randomness is carefully introduced into the low-discrepancy samples to achieve unbiased integration and to perform variance estimation while preserving the low-discrepancy properties. For examples of RQMC use in computer graphics, see the works by Keller [1995; 2006; 2013]. For a general overview of RQMC methods, see the survey by L'Ecuyer and Lemieux [2005].

The Sobol sequence is a popular choice due to its multidimensional stratification and computationally efficient base-two construction, and *random digit scrambling* [Matoušek 1998] is often used as the randomization method due to its ease of application. However, of the various randomization methods, only the *nested uniform scrambling* method introduced by Owen [1995] fully randomizes a low-discrepancy sequence which can improve convergence over QMC for both best- and worst-case integrands. In addition to having potentially lower error, nested uniform scrambling (commonly referred to as "Owen scrambling") is also less susceptible to structured artifacts than random digit scrambling, as illustrated in Figure 1.

Convergence plots comparing Owen scrambling with random digit scrambling for various functions are shown in Figure 2. Independent sampling converges as $O(N^{-1/2})$ in every case, as predicted by Monte Carlo theory. For the two smooth functions, *gaussian* and *bilinear*, Sobol sampling with random digit scrambling converges as $O(N^{-1})$, while Owen-scrambled Sobol approaches $O(N^{-3/2})$ when N is a power of two. In all examples tested, Owen-scrambled Sobol results in comparable or smaller error than random digit scrambling—profoundly smaller in some cases, such as the *pulsetrain* function shown in the figure.

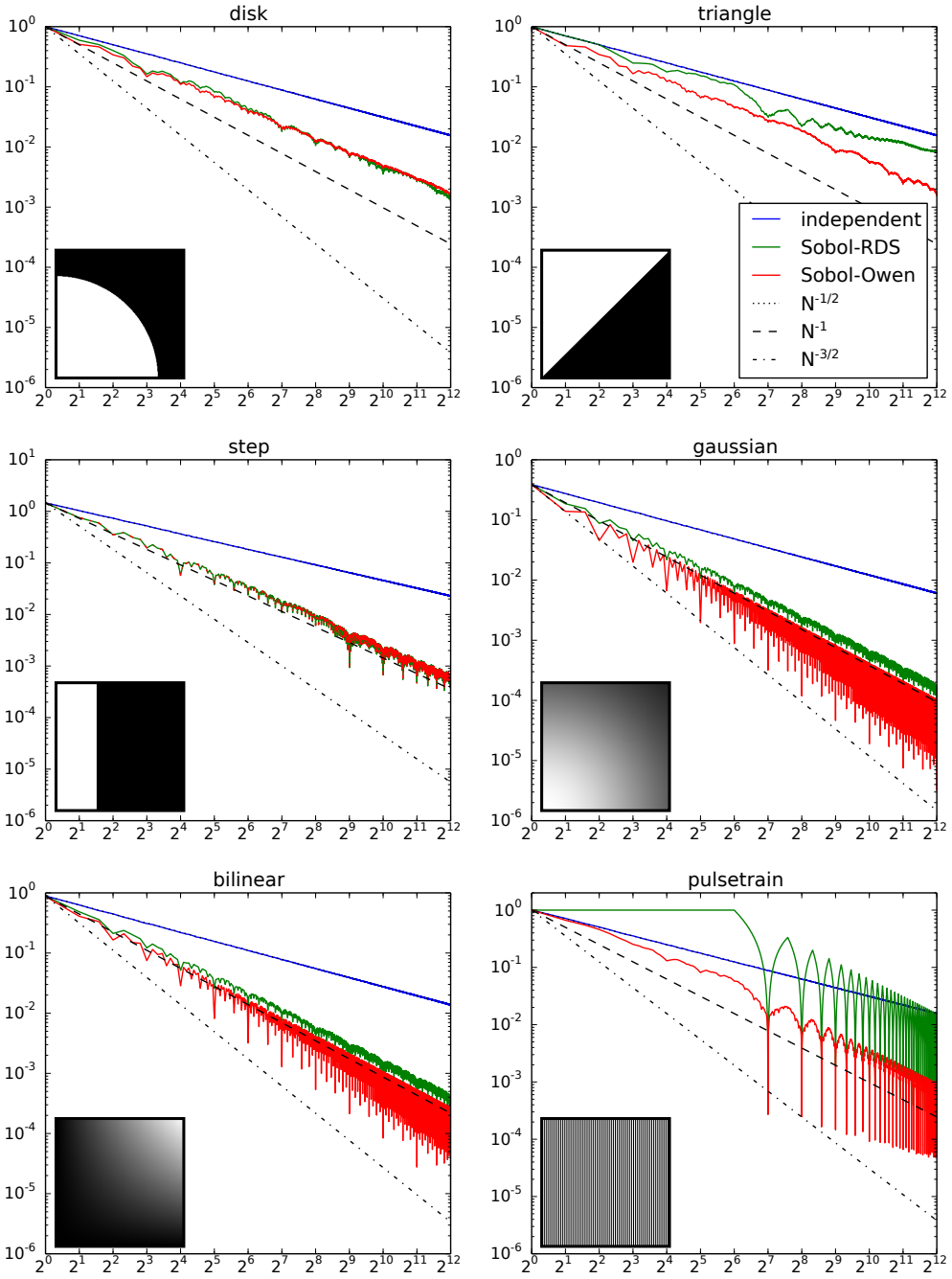

Figure 2. Convergence plots of error vs. sample count (N) for various functions (all with an expected value of 1): disk $= 2$ if $x^2 + y^2 < \frac{2}{\pi}$; triangle $= 2$ if $y > x$; gaussian $= \frac{4}{\pi \text{erf}^2 1} \exp^{-x^2-y^2}$; bilinear $= 4xy$; pulsetrain $= 2$ if $64x \bmod 1 < \frac{1}{2}$. The RMSE is computed from 10,000 trials for each value of N from 1 to 4096. All except pulsetrain are reproduced from the paper by Christensen et al. [2018].

In Section 2 we briefly review the Sobol sequence, Owen scrambling, and related work. In Section 3 we describe how Owen scrambling can be implemented using a hash function, and we show how the particularly efficient base-two hash function from Laine and Karras [2011] can be adapted to Owen scrambling. In Section 4 we show how nested uniform scrambling and nested uniform shuffling can be combined for padding the randomized Sobol sequence to higher dimension. In Section 5 we review the benefits of Owen scrambling in more detail, we discuss considerations for practical use, and we explain how hash-based Owen scrambling can be extended to arbitrary bases.

2. Background

2.1. Low-discrepancy Sequences

Discrepancy is a measure of how unevenly a point set is distributed. A *low-discrepancy point set* is one where the points are more evenly distributed, without the large gaps or significant clumps of points typical of randomly distributed points. A *low-discrepancy sequence* is a progressive construction that fills in the gaps and maintains the low-discrepancy quality as points are added.

Some low-discrepancy sequences can be characterized in *(t,s)*-notation [Niederreiter 1987] where s is the dimensionality and t is a measure relating to the discrepancy:

> A *(t,s)-sequence* constructed in base b can be considered an infinite series of *(t,m,s)-nets* for every positive integer m, each net being a set of b^m points, with each *elementary interval* of volume $1/b^m$ containing at most b^t points.

Importantly, if $t = 0$, all elementary intervals contain exactly one point and the sequence is considered to be perfectly stratified, as illustrated in Figure 3.

The *radical-inverse sequence* is a (0,1)-sequence where the base-b digits of the sample index are simply reflected across the radix point. For example, in base two, sample index $13 = 1101_2$ has value $0.1011_2 = 1/2+1/8+1/16 = 11/16$. Most low-discrepancy sequences are derived in some way from the radical-inverse sequence.

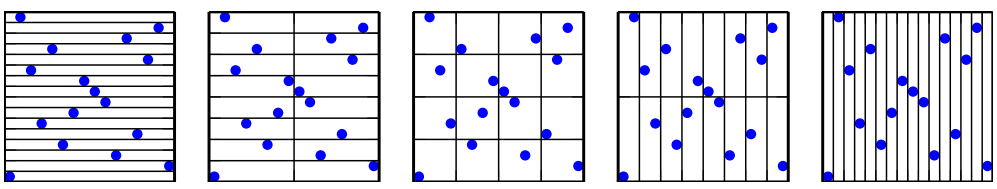

Figure 3. The first 16 points of the 2D Sobol sequence with all elementary intervals with volume $1/16$ indicated. Observe that each elementary interval contains exactly one point.

2.2. Sobol Sequence

The Sobol sequence [Sobol' 1967] is a multidimensional (t, s)-sequence in base two. Each dimension uses the radical-inverse sequence but with carefully permuted sample ordering. For instance, the first 16 samples of the Sobol sequence in *every* dimension are: (0, 1/2, {1/4, 3/4}, {{1/8, 5/8}, {3/8, 7/8}}, {{{1/16, 9/16}, {5/16, 13/16}}, {{3/16, 11/16}, {7/16, 15/16}}}) where braces indicate allowable permutations. Nested binary permutations of this form preserve the 1D stratification of the radical-inverse sequence such that each dimension remains a (0,1)-sequence.

The Sobol permutations are not chosen randomly, but rather are carefully chosen to reduce multidimensional discrepancy. The first two dimensions of the Sobol sequence are perfectly stratified, forming a (0,2)-sequence, but the higher-dimensional sequence in general has $t > 0$.

The permutations are applied using a matrix of *direction numbers* that is predefined for each dimension. Conceptually, to compute a sample, the vector of binary digits from the sample index is multiplied by the matrix (mod 2) to produce the sample value as a binary fraction. In practice, each non-zero bit from the sample index is used to select the corresponding direction number, and these direction numbers are simply XOR'd together to produce the sample value [Bratley and Fox 1988]. An implementation is given in the supplemental materials.

2.3. Padding and Shuffling

Sampling using high-dimensional Sobol points can be problematic due to the t numbers increasing as the dimensionality is increased. According to Dick and Niederreiter [2008, Remark 4], the Sobol sequence with four dimensions has $t = 3$, with five dimensions $t = 5$, and with six–ten dimensions $t = (8, 11, 15, 19, 23)$, respectively. Owen [1998b] reported that advantageous convergence rates are only achieved for sample counts of $N \gtrsim 2^{ts}$ which suggests, for example, that integrating with 4D Sobol points may require on the order of 2^{12} samples.

Instead of using a high-dimensional Sobol sequence, a low-dimensional sequence can be used for the first few dimensions and the remaining dimensions can simply be "padded" with uniform random samples [Spanier 1995]. Alternatively, Owen [1998a] showed that better results can be obtained by padding with shuffled RQMC point sets. Shuffling the order of points in each reused point set effectively decorrelates the reused dimensions while preserving the properties within each set.

Shuffling a fixed, pregenerated point set is straightforward, but this would not be suitable for progressive sampling. As an alternative to shuffling a fixed point set, Laine and Karras [2011] used a hash function to progressively shuffle the sample order of the base-two radical-inverse sequence for decorrelated reuse. As with the Sobol permutations, the Laine-Karras hash preserves 1D stratification in each dimension;

however, unlike the Sobol permutations, the Laine-Karras hash achieves no multidimensional stratification.

In Section 3.1 we analyze the Laine-Karras hash and show that it achieves a *nested uniform shuffle*, recursively swapping complete subnets by permuting the digits of the sample index, thus preserving the low-discrepancy properties of the sequence for all sample counts. In Section 4 we show how to use the hash to shuffle the multidimensional Sobol sequence.

2.4. Owen Scrambling

To fully randomize (t,s)-sequences, Owen [1995] proposed applying a unique permutation to each elementary interval. The set of all elementary intervals can be imagined as a b-ary tree, as in Figure 5, with the most significant digit of a sample value selecting a topmost interval in the tree, and each successive digit selecting a subinterval. Permuting a digit effectively rearranges the subintervals but does not change the number of points in those or any other subintervals. Owen referred to this as nested uniform scrambling, and this has come to be known simply as Owen scrambling.

Owen scrambling requires a large number of independent permutations (sb^K where K digits are being permuted) which could require a prohibitive amount of memory. Friedel and Keller [2002] implemented Owen scrambling for *fixed-length* point sets in arbitrary base without requiring excessive memory or computation. The authors first generated complete sets of unscrambled points, then sorted and permuted them recursively by interval.

Avoiding the sequence length restriction and eliminating the memory requirement entirely, Owen [2003] suggested the much simpler possibility of using a hash of the interval's address to compute permutations on-the-fly. Recomputing permutations on-the-fly for each generated digit could be inefficient for large b, however this is not a problem for us as we only need base two. While performing such a per-digit permutation is straightforward, it still involves quite a few steps of computation. The Laine-Karras hash by comparison permutes all the digits at once. We show in Section 3.2 that a nested uniform scramble can be achieved using the Laine-Karras hash if applied in a bit-reversed order.

2.5. Scrambling vs. Shuffling

It is important to distinguish between scrambling and shuffling. Scrambling refers to randomizing the sample *value* whereas shuffling refers to reordering samples by randomizing the sample *index*. While scrambling aims to improve the properties of the sequence, shuffling is intended solely to decorrelate the ordering of samples for padding without otherwise changing the properties of the sequence. Owen [1998a] explained why padding with unshuffled points using independent scramblings (as suggested by Kollig and Keller [2002]) cannot be expected to work well in practice; this is

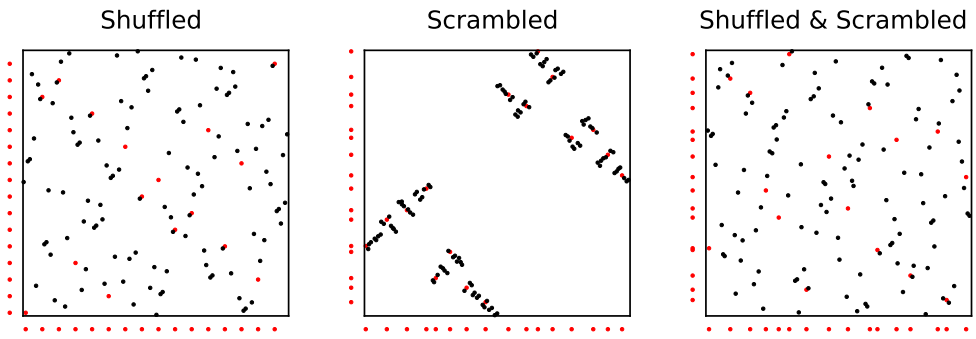

Figure 4. 128 radical-inverse points padded to 2D; the first 16 points are colored red with 1D projections shown in the margins. Left: Shuffling effectively decorrelates the two dimensions but adds no randomization, as exhibited by the regular spacing in the 1D projections. Middle: Scrambling adds randomization, visible in the jittered spacing in the 1D projections, but fails to decorrelate the two dimensions. Right: Combining scrambling and shuffling achieves randomization and decorrelation.

demonstrated in Figure 4. The figure also demonstrates that scrambling and shuffling can be combined with complementary results.

2.6. Owen Scrambling Approximations

For efficient scrambling in arbitrary base, Matoušek [1998] proposed various approximations to Owen scrambling in order to avoid the memory and computation required.

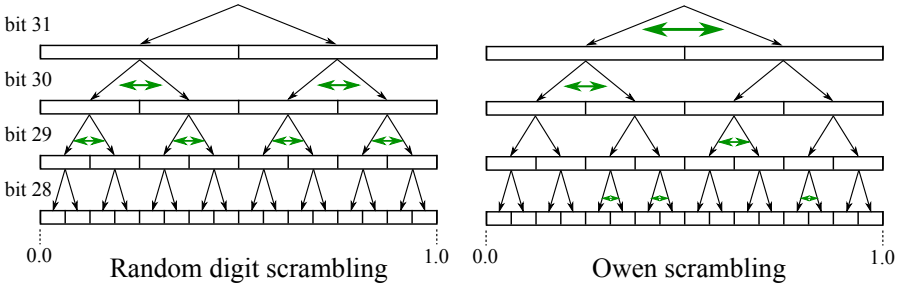

Figure 5. In base two, each bit of a sample value's binary fraction, starting with the most significant bit, places the sample value into a particular sub-interval. Toggling a bit for all samples swaps all pairs of sub-intervals corresponding to that bit; this is illustrated above for random digit scrambling where bits 29 and 30 are toggled for all samples. Alternatively, if a bit is toggled only for samples falling within a particular interval, then only the two subintervals within that interval will be swapped; this is illustrated for Owen scrambling where an arbitrary subset of intervals is swapped. Both scrambling methods preserve stratification as the number of points in each interval is unchanged, but Owen scrambling achieves a more complete randomization.

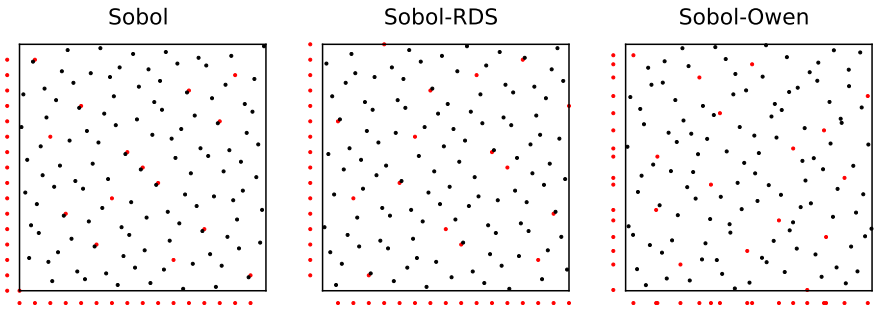

Figure 6. The first 128 points of the 2D Sobol sequence along with two scramblings; the first 16 points are colored red with 1D projections in the margins. Left: The Sobol points are perfectly stratified but have regular spacing in 1D projections, and the points tend to align on diagonals in 2D. Middle: Random digit scrambling offsets the 1D projections but does not affect their regular spacing, and though mutated, the structure and alignment of 2D points remain. Right: Owen scrambling jitters the points in 1D and 2D while preserving their stratification.

Two notable approximations proposed by Matoušek are *nested linear scrambling*, where nested permutations of the form $Ax_j + C \mod b$ are used rather than fully uniform ones, and *random digit scrambling*, where nested permutations are replaced with positional ones, using an independent permutation per digit position rather than per interval.

Kollig and Keller [2002] implemented random digit scrambling in base two using bitwise XOR, and this remains a widely used method for scrambling the Sobol sequence. An illustration comparing Owen scrambling and random digit scrambling is shown in Figure 5, and a sample point set is shown in Figure 6.

Owen [2003] proved (Prop. 3.1) that both random linear scrambling and random digit scrambling are sufficient for unbiased integration. Owen also proved (Prop. 3.3) that random linear scrambling achieves both the smooth-integrand convergence rate and the square-integrable variance bound of nested uniform scrambling, but random digit scrambling achieves neither.

2.7. Related Work

Grünschloß et al. [2012] performed image synthesis using a single low-discrepancy sequence spread across the image plane, allocating a subinterval of the sequence to each pixel. To allow parallel rendering, the authors devised an algorithm to efficiently enumerate the samples within a given pixel. Using a single sequence for the entire image achieves stratification across pixels rather than solely within each pixel, and this can reduce the overall image error as seen in Figure 1. However, the sequence is still subject to structured artifacts as shown in the figure.

Perrier et al. [2018] applied randomly selected, precomputed Owen-scrambling subtrees to the Sobol sequence that were each optimized for 2D blue-noise quality rather than using independent permutations per interval. For higher-dimensional sampling, the authors applied their blue-noise scrambling to two or three select pairs of dimensions, and applied ordinary Owen scrambling to remaining dimensions.

Christensen et al. [2018] generated fully randomized $(0, 2)$-sequences by brute-force search while (optionally) optimizing for blue-noise properties. The authors extended the sequence to four dimensions by pairing 2D sequences and making greedy swaps of strata pairs within the second sequence to obtain 4D stratification. For independent uses of the sequence, and to extend sequence length and dimensionality, the authors randomly selected among precomputed realizations of the sequence, each shuffled during construction for decorrelation. The shuffling is performed progressively such that each prefix of samples with power-of-two length remains a complete net.

Heitz et al. [2019] applied random digit scrambling to a precomputed Owen-scrambled Sobol sequence, rearranging the seeds in an optimization step to achieve a blue-noise distribution of error across the image plane. In addition to having fixed sequence length and dimension, the authors used the same Owen scrambling for every pixel to minimize table size. A hash-based implementation presumably would allow the Owen scrambling to be varied per pixel (in place of the random digit scrambling), in addition to eliminating the length and dimension limits.

3. Hash-based Owen Scrambling

Owen's nested uniform scrambling is formulated as follows. Given the b-ary representation of unscrambled sample value $x = 0.a_1a_2a_3...a_k$, the scrambled sample is $\sigma(x) = 0.b_1b_2b_3...b_k$ where $b_j = \pi_{a_1,a_2,...,a_{j-1}}(a_j)$ with $\pi_{a_1,a_2,...,a_{j-1}}$ each being an independently chosen uniform permutation of the set $\{0, 1, ..., b - 1\}$. That is, for each digit of the b-ary fraction, the digits to the left are used to select a permutation to be applied to the digit. These left-most digits thus represent the "address" of the interval containing the digit being scrambled.

A uniform digit permutation in base two is trivially achieved by XOR'ing the given digit with a random bit value. Implementing a nested uniform permutation is straightforward: for each bit, use a hash of the bits to the left (along with a unique randomization seed) to generate the random bit value and XOR the random bit with the bit from the sample value. The cost in a naive implementation is proportional to the number of bits being permuted. However, Laine and Karras devised a more efficient nested permutation which we will now consider.

```
uint32_t laine_karras_permutation(uint32_t x, uint32_t seed)
{
    x += seed;
    x ^= x * 0x6c50b47cu;
    x ^= x * 0xb82f1e52u;
    x ^= x * 0xc7afe638u;
    x ^= x * 0x8d22f6e6u;
    return x;
}
```

Listing 1. Laine-Karras permutation.

3.1. Laine-Karras Permutation

Laine and Karras [2011] proposed a particularly efficient nested uniform permutation in base two, shown in Listing 1. Laine and Karras observed that when multiplying an input value by an even constant, each input bit only affects bits to the left and thus performs a nested hash. And because each multiplication result is used only to select one of the two possible permutations of each bit (via the XOR operator), the result is a nested uniform permutation.

Laine and Karras used fixed multipliers that they found to have good hashing properties, resulting in a deterministic permutation. To add randomness, a provided seed is added into the sample value before applying the permutation. When adding the constant, in addition to each bit affecting bits to the left through arithmetic carry, each bit also affects itself; however, this constant value added to each bit merely acts to select an additional fixed permutation of the bit. Thus, adding the random constant can also be viewed as a nested uniform permutation, and the composition of nested uniform permutations is likewise a nested uniform permutation.

The first 16 samples of a Laine-Karras-permuted sequence are shown in Table 1. In the third column of the table, we demonstrate that the Laine-Karras sequence is a nested binary permutation composed with a random digit scrambling. We interpret the permutation of $x = 0$ as the random digit-scrambling constant, `0x71b1c2ac`. XOR'ing this value with the entire column recovers the "unscrambled" permuted sequence which we can see conforms to the same allowable nested permutations as we described in Section 2.2 for the Sobol sequence. As pointed out in their paper [Laine and Karras 2011], the Laine-Karras hash replaces the Sobol direction numbers with random permutations rather than ones that minimize multidimensional discrepancy. However, the result is *not* an Owen scramble as the hash serves primarily to permute the sample order, achieving only minimal randomization equivalent to random digit scrambling; thus, it is more appropriate to refer to the Laine-Karras permutation as a *nested uniform shuffle*.

radical-inverse		Laine-Karras	$\oplus$ 71b1c2ac	
00000000	(0)	71b1c2ac	00000000	(0)
80000000	(1/2)	f1b1c2ac	80000000	(1/2)
40000000	(1/4)	b1b1c2ac	c0000000	(3/4)
c0000000	(3/4)	31b1c2ac	40000000	(1/4)
20000000	(1/8)	d1b1c2ac	a0000000	(5/8)
a0000000	(5/8)	51b1c2ac	20000000	(1/8)
60000000	(3/8)	11b1c2ac	60000000	(3/8)
e0000000	(7/8)	91b1c2ac	e0000000	(7/8)
10000000	(1/16)	c1b1c2ac	b0000000	(11/16)
90000000	(9/16)	41b1c2ac	30000000	(3/16)
50000000	(5/16)	01b1c2ac	70000000	(7/16)
d0000000	(13/16)	81b1c2ac	f0000000	(15/16)
30000000	(3/16)	a1b1c2ac	d0000000	(13/16)
b0000000	(11/16)	21b1c2ac	50000000	(5/16)
70000000	(7/16)	e1b1c2ac	90000000	(9/16)
f0000000	(15/16)	61b1c2ac	10000000	(1/16)

Table 1. The first 16 samples of the radical-inverse sequence and corresponding Laine-Karras permutation. The first column is the input sample value, x, in binary fraction and decimal forms. The middle column is `laine_karras_permutation(x, 0x552553bc)` where `0x552553bc` is an arbitrarily chosen seed. The third column shows values from the middle column after XOR'ing with 0x71b1c2ac (the permutation of $x = 0$), making it easier to see that the Laine-Karras sequence is a nested binary permutation.

3.2. Nested Uniform Scrambling in Base Two

As previously described, Owen scrambling permutes each digit of the sample value based on the digits of higher significance (i.e., ones to the left), whereas the Laine-Karras hash permutes each digit based on digits of lower significance (i.e., ones to the right). We observe therefore that if the Laine-Karras permutation is applied in reverse, as shown in Listing 2, a nested uniform scramble results.

```
uint32_t nested_uniform_scramble(uint32_t x, uint32_t seed)
{
    x = reverseBits(x);
    x = laine_karras_permutation(x, seed);
    x = reverseBits(x);
    return x;
}
```

Listing 2. Nested uniform scrambling in base two.

4. Shuffled Scrambled Sobol Sampling

As demonstrated in Section 3.1, the Laine-Karras permutation achieves a nested uniform shuffle when applied to the sample values of the radical inverse sequence. To adapt this to shuffling the multidimensional Sobol sequence we must apply it to the sample index before any sample values are computed. To do so, we simply reverse the bits before and after applying the permutation.

This (perhaps confusingly) results in exactly the same code as for nested uniform scrambling shown in Listing 2. The critical distinction is that the function is applied to the sample index rather than the sample value to achieve a nested uniform shuffle rather than a nested uniform scramble.

We can easily verify that our shuffling approach is equivalent to that used by Laine and Karras [2011, Figure 4]. In their code, the sample index is given as `sampleId + pixelId*samplesPerPixel`. We first shuffle the given sample index using Listing 2 (with `seed=surfaceId`) and then we reverse the bits of the shuffled index to compute the corresponding radical inverse sample value. We note that this final `reverseBits` cancels with the second `reverseBits` function in Listing 2; if these two calls are eliminated, then the remaining code is identical to Laine and Karras'. The difference in our approach is that we can now apply the shuffling to sequences of arbitrary dimension, and we can readily combine shuffling with scrambling.

Putting this all together, we perform shuffled scrambled Sobol sampling using the code in Listing 3. We first apply the `nested_uniform_scramble` function from Listing 2 to the sample index, achieving a nested uniform shuffle of the sample ordering. We then use the shuffled index to compute the corresponding 4D Sobol sample. Finally, we apply `nested_uniform_scramble` again to each dimension, using a different seed in each dimension. The first 256 points generated using this code are shown in Figure 7.

For padding to higher dimensions, one merely needs to supply a different seed to each group of four dimensions. This effectively decorrelates each group of four dimensions against all others while preserving the stratification within each group.

```
void shuffled_scrambled_sobol4d(uint32_t index, uint32_t seed,
                                uint32_t X[4])
{
    index = nested_uniform_scramble(index, seed);
    sobol4d(index, X);
    for (int i = 0; i < 4; i++) {
        X[i] = nested_uniform_scramble(X[i], hash_combine(seed, i));
    }
}
```

Listing 3. Shuffled scrambled Sobol sampling.

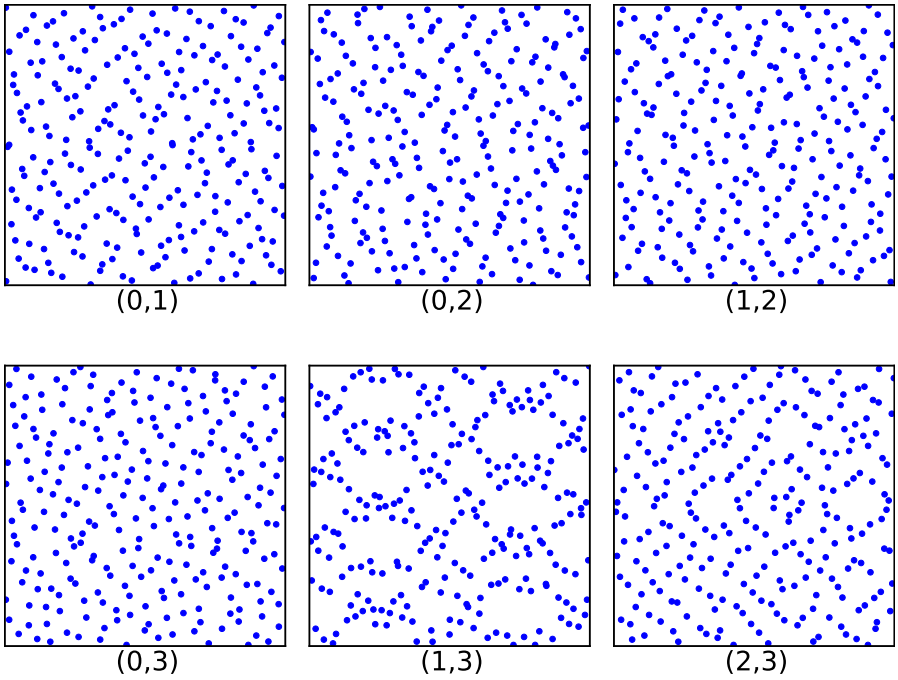

Figure 7. 256 4D points generated using Listing 3 with `seed=12345` with all 2D projections shown. Stratification can be observed in all projections, though higher discrepancy can be observed in the higher dimensions.

5. Discussion

5.1. Benefits of Owen Scrambling

Improved rate of convergence. For smooth-enough functions, i.e., ones with bounded mixed-partial derivatives, the expected RMSE for Owen-scrambled nets reduces to $O\left(N^{-3/2}(\log N)^{(s-1)/2}\right)$ rather than $O\left(N^{-1}(\log N)^{s-1}\right)$ for QMC [Owen 1997b]. This is evidenced in the integrals of the *gaussian* and *bilinear* functions in Figure 2.

Owen offered an intuitive explanation of the improved convergence rate in the 1D case [Owen 1997b, attributed to Fred Hickernell]. An unscrambled radical-inverse net of $N = b^m$ points has each data point at the lower end of its $1/N$ interval. Given large enough N and a "well-behaved" integrand, most intervals will have their largest or smallest value at these points with no cancellation generally expected, and thus the error is typically proportional to $1/N$. Given that Owen-scrambled nets randomize samples independently within their intervals, there is a tendency towards error cancellation with an accordingly lower convergence rate.

This explanation also provides a caution in that optimal convergence may only be expected when sampling with complete nets (i.e., with $N = b^m$), and adding even one more sample may add error proportional to $1/N$; this effect is illustrated in Figure 8.

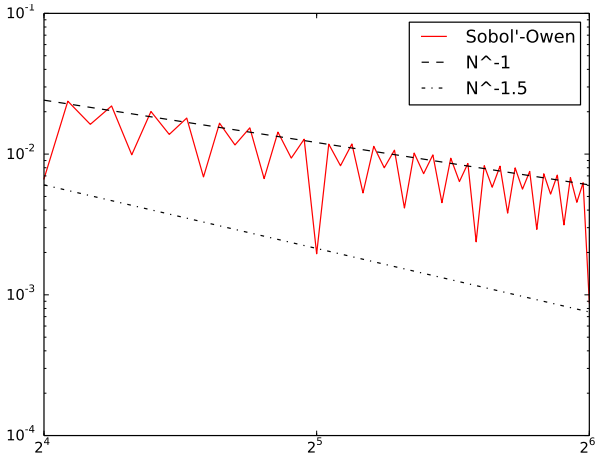

Figure 8. Closeup of convergence plot for gaussian function from Figure 2. The error is seen to increase and decrease with sample count as full nets are reached. At powers of 2, the convergence follows $O(N^{-1.5})$. However, adding just one more sample adds error proportional to $O(N^{-1})$. Additionally, the error reduces to a local minimum halfway between powers of two, e.g, at 24 and 48 samples.

It is likely the smooth integrand condition is satisfied rarely in practice, but it clearly can occur, for instance in computing unshadowed direct lighting from an area light as in Figure 1(a). Also, if an integrand is smooth over part of the domain, one would expect error cancellation at least within nets contained within that part. This may not improve the convergence rate as the error for the non-smooth part may only be proportional to $1/N$ (or worse), but it could still reduce the overall error. Alternately, if the integrand is piecewise-smooth with parts falling on sub-interval boundaries, then one would expect error cancellation across the domain once a sufficient sample count is reached.

Bounded error. For square-integrable functions with unbounded variation, unlike QMC which can have unbounded variance, integrating with Owen-scrambled $(0, m, s)$-nets offers a variance bound of $\frac{\sigma^2}{N}\,(b/(b-1))^{\min(s-1,m)}$ which is never more than $\mathrm{e} \doteq 2.72$ times greater than with independent sampling [Owen 1997a]. This is a quality that Owen has referred to as "do no harm," suggesting that an Owen-scrambled net is more robust against unfavorable integrands.

Avoiding structured artifacts When sampling a function at regular intervals such as the 1D projections of Sobol-RDS sampling, aliasing can occur when the sample rate is lower than frequencies contained within the function. This can manifest as phantom low-frequencies in the rendered image, such as the artifacts in Figure 1(c) for Sobol-RDS and (b) and (c) for Sobol-enum. Jittering the sample positions, such as with Owen scrambling, avoids such aliasing.

5.2. On the Laine-Karras Hash

The Laine-Karras hash *is* a nested uniform permutation—the permutations are demonstrably nested and are obviously uniform in the sense that the permutations for each digit are randomly and equally chosen from the set of possible permutations (of which there are only two possibilities for each bit). However, the "randomness" of the hash is clearly affected by how many multiply-xor steps are included, and which particular constants are used. This apparent contradiction comes from the fact that when we say random we really mean *pseudorandom*, and how random this actually is can vary to a large degree. Measuring randomness and hash quality is a complicated endeavor and beyond the scope of this paper; as such, we simply include the Laine-Karras hash as-is and rely on the testing and claims by the authors that the hash was chosen to have good properties. For what it is worth, in our testing we have found the hash to work well, with convergence comparable to the naive per-bit hash using a well-regarded hash function, and we cannot justify making any changes to it.

5.3. Practical Use

Padding. As previously discussed, when performing integration with less than a few thousand samples (which is typical for computer graphics), there may be little benefit to using more than four dimensions, yet integrals often have many more dimensions; e.g., multi-bounce path tracing can require hundreds. The question remains whether there is merit to padding with the randomized Sobol sequence rather than using independent pseudo-random sampling that is simpler to compute. One argument to do so is that an integral may have low *effective dimension* where a select few dimensions contribute to most of the variance, yet it may not be known which dimensions those are. For instance, lens samples may dominate for out-of-focus depth-of-field, but may be unimportant for in-focus regions; likewise, the time dimension may dominate for a pixel encountering high motion blur, but may be unimportant for static or slow-moving objects. Even dimensions deep along the path may dominate, for example, if a path undergoes several bounces of smooth reflection before hitting a diffuse brightly lit surface. In these cases, padding with low-discrepancy samples for all dimensions ensures that the important ones will be sampled well when the other dimensions are unimportant.

Use in a Path Tracer Each path will typically carry the sample index as well as a random seed for deterministic sampling; the sample index remains constant for the entire path. This seed should be varied per pixel and may have other data such as the frame number mixed in, and a high-quality hash should be applied to the seed. A convenient strategy for path tracing is to draw a single multi-dimensional sample per bounce, and then advance the seed (e.g., using a random number generator) for the next bounce. If more than four dimensions are needed per bounce, then the seed can be advanced immediately to sample additional dimensions.

Potential Optimizations

- If a maximum sample count is known, then some of the Sobol direction numbers can be omitted. For example, only the first 16 directions are needed for sequences up to 65536 samples. The direction numbers can also be easily vectorized.
- In Listing 3, a bit reversal of `x` occurs both before and after the call to `sobol` as part of `nested_uniform_scramble`. These bit reversals can be eliminated by flipping the matrix of direction numbers in both directions.
- The first Sobol dimension is the 1D radical-inverse sequence and can be computed as a special case by bit reversal rather than using direction numbers.
- Some architectures have an intrinsic instruction for bit reversal, e.g., `__brev` in CUDA.

5.4. Extension to Arbitrary Base

Having arbitrary base support would allow scrambling of the popular Halton sequence that uses a different prime base in every dimension. However, it must be pointed out that the beneficial properties of Owen scrambling, such as improved convergence rate and strict error bound, rely on complete nets which is impossible with the Halton sequence given that each dimension reaches a complete net at a different sample count. For unbiased integration and variance estimation, random-digit scrambling is already sufficient, and state-of-the-art deterministic scramblings [Faure and Lemieux 2009] have been found to outperform randomly chosen ones for the Halton sequence.

An additional application of arbitrary base support would be to scramble so-called $(0, s)$-sequences which have the optimal $t = 0$ property and are constructed in base $b \geq s$ [Faure 1982; Faure 2001]. The beneficial properties of Owen scrambling would now be applicable for complete nets.

To sample with complete nets of size b^m, b must necessarily remain small. For instance, complete nets for $b = 5$ permit sample counts of 5, 25, 125, 625, etc., and these jumps may already be impractically large for progressive rendering. However, there may still be a (reduced) benefit as long as complete sub-nets are used, e.g., using multiples of 5 up to 25, multiples of 25 up to 125, multiples of 125 up to 625, etc.

Application of hash-based Owen scrambling in arbitrary bases is straightforward; the main challenge is not in computing the hash but rather in computing uniform permutations efficiently on-the-fly. For small-enough bases, the set of possible permutations can simply be pretabulated. For instance, the set of all base five permutations requires only $5! \cdot 5 = 600$ bytes; this would already permit scrambling a (0,5)-sequence. However, bases $b \gtrsim 9$ are likely impractical to pretabulate, in which case on-the-fly permutations must be used.

On-the-fly *uniform* permutations would likely by impractical for large b. However, as mentioned in Section 2.6, nested linear scrambling provides all the benefits of nested uniform scrambling without the overhead of computing full uniform permutations. Instead, permutations of the form $Ax_j + C \mod b$ are used, with A and C chosen from the sets $\{1, 2, ..., b-1\}$ and $\{0, 1, ..., b-1\}$, respectively; these constants can be computed using a hash of the interval's address and digit position. Though the per-digit cost may be significant, the number of digits that need to be scrambled is likely small, especially for higher bases. It is unknown to the author whether it is possible to scramble multiple digits at once in an arbitrary base as with the Laine-Karras hash.

6. Conclusion

In this paper, we have presented an efficient implementation of hash-based Owen-scrambling for Sobol sampling that also performs nested uniform shuffling to enable multidimensional padding, and we have discussed various practical considerations for its use. A complete implementation is given in the supplemental materials. A point visualizer is also provided in the supplemental materials, along with an implementation of the Owen-scrambled Faure (0,5) sequence.

Acknowledgments

I would like to thank Per Christensen, Andrew Kensler, and Thomas Müller for their helpful insights, and especially Per for encouraging me to write this paper. I would also like to the thank the anonymous reviewers for their many corrections and suggestions.

Supplemental Materials

Supplemental materials are available at http://jcgt.org/published/0009/04/01/suppl.zip.

References

BRATLEY, P., AND FOX, B. L. 1988. Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software 14*, 1 (Mar.), 88–100. URL: http://doi.acm.org/10.1145/42288.214372. 5

CHRISTENSEN, P., KENSLER, A., AND KILPATRICK, C. 2018. Progressive Multi-Jittered sample sequences. *Computer Graphics Forum 37*, 4, 21–33. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13472. 3, 9

DICK, J., AND NIEDERREITER, H. 2008. On the exact t-value of Niederreiter and Sobol' sequences. *Journal of Complexity 24*, 5-6 (Oct.), 572–581. URL: http://dx.doi.org/10.1016/j.jco.2008.05.004. 5

FAURE, H., AND LEMIEUX, C. 2009. Generalized Halton sequences in 2008: A comparative study. *ACM Transactions on Modeling and Computer Simulation 19*, 4 (Nov.), 15:1–15:31. URL: http://doi.acm.org/10.1145/1596519.1596520. 16

FAURE, H. 1982. Discrépance de suites associées à un système de numération (en dimension s). *Acta Arithmetica 41*, 4, 337–351. URL: http://eudml.org/doc/205851. 16

FAURE, H. 2001. Variations on (0,s)-sequences. *Journal of Complexity 17*, 4, 741–753. URL: http://www.sciencedirect.com/science/article/pii/S0885064X01905904. 16

FRIEDEL, I., AND KELLER, A. 2002. Fast generation of randomized low-discrepancy point sets. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, Springer, Berlin,Heidelberg, Germany, K.-T. Fang, H. Niederreiter, and F. J. Hickernell, Eds., 257–273. URL: https://doi.org/10.1007/978-3-642-56046-0_17. 6

GRÜNSCHLOSS, L., RAAB, M., AND KELLER, A. 2012. Enumerating quasi-monte carlo point sequences in elementary intervals. In *Monte Carlo and Quasi-Monte Carlo Methods 2010*, Springer, Berlin, Heidelberg, Germany, L. Plaskota and H. Woźniakowski, Eds., 399–408. URL: https://link.springer.com/chapter/10.1007/978-3-642-27440-4_21. 8

HEITZ, E., BELCOUR, L., OSTROMOUKHOV, V., COEURJOLLY, D., AND IEHL, J.-C. 2019. A low-discrepancy sampler that distributes monte carlo errors as a blue noise in screen space. In *ACM SIGGRAPH 2019 Talks*, Association for Computing Machinery, New York, NY, USA, SIGGRAPH 19. URL: https://doi.org/10.1145/3306307.3328191. 9

KELLER, A. 1995. A Quasi-Monte Carlo algorithm for the global illumination problem in the radiosity setting. In *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, Springer, Berlin, Heidelberg, Germany, H. Niederreiter and P. J.-S. Shiue, Eds., 239–251. URL: https://link.springer.com/chapter/10.1007/978-1-4612-2552-2_15. 2

KELLER, A. 2006. Myths of computer graphics. In *Monte Carlo and Quasi-Monte Carlo Methods 2004*, Springer, Berlin, Heidelberg, Germany, H. Niederreiter and D. Talay, Eds., 217–243. URL: https://link.springer.com/chapter/10.1007/3-540-31186-6_14. 2

KELLER, A. 2013. Quasi-Monte Carlo image synthesis in a nutshell. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, Springer, Berlin, Heidelberg, Germany, J. Dick, F. Kuo, G. Peters, and I. Sloan, Eds., 203–238. URL: https://link.springer.com/chapter/10.1007/978-3-642-41095-6_8. 2

KOLLIG, T., AND KELLER, A. 2002. Efficient multidimensional sampling. *Computer Graphics Forum 21*, 3, 557–563. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00706. 6, 8

LAINE, S., AND KARRAS, T. 2011. Stratified sampling for stochastic transparency. *Computer Graphics Forum 30*, 4, 1197–1204. URL: https://diglib.eg.org/handle/10.1111/v30i4pp1197-1204. 4, 5, 10, 12

L'ECUYER, P., AND LEMIEUX, C. 2005. *Recent Advances in Randomized Quasi-Monte Carlo Methods*. Springer US, Boston, MA, 419–474. URL: https://doi.org/10.1007/0-306-48102-2_20. 2

MATOUŠEK, J. 1998. On the L_2-discrepancy for anchored boxes. *Journal of Complexity 14*, 4 (Dec.), 527–556. URL: http://dx.doi.org/10.1006/jcom.1998.0489. 2, 7

NIEDERREITER, H. 1987. Point sets and sequences with small discrepancy. *Monatshefte für Mathematik 104*, 4, 273–337. URL: https://doi.org/10.1007/BF01294651. 4

OWEN, A. B. 1995. Randomly permuted (t,m,s)-nets and (t, s)-sequences. In *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, Springer, Berlin, Heidelberg, Germany, H. Niederreiter and P. J.-S. Shiue, Eds., 299–317. URL: https://doi.org/10.1007/978-1-4612-2552-2_19. 2, 6

OWEN, A. B. 1997. Monte Carlo variance of scrambled net quadrature. *SIAM Journal on Numerical Analysis 34*, 5, 1884–1910. URL: https://doi.org/10.1137/S0036142994277468. 14

OWEN, A. B. 1997. Scrambled net variance for integrals of smooth functions. *The Annals of Statistics 25*, 4, 1541–1562. URL: http://www.jstor.org/stable/2959062. 13

OWEN, A. B. 1998. Latin Supercube Sampling for very high-dimensional simulations. *ACM Transactions on Modeling and Computer Simulation 8*, 1 (Jan.), 71–102. URL: http://doi.acm.org/10.1145/272991.273010. 5, 6

OWEN, A. B. 1998. Scrambling Sobol'and Niederreiter–Xing points. *Journal of complexity 14*, 4, 466–489. URL: https://doi.org/10.1006/jcom.1998.0487. 5

OWEN, A. B. 2003. Variance with alternative scramblings of digital nets. *ACM Transactions on Modeling and Computer Simulation. 13*, 4 (Oct.), 363–378. URL: http://doi.acm.org/10.1145/945511.945518. 6, 8

PERRIER, H., COEURJOLLY, D., XIE, F., PHARR, M., HANRAHAN, P., AND OSTROMOUKHOV, V. 2018. Sequences with low-discrepancy blue-noise 2-D projections. *Computer Graphics Forum 37*, 2, 339–353. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13366. 9

SOBOL', I. M. 1967. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki 7*, 4, 784–802. URL: https://doi.org/10.1016/0041-5553(67)90144-9. 5

SPANIER, J. 1995. Quasi-Monte Carlo methods for particle transport problems. In *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, Springer, Berlin,Heidelberg, Germany, H. Niederreiter and P. J.-S. Shiue, Eds., 121–148. URL: https://link.springer.com/chapter/10.1007/978-1-4612-2552-2_6. 5

Author Contact Information

Brent Burley
Walt Disney Animation Studios
500 S. Buena Vista St.
Burbank, CA 91521
brent.burley@disneyanimation.com

Brent Burley, Practical Hash-based Owen Scrambling, *Journal of Computer Graphics Techniques (JCGT)*, vol. 9, no. 4, 1–20, 2020
http://jcgt.org/published/0009/04/01/

Received:	2020-04-02		
Recommended:	2020-11-11	Corresponding Editor:	Matt Pharr
Published:	2020-12-29	Editor-in-Chief:	Marc Olano